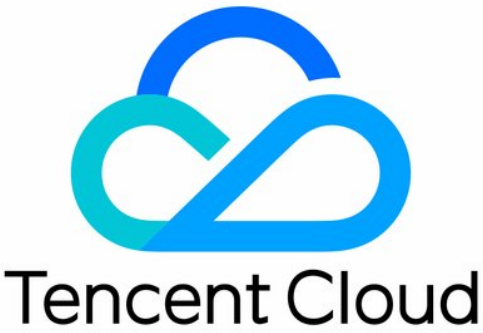
Video on Demand

Player SDK

Product Documentation



Contents

Player SDK
- Overview
- Basic Concepts
- Features
- Free Demo
- Free Trial License
- Purchase Guide
  - Billing Overview
  - Refund Policies
  - Overdue and Suspension Policy
- SDK Download
  - SDK Download
  - Release Notes(Web)
  - Release Notes(iOS & Android)
  - Release Notes(Flutter)
- Licenses
  - Adding and Renewing a License
  - Configuring and Viewing a License
- Player Guide
  - Stage 1. Play back a source video
  - Stage 2. Play back a transcoded video
  - Stage 3. Play back an adaptive bitrate streaming video
  - Stage 4. Play back an encrypted video
  - Stage 5. Play back a long video
- Integration (UI Included)
  - Web Integration
    - TCPlayer Integration Guide
    - TCPlayer Resolution Configuration Guide
    - TCPlayer Swift Live Streaming Downgrade Notice
  - iOS Integration Guide
  - Android Integration Guide
  - Flutter Integration Guide
- Integration (No UI)
  - Web Integration
    - TCPlayer Integration Guide

- TCPlayer Resolution Configuration Guide
- TCPlayer Swift Live Streaming Downgrade Notice
- iOS Integration
  - Integration Guide
  - VOD Scenario
- Android Integration
  - Integration Guide
  - VOD Scenario
- Flutter Integration
  - Integration Guide
  - VOD Scenario
  - Live Streaming Scenario
- Advanced Features
  - Web Advanced Features
    - Security Check Plugin (TCPlayerSafeCheckPlugin)
    - VR Playback Plugin (TCPlayerVRPlugin)
  - Advanced Mobile Features
    - Picture-in-Picture Component (TUIPIP)
      - iOS
    - TUIPlayerShortVideo
      - iOS
      - Android
    - VR Playback Plugin
- API Documentation
  - Web
  - iOS
    - TXVodPlayer
    - TXVodPlayListener
    - Player Config
      - TXPlayerGlobalSetting
      - TXVodPlayConfig
    - Media Download
      - TXVodPreloadManagerDelegate
      - TXVodDownloadDataSource
      - TXVodDownloadMediaInfo
      - TXVodDownloadDelegate
      - TXVodDownloadManager
      - TXVodPreloadManagerDelegate

- TXVodPreloadManager
- Type Definition
  - TXVodSDKEventDef
  - TXPlayerSubtitleRenderModel
  - TXTrackInfo
  - TXVodDef
  - TXBitrateItem
  - TXPlayerAuthParams
  - TXImageSprite
  - TXPlayerDrmBuilder
- Android
  - TXVodPlayer
  - ITXVodPlayListener
  - ITXVodSubtitleDataListener
  - Player Config
    - TXPlayerGlobalSetting
    - TXVodPlayConfig
  - Media Download
    - TXVodDownloadManager
    - ITXVodDownloadListener
    - TXVodDownloadDataSource
    - TXVodDownloadMediaInfo
    - TXVodPreloadManager
    - ITXVodPreloadListener
    - ITXVodFilePreloadListener
  - Type Definition
    - TXVodConstants
    - TXPlayInfoParams
    - TXVodDef
    - TXTrackInfo
    - TXSubtitleRenderModel
    - TXBitrateItem
    - TXPlayerDrmBuilder
    - TXImageSprite
- Flutter
- Player Adapter
  - Player Adapter for iOS
  - Player Adapter for Android

Player Adapter for Web
Player SDK Policy
Privacy Policy

Player SDK
Overview

Last updated：2024-10-17 22:24:43

Product Description

Player SDK is a comprehensive, stable, and smooth video playback service provided for VOD businesses. It helps you connect to Tencent Cloud services and build cloud-based integrated capabilities. The Player and Player Adapter are offered for VOD playback scenarios and support various platforms including web, iOS, Android, and Flutter. In addition, it comes with a variety of video playback solutions to meet your diverse requirements in different business scenarios.

Quick Introduction

To help you quickly understand the Player SDK, we recommend you first read Concepts of Player and quickly select the player type suitable for your business as instructed in How to Select Player before using Player and Player Adapter.

Core Strengths

Cloud-based integration

Player integrates powerful VOD audio/video service capabilities and provides comprehensive cloud-based integration features to offer your business more capabilities.

All-around video security

Player supports video security solutions such as hotlink protection, URL authentication, HLS encryption, private protocol encryption, and offline download. It also features video security capabilities such as dynamic watermarking to help protect the security of your media assets in different scenarios.

Comprehensive data support

Player supports monitoring the video playback quality over the entire linkage through metrics such as playback performance, user behaviors, and file characteristics.

Ultimate playback experience

Player relies on Tencent Cloud's high numbers of cache nodes to provide complete video delivery acceleration capabilities with millisecond-level latency, delivering an ultrafast video playback experience.

Diverse playback capabilities

Player provides various features such as instant broadcasting of the first frame, buffering during playback, adjustable-speed playback, video timestamping, on-screen commenting, and addon subtitles.

Use Cases

Short video playback

By integrating various VOD features, including content moderation, media asset management, seamless switch, instant broadcasting of the first frame, and interactive floating window, Player is often used in UGSV application development. For more information, see Demo.

Long video playback

Player integrates VOD features such as adaptive bitrate streaming, seamless definition switch, thumbnail generation, screencapturing, and adjustable-speed playback. It can be used for playback of long videos such as TV series on video platforms as well as portal development. For more information, see Overview.

Video copyright protection

Player supports video security capabilities of VOD, including private protocol encryption, offline download, scrolling text, and hotlink protection, to help you protect your video security. For more information, see Overview.

Live recording

Player supports live recording playback, time shifting during live streaming, and pseudo-live streaming to help you deliver an integrated viewing experience in audio/video live and VOD playback scenarios.

SDK Download

To try out the Player demo, see Free Demo.
To download the applicable SDK for integration, see SDK Download.
To learn more about the features and capabilities of Player, see Feature Description.

Integration Guide

To help you quickly integrate Player, we provide a Player integration guide to describe the integration steps using demos.
If you encounter any playback issues, see FAQs.

More Media SDKs

In addition to Player SDK, we also provide User Generated Short Video (UGSV) , Player and Effect SDKs to quickly meet different applications requirement. You can choose the appropriate SDK you need. If you need both MLVB, UGSV, Player functions, you can also choose the All-In-One SDK.
Player SDK
MLVB SDK
UGSV SDK
TRTC SDK
All-In-One SDK

Basic Concepts

Last updated：2023-06-16 10:59:52

This document describes the basic concepts involved in the VOD playback scenarios of Tencent Player to help you quickly understand and use its VOD capabilities.

Concepts

Player can be connected to Tencent Cloud VOD through Player or Player Adapter.

Player

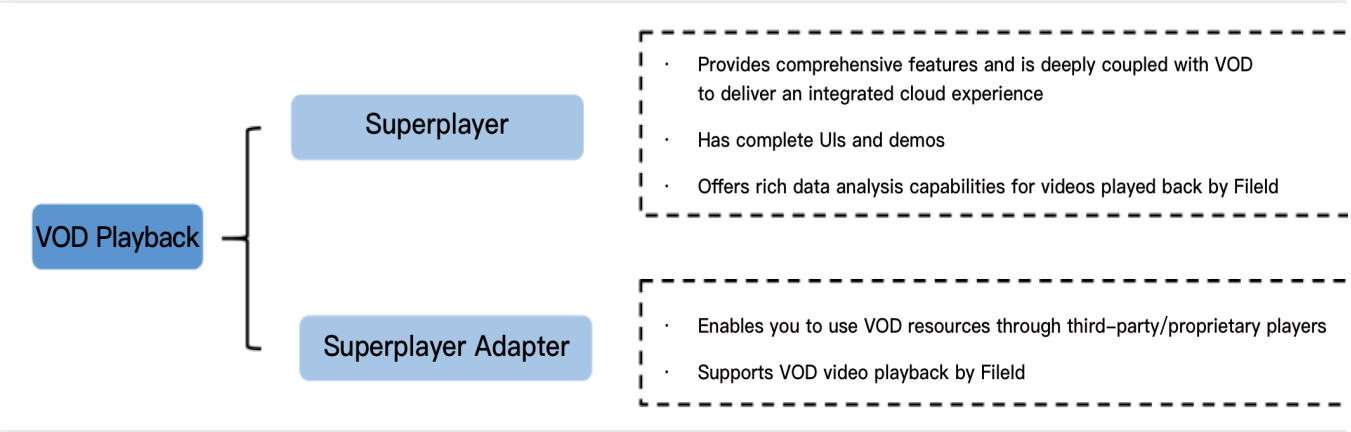
Player is a standalone and complete video player with a full range of video playback features, such as video encryption, thumbnail preview, and definition switch. It comes with complete UIs and demos, is deeply integrated with VOD, and can play back resources in VOD through their `FileID`. In addition, it provides a full-linkage VOD playback quality data service. To quickly connect Player, see Stage 1. Play back a video with player.

Player Adapter

Player Adapter is a player plugin used to connect a third-party player with resources in VOD. It offers diverse capabilities, including video playback and encryption, so you can integrate it into your third-party players to play back resources in VOD through their `FileID`. To quickly connect Player Adapter, see the integration documents for different platforms.

How to Select Player

To simplify connection and match your business scenarios, we recommend you select the most suitable player type when connecting the player service.



Player: Recommended for users who haven't integrated a player but need to quickly build VOD playback capabilities. Player is fully featured and easy to connect. VOD provides a detailed guide on how to connect it.

Player Adapter: Recommended for users who need to play back resources in VOD with a proprietary or third-party player. VOD offers Player Adapter to help you smoothly connect and use resources in VOD.

Platforms supported by Player include:

Player Type	Player	Player Adapter
Web	✓	✓
iOS	✓	✓
Android	✓	✓
Flutter	✓	-
UI	✓	-
Demo	✓	-

Features

Last updated：2024-06-28 17:41:57

The Player SDK provides video playback capabilities for live streaming and video-on-demand (VOD) on platforms such as web/HTML5, iOS, Android, and Flutter. Supported features are detailed below.

Feature Module	Functional Item	Overview	Web	iOS & Android	Flutter
Playback protocol/format	VOD/live streaming	Supports both VOD and live streaming	✓	✓	✓
	Live streaming formats	Supports formats such as RTMP, FLV, HLS, DASH, and WebRTC	WebRTC, FLV,HLS, DASH	RTMP, FLV,HLS, WebRTC	RTMP, FLV,HLS,WebRTC
	VOD formats	Supports audio and video formats such as HLS, DASH, MP4, and MP3	HLS, MP4, MP3,FLV, DASH	MP4, MP3, HLS, DASH (DASH is only supported in the premium version)	MP4, MP3,HLS
	URL playback	Supports URL playback for online videos, where the URL can be used for either VOD or live streaming	✓	✓	✓
	File ID playback	Supports video playback through VOD file identification (FileID), including videos of multiple resolutions,	✓	✓	✓

	thumbnails, markers, and other information			
Local video playback	Supports playback of videos stored locally	-	✓	✓
Live Event Broadcasting	Supports Tencent Cloud Live Event Broadcasting playback with millisecond-level ultra-low latency	✓	✓	✓
DASH protocol	Supports DASH video playback with standard protocols	✓	✓ (supported only in the premium version)	×
Panoramic VR video	Supports playback of panoramic VR video sources. Mobile devices allow finger dragging or gyroscope operations to view panoramic video content, while PC devices support dragging with a mouse to animate and view the interface	✓ (supported only in the premium version)	×	×
QUIC acceleration	Supports the QUIC transport	-	✓	✓

	protocol, effectively improving video transmission efficiency		(supported only in the premium version)	
SDR/HDR video playback	Supports playback of SDR videos and HDR videos in HDR 10/HLG standards	-	✓	✓
H.264 video playback and software and hardware decoding	Supports playback of H.264 video sources, including software and hardware decoding	✓	✓	✓
H.265 video hardware decoding	Supports hardware decoding playback of H.265 video sources	-	✓	✓
AV1	Supports playback of videos encoded in AV1 format	Partially supported	Partially supported (supported only in the premium version)	×
Audio playback	Supports playback of pure audio files such as MP3	✓	✓	✓
Dual-channel audio	Supports playback of dual-channel audio	×	✓	✓

	Multiple audio tracks	Supports playback of video files with multiple audio tracks, allowing for switching between tracks, such as from English to Chinese	✓	✓ (supported only in the premium version)	×
	Http Header Setting	When requesting video resources, you can customize HTTP Header content	×	✓	✓
	HTTPS	Supports playback of video resources over HTTPS	✓	✓	✓
	HTTP 2.0	Supports the HTTP 2.0 protocol	✓	✓	✓
Playback performance	Short Video Playback Component	Achieve ultra-fast first frame, seamless start playback, and silky-smooth short video playback experience at an extremely low cost of integration. By combining pre-play, pre-download, player reuse, precise traffic control, and loading	-	✓ (supported only in the premium version)	×

		strategies, this ensures an extremely smooth playback effect with low energy consumption			
	Pre-downloading	Supports pre-downloading of specified video file content, and configuration of specified size and resolution for pre-downloaded video files.	✓	✓	✓
	Streaming with caching	Supports content caching during playback to reduce network usage. A caching policy can be set	✓	✓	✓
	Exact seek	Supports jumping to a specific point for playback on the progress bar. Mobile playback supports frame-level accuracy, while Web playback offers millisecond-level precision	✓	✓	✓
	Adaptive bitrate (ABR)	Supports ABR streaming of HLS, DASH, and WebRTC,	✓	✓ (Only the premium version	✓ (DASH is not supported)

		which can automatically select the appropriate bitrate for playback based on network bandwidth		supports DASH)	
	Real-time download speed	Provides real-time download speed monitoring, allowing display to end-users during buffering as needed. It is also crucial for using the ABR bandwidth prediction module	✓	✓	✓
	Multiple instances	Supports adding multiple players for simultaneous playback on a single interface	✓	✓	✓
	Dynamic frame rate adjustment	When lag occurs, a "fast forward" method can be employed to automatically catch up to maintain real-time quality during live streaming	✓	×	×
	PDT Seek	Jump to a specific PDT in the video stream to	×	✓ (supported only in the	✓

		enable fast-forward, rewind, and progress bar jumps		premium version)	
Playback control	Basic controls	Supports start, end, pause, and resume playback control features	✓	✓	✓
	Basic picture-in-picture component	Supports switching to picture-in-picture for playback in a small window, with mobile support for picture-in-picture playback both within and outside the integrated App.	✓	✓	✓
	Advanced picture-in-picture component	Compared to basic picture-in-picture, it adds support for encrypted video picture-in-picture, offline playback picture-in-picture, and the "instant switch" effect	-	✓ (supported only in the premium version)	×
	Seek within caching	Supports not clearing Caching video during seek and allows for fast seek	✓	✓	✓

	Live streaming time shifting	Supports live streaming time shifting playback, with settings of start, end, and current positions and a draggable progress bar	✓	×	×
	Progress bar marking and thumbnail preview	Supports adding marker information on the progress bar and supports Sprite Thumbnail previews	✓	✓	✓
	Cover	Supports setting the video cover for playback	✓	✓	✓
	Replay, loop playback, list playback	Supports automatic or manual replay after video playback ends; also supports playing videos in a list sequentially, and even looping them by automatically playing the first video again after the last one finishes.	✓	✓	✓
	Breakpoint resume	Supports playback from the last stop position	✓	✓	✓

Custom playback start time	Supports custom video start time	✓	✓	✓
Playback speed change	Supports 0.5–3x speed change for playback, with pitch-preserving audio speed change	✓	✓	✓
Background playback	Supports continued audio and video playback when the interface is switched to the background	-	✓	✓
Playback callback	Supports callback for playback status, first frame, completion, or failure of playback	✓	✓	✓
Retry upon playback failure	Automatic retry upon playback failure, includes auto-reconnection during live streaming	✓	✓	✓
Volume settings	Supports real-time adjustment of system volume and mute operations	✓	✓	✓
Resolution switching and	Supports seamless and	✓	✓	✓

	naming	buffer-free switching of HLS video streams at various resolutions, and allows for custom naming of different resolution streams			
	Screen-capturing	Supports intercepting any frame of the playback screen	-	✓	×
	Preview	Supports playing videos with the preview feature enabled	✓	✓	×
	On-screen comments	Supports displaying on-screen comments for the video	✓	✓	×
	External subtitles	Supports importing custom subtitle files; Web version supports WebVTT format, while mobile version supports VTT and SRT formats	✓	✓ (supported only in the premium version)	x
	SEI Callback	Parse SEI frames in the video stream	×	✓ (supported only in the	×

		and perform event callbacks		premium version)	
	HEVC Degraded Playback	The player supports the simultaneous input of HEVC and other video encoding formats such as: H.264 playback links. When the playback device does not support the HEVC format, it will automatically degrade to the video playback of the other configured encoding formats (such as: H.264)	×	✓ (supported only in the premium version)	×
	Volume Equalization	Automatically adjust the volume during audio playback to ensure a consistent volume level across all audio tracks	×	✓ (supported only in the premium version)	×
Video security	Referer blocklist/allowlist	Supports identifying the request source through the Referer field in the playback request and controlling source requests	✓	✓	✓

	through the blocklist or allowlist			
Key hotlink protection	Supports adding control parameters in the playback link to manage link validity, preview duration, and number of IPs allowed for playback	✓	✓	✓
HLS encryption	Supports AES encryption provided through HLS by using a key to encrypt video data	✓	✓	✓
HLS private encryption	Supports encrypting videos in Cloud VOD's private protocol, which can only be decrypted and played through the Player SDK, effectively preventing the cracking by various browser plugins and gray-market tools	✓	✓	✓
Commercial-grade DRM	Provides native encryption solutions like Apple FairPlay	✓	✓ (supported only in the premium version)	×

		and Google Widevine			
	Secure download	Supports the decryption and playback of offline downloaded encrypted videos only through the Player SDK	-	✓	✓
	Dynamic watermarks	Supports adding dynamic text watermarks to the playback interface to effectively prevent piracy	✓	✓	×
	Ghost watermarks	Randomly appear on the playback interface for a short time at random positions, and the video playback will automatically pause if an exceptional removal of the watermark is detected. This ensures video security with minimal impact on viewing experience.	✓	×	×
	Web security plugins	Checks the Web playback environment and status and	✓ (supported only in the	-	-

		pauses exceptional video playback to protect video security. The plugins cover MSE environment detection, security architecture check, and interface response integrity verification.	premium version)		
Display effect	Custom UI	The SDK offers an integrated solution with UI, which provides UI-included common playback components that can be selected as needed	✓	✓	✓
	Screen filling	Supports choosing different fill patterns to fit the video screen size	✓	✓	×
	Player size settings	Supports customizing the player size	✓	✓	✓
	Image stickers	Supports adding image stickers for advertising during playback pause	✓	✓	×

	Video mirroring	Supports mirroring in horizontal, vertical, and other directions	✓	✓	×
	Video rotation	Supports rotating video images by angle and automatically rotating videos based on the rotate parameter in the video file	×	✓	×
	Screen locking	Supports the screen locking feature, including locking rotation and hiding interface elements	-	✓	×
	Brightness adjustment	Supports adjusting system brightness during video playback	-	✓	✓

Note:

"-" means the feature is not supported or the concept doesn't exist on the platform.

"✓" without "supported only in the premium version" means it is supported in the basic version.

Free Demo

Last updated：2024-11-30 16:43:31

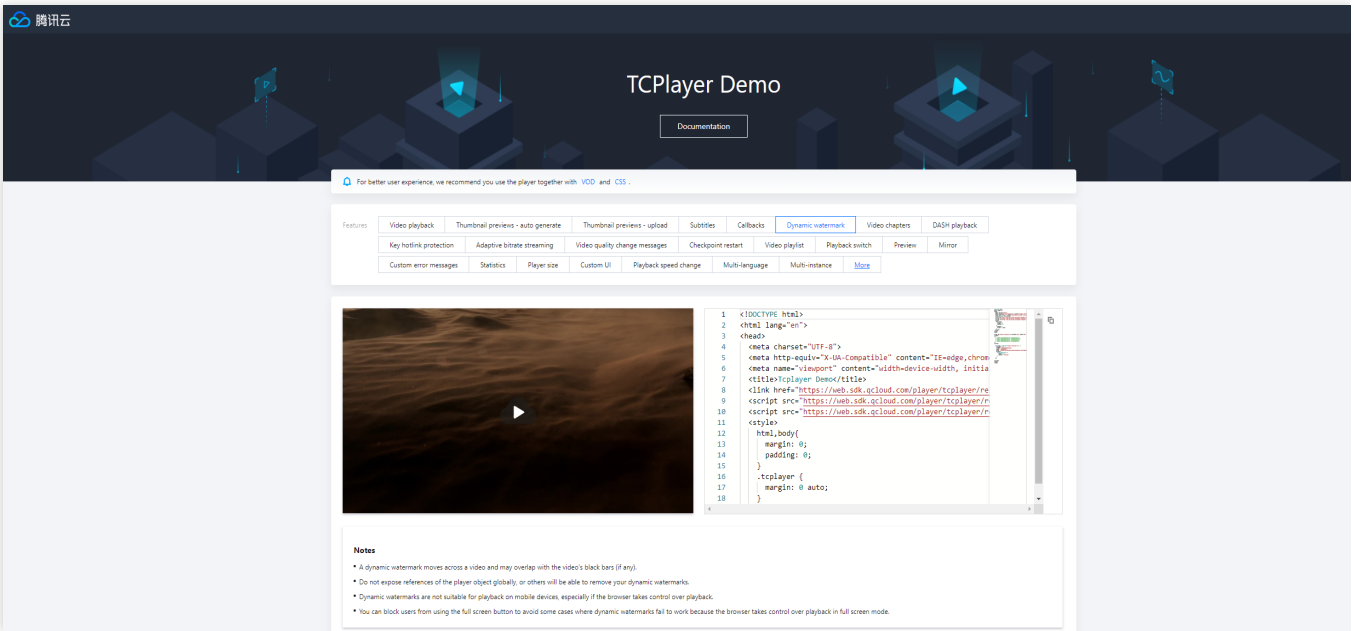
The Player SDK demo provides complete product-grade interactive UIs and business source code for you to use on demand.

Demo

Demos

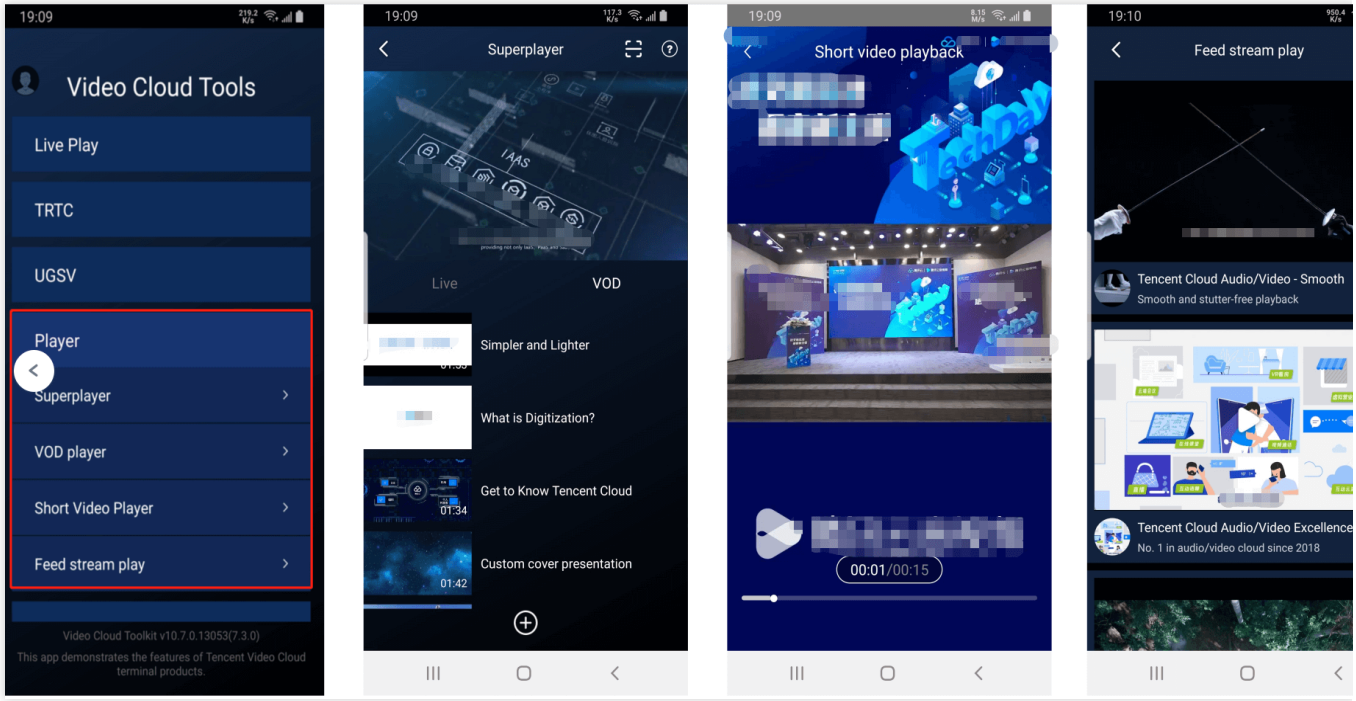
Player for web (TCPlayer)

TCPlayer supports video playback in a browser on a PC or mobile device. It provides demo pages for you to compare and view the video playback features and code. You can modify the code samples to view the effect of modified features in the playback area in real time.



Develop and Debug Demo

To help developers better understand how to use the Player SDK, the Player SDK mobile version offers a demo source code for development and debugging, along with instructions on interfacing. You can follow the steps below for usage.



Step 1: Access the Demo project source code

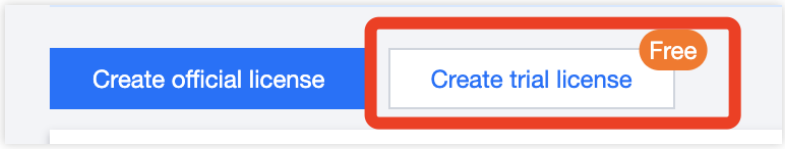
You can visit the following GitHub address to access the demo source code for debugging, or download the corresponding ZIP package.

Platform	Source code address	Download ZIP package
iOS	GitHub	ZIP package
Android	GitHub	ZIP package
Flutter	GitHub	-

Step 2. Configure the license

The Player SDK for mobile (iOS & Android & Flutter) requires access to a License for use.

1. Sign in to the VOD console or CSS console, select **License Management** > **Mobile License** on the left menu, and click **Create Trial License**.



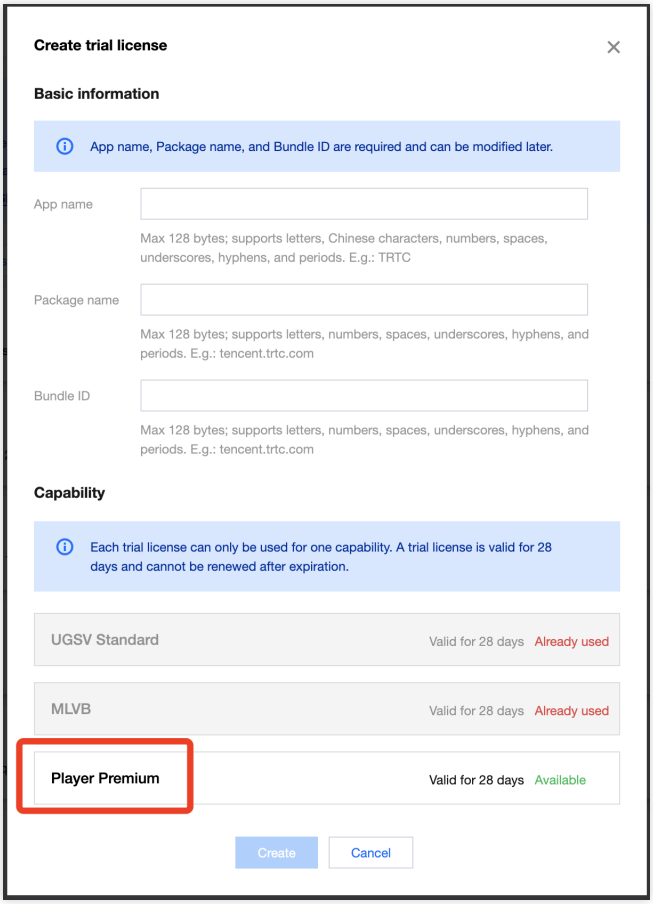
2. Fill in the `App Name`, `Package Name`, and `Bundle ID` according to your actual needs, check the feature module **Player Premium Version**, and click **Confirm**.

Package Name: Please peek at the **build.gradle** file in the App directory for the **applicationId**.

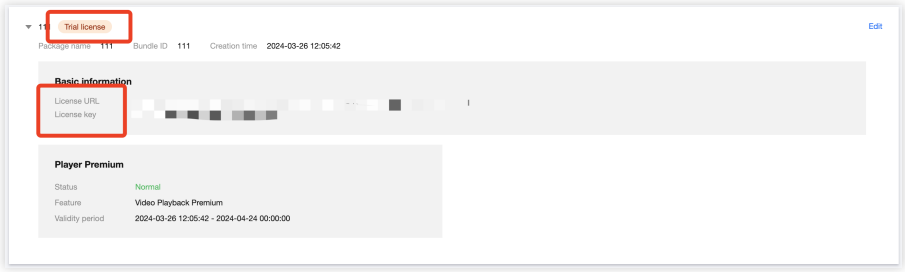
Bundle ID: Please peek at the **xcode** for the project's **Bundle Identifier**.

Note:

If the Package Name or Bundle ID applied for in the Tencent Cloud Console is inconsistent with the actual Package Name or Bundle ID in the project, playback will fail.



3. After the trial version License is successfully created, the page will display the generated License information. **Upon SDK initialization, the License URL and License Key are required; please save the following information carefully.**



4. After accessing the Licence URL and Licence Key, please refer to the tutorial below to configure them in the Demo project.

Android configuration for Licence

iOS configuration for Licence

Flutter configuration for Licence

Open the Demo/app/src/main/java/com/tencent/liteav/demo/TXCSDKService.java file, and replace the Licence URL and License Key with the applied Licence content.

Open the Demo/TXLiteAVDemo/App/config/Player.plist file, and replace the Licence URL and Licence Key with the applied Licence content.

Open the Flutter/example/lib/main.dart file, and replace the Licence URL and Licence Key with the applied Licence content.


main.dart
}
/// set player license
Future<void> initPlayerLicense() async{
String licenceURL = ""; // 获取到的 licence url
String licenceKey = ""; // 获取到的 licence key
await SuperPlayerPlugin.setGlobalLicense(licenceURL, licenceKey);
}

Free Trial License

Last updated：2024-11-11 15:08:18

The Player SDK offers a trial License, including **Mobile** trial License and **Web** trial License. You can refer to this document for guidance and apply for a free trial as needed.

Trial License Types	Features that can be authorized	Validity Period
Mobile trial License	Player SDK **Mobile** all features (including premium features)	By default, 28 days after application
Web trial License	Player SDK **Web** all features (including premium features)	By default, 28 days after application

Note:

After purchase, you can add and renew the player License via the VOD console or CSS console. For details, please see Adding and Renewing License.

Mobile trial License

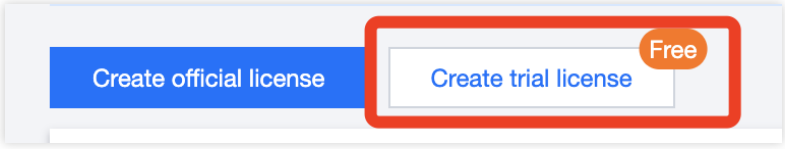
Trial License Application

You can apply for a free trial of the Player Mobile License (valid for total of 28 days) for testing. When applying for the test module, you can choose to **create a new trial License and select Player trial License** or **apply for a new License in an existing test application**.

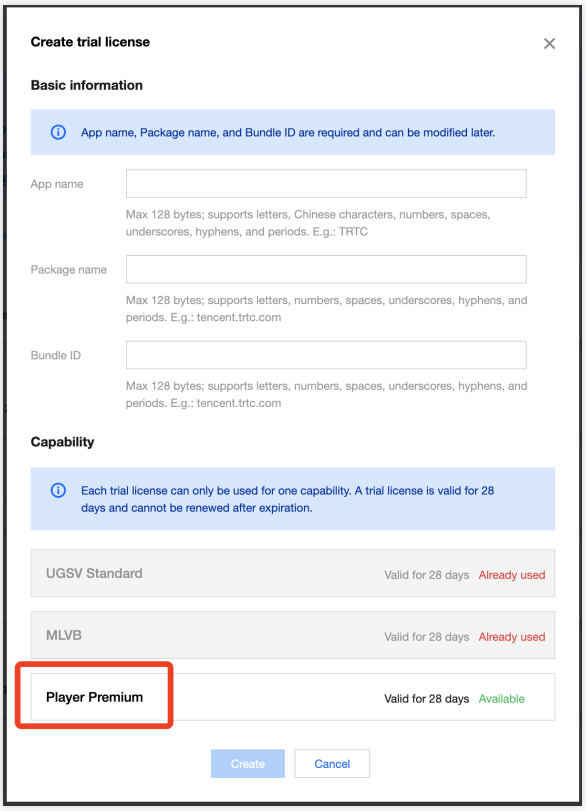
Method 1: Create a new trial License and select Player trial License

Method 2: Apply for a New Trial License within an Already Created Test Application

1. Sign in VOD console **or** CSS console**> License Management > Mobile Client License**, click **Create Trial License**.



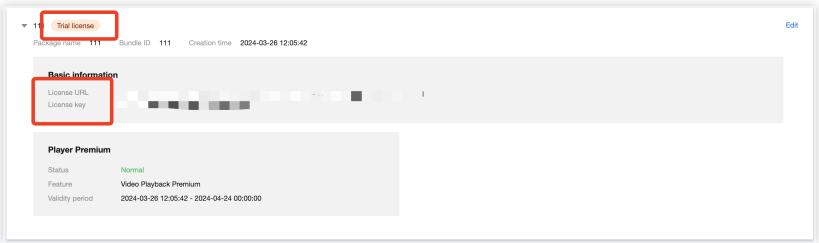
2. Fill in the `App Name` , `Package Name` , and `Bundle ID` according to actual needs, select **Player Premium** , and click **Create**.



3. After the trial License is successfully created, the page will display the generated License information. **When initializing the SDK configuration, you need to input both the Key and License URL. Please save the following information accordingly.**

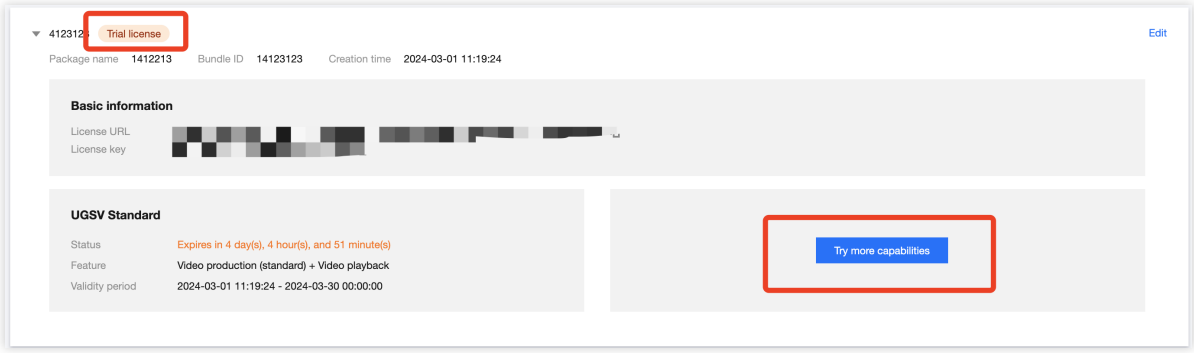
Note:

For the same application, the License URL and Key are unique. When the trial License upgrades to an official version, the License URL and Key remain unchanged.

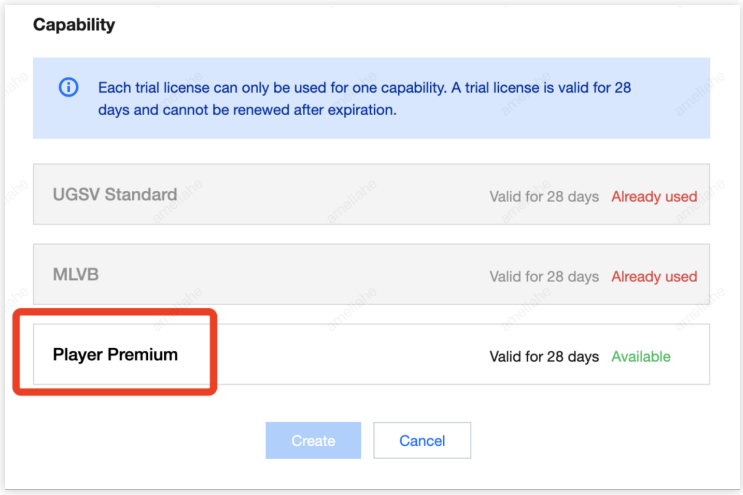


If you wish to apply for a trial of the **Player** feature within an already created test application, follow these steps:

1. Sign in to VOD console **or** CSS console**> License Management > Mobile License**, select the application you wish to test, and click **Test New Feature**.



2. Select **Player Premium**, and click **OK**.



Note

During the trial License's validity period, you can click on the **Edit** button on the right, enter to modify the Bundle ID and Package Name information, and click **OK** to save.

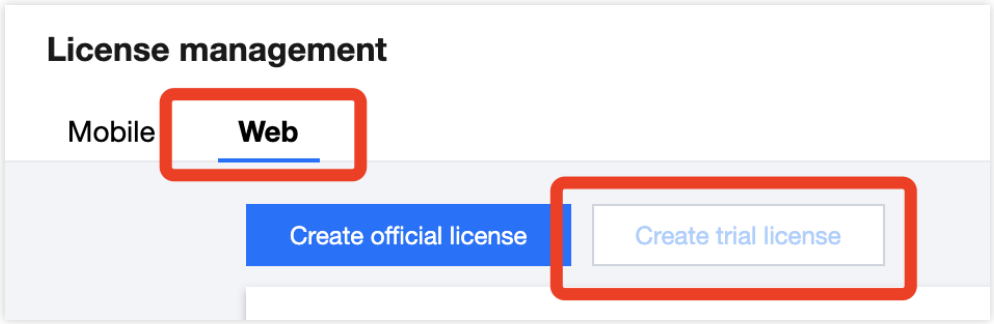
If there is no Package Name or Bundle ID, you may fill in "-".

Web Trial License

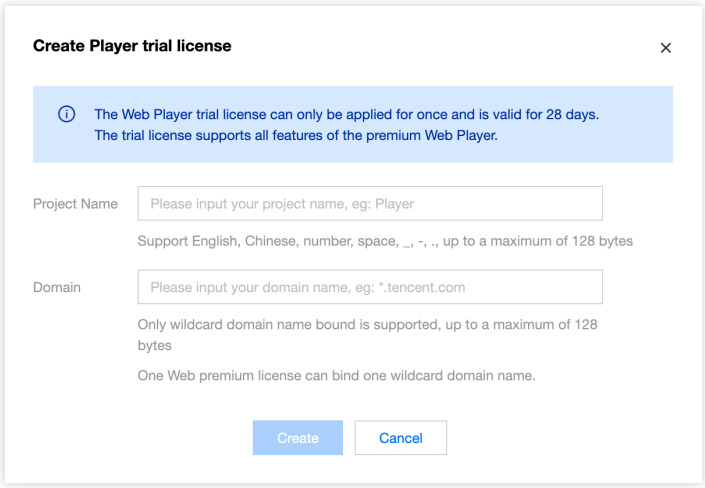
Trial License Application

You can apply for a free Player Web Client trial License (valid for 28 days) for a test experience.

1. Sign in to VOD console **or** CSS console**> License Management >,** click **Create Trial License**.



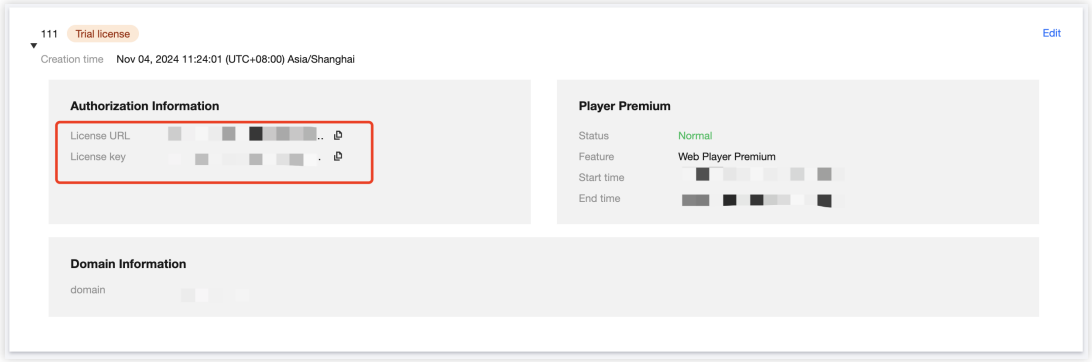
2. Fill in the **Project Name** and **Domain**, then click **Create** to complete the application.



Note:

Only version 5.1.0 and above of the Web Player SDKsupport the use of wildcard domain authorization.

3. After the trial License is successfully created, the page will display the generated License information. **When initializing the SDK configuration, you need to input both the Key and License URL. Please save the following information accordingly.**



Note:

A single Web Premium Version License can only be associated with one wildcard domain.

You can click the **Edit** on the right side to modify the domain name.

Purchase Guide
Billing Overview

Last updated：2024-04-11 16:11:38

The cost of the Player SDK includes the following:

Type of Expense	Description
SDK authorization fees	The use of authorization fees for Player SDK.
Other related cloud service fees	When using the Player SDK in conjunction with cloud services like VOD and CSS, corresponding fees are generated.

Note:
The Player SDK offers a cumulative 28-day trial License for both the Mobile Premium Version and the Web Premium Version. You can apply for free in the console.

SDK Authorization Fees

The Player SDK authorizations for Mobile and Web are separate. Both platforms offer Basic and Premium versions, and you can obtain the usage authorization for the corresponding version by purchasing the designated License. For differences between the platforms and the versions, see Product Features.

Acquisition Method

Player SDK Platform	Feature Version	Required License Type	Price	Authorization Unit
Mobile Player (iOS & Android & Flutter)	Basic version	Player Mobile Basic Version License /UGSV License /MLVB License	0 USD (1 year, renewable for free) Apply for free	One license can authorize one iOS application Bundle ID and one Android application Package Name
	Premium version	Player Mobile Premium Version License	499 USD/month Purchase Now	
Web Player	Basic version	Player Web Basic Version License	0 USD (1 year, renewable for free) Apply for free	Precise domain (one License can authorize up to ten precise domains)
	Premium	Player Web Premium	99 USD/month	Wildcard Domain

	version	Version License	Purchase Now	(one License can authorize up to one wildcard domain)

If you have already purchased one of the following three Licenses, you can also obtain the Player Mobile SDK authorization with the Basic version without needing to purchase an additional player License.

Official License Type	Validity Period	Price (USD/Year)
UGSV Lite Version License	1 Year (From the Date of Purchase)	1,899
UGSV Basic Version License	1 Year (From the Date of Purchase)	9,999
MLVB License	1 Year (From the Date of Purchase)	5,988

Billing

Each account can apply for one free mobile player trial license and one web player trial license via the VOD console or CSS console to experience and test the product. The first application provides a 14-day validity period, which can be renewed in the console for another 14 days, totaling 28 days. Mobile and web trial licenses have separate application counts and durations.

A mobile license can be associated with one iOS application Bundle ID and one Android application Package Name, while a web license can be associated with up to ten precise domains. Associating a license with an application/domain provides the corresponding authorization without distinction between business environments. If multiple applications need to be integrated, the corresponding number of licenses must be purchased for binding.

Domain Name Description:

A precise domain refers to a specific fixed domain address, such as: `a.com` , `a.b.com` , `a.b.c.com` and other unique and fixed domain addresses.

A wildcard domain refers to a series of domain addresses with the same domain suffix, such as a wildcard domain `*.a.com` associated with a license. It allows `a.a.com` , `b.a.com` , `c.a.com` and other domains with the same suffix to unlock package capabilities. The `*` position can support custom definitions of multi-level domains, such as `b.c.a.com` and `b.c.d.a.com` .

An application supports changing the associated license. The authorization validity period for changing the application is the validity period of the newly associated license. The replaced license can be used to associate with other applications, and its validity period remains unchanged.

For refund-related content, see Refund Explanation.

New License Validity Period Example

A customer purchased the Player Mobile Premium Version License A on July 1, 2022, at 11:36:59, and associated it with iOS application A1 and Android application A2. Later, on July 2, 2022, at 11:36:59, the customer purchased

another Player Mobile Premium Version License, obtained its License B, and associated it with iOS application B1 and Android application B2. Therefore:
The validity period of Player Mobile Premium Version License A is from July 1, 2022, 11:36:59 to August 2, 2022, 00:00:00.
The validity period of Player Mobile Premium Version License B is from July 2, 2022, 11:36:59 to August 3, 2022, 00:00:00.
Applications A1, A2, B1, and B2 all receive authorization to use features of the Player SDK Mobile Premium Version. Meanwhile, the authorization validity period for applications A1 and A2 is based on the validity of License A, and for applications B1 and B2, it is based on the validity of License B.

Validity Period Update Example

Based on the previous example, the same customer purchased the Player Mobile Premium Version License C on December 1, 2022, 15:48:12, and used License C to renew applications A1 and A2. As a result:
The validity period of Player Mobile Premium Version License C is from August 1, 2022, 15:48:12 to September 2, 2022, 00:00:00. When renewing for applications A1 and A2, License C will replace the originally affiliated License A with applications A1 and A2.
Since an application's authorization validity period is based on the actual affiliated License's validity period, the renewal changes the validity period of applications A1 and A2 to that of License C, which is from August 1, 2022, 15:48:12 to September 2, 2022, 00:00:00. Thus, the authorization validity period of applications A1 and A2 is extended from the original August 2, 2022, 00:00:00 to September 2, 2022, 00:00:00.
After License C becomes affiliated with applications A1 and A2, the originally affiliated License A with applications A1 and A2 will naturally disaffiliate, while the validity period of License A remains unchanged, still serving as an available License resource that can be affiliated with other applications.

Other Related Cloud Services Fee

In addition to license fees, using the Player SDK may also incur the following service fees. Related fees will not be generated if the related services are not used.

Video on Demand (VOD)

Tencent Cloud's VOD service provides functions such as video upload, storage, transcoding, acceleration distribution playback, copyright protection, and play quality monitoring.
Billing of VOD:
Storage fees are charged based on the storage space used by files uploaded to VOD and their transcoding outputs.

If you transcode files stored in VOD, transcoding fees are charged based on the specifications and durations of the outputs.
If you use VOD's acceleration service to deliver videos, acceleration fees will be charged based on the traffic consumed for playback.

Note

For more information about the billing of VOD, see Billing Overview.

Cloud Streaming Services (CSS)

CSS offers capabilities including live stream receiving, on-cloud recording, live transcoding, live screencapture, and live stream delivery and playback. If you need to use related functions, we recommend you use CSS.
Using CSS to receive and deliver live streams will incur basic service fees, which are charged based on the traffic/bandwidth consumed.
Using CSS features such as live transcoding (including stream mixing and watermarking), live recording, live screencapture, RTC-based co-anchoring, and relay to CDN will incur value-added service fees.

Note

For more information about the billing of CSS, see Pricing Overview.

Refund Policies

Last updated：2024-04-11 16:11:38

Five-Day Unconditional Refund

To facilitate the use of the Player SDK, if your purchased License meets the following conditions, Tencent Cloud supports an unconditional refund within 5 days:
Mobile License not bound to an application package name (including both new additions and renewals binding methods), and Web License not bound to a domain.
The purchase date of the License is no more than 5 days ago (including the 5th day).

Refund Policy

Orders that meet the Five-Day Unconditional Refund policy can submit a refund request via ticket, see Refund Steps for details.
If there is suspected abnormal or malicious return, Tencent Cloud reserves the right to reject your return request.
For orders that meet the 5-day unconditional refund policy, the refund amount will be the total payment made at the time of purchase, including the cash account balance, transferred earnings, and gift account balance.
Note
Rebates and vouchers will not be returned.
A full refund will be returned to your Tencent Cloud account.

Refund Steps

1. Click Submit Ticket to go to the ticket submission page.
2. In the search box on the right, search for and select "**Other Tencent Cloud Products**".
3. Select the problem type as "**Feature Consultation**".
4. Go to the ticket creation page, fill in the relevant information, and click **Submit Ticket** to complete.

Note
Before returning the resources, please check if they meet the conditions for a 5-day unconditional refund. If they do not, a refund cannot be processed.

Overdue and Suspension Policy

Last updated：2024-04-11 16:18:08

When the Tencent Cloud billing platform detects that your Tencent Cloud account balance is insufficient, it will remind you to top up. You have 24 hours to replenish your account after it falls into arrears. If you do not top up your account after this period, services such as Cloud Streaming Services, Video on Demand, and live co-anchoring will be temporarily suspended. If you top up your account within 24 hours of falling into arrears, your co-anchoring service will not be affected.

SDK Download

SDK Download

Last updated：2024-04-11 16:11:38

Player SDK

Release Notes(Web)

Last updated：2024-04-11 16:11:38

TCplayer update log

TCplayer 5.1.0 @ 2023.11.13

New support for configuration native decoding strategy

MacOS iOS optimize subtitle display effect

Full scene support for loadVideoByID method

Support for parsing wildcard domain names

Added new server-side error codes and license-related error codes

Fixes

TCplayer 5.0.0 @ 2023.08.16 Breaking Change

Adding a License.

New support for VR and security environment check plugins.

Live Event Broadcasting ABR interface adjustment.

Live Event Broadcasting downgrade logic adjustment, added automatic downgrade.

Open M4A.

Optimized pseudo-fullscreen effect.

macOS Safari supports private encryption.

TCplayer 4.8.0 @ 2023.04.20

New support for P2P.

Added a switch for downgrading WebRTC.

Fixes.

TCplayer 4.7.2 @ 2023.1.10

Fixed an issue with incorrect iOS environment detection.

TCplayer 4.7.0 @ 2022.12.20

Added automatic asynchronous loading of dependency SDK.

Adding Element (XML) checks.

New support for ghost watermark.

Improving code aliasing.

TCplayer 4.6.0 @ 2022.11.04

New support for multiple audio tracks.
New support for URL-based resuming playback.
Fixed some issues.

TCplayer 4.5.4 @ 2022.08.26

New support for Live Event Broadcasting ABR, updated txliveplayer.
New support for AV1 in FLV, updated flv.js.
New support for SDMC DRM.
New support for unifying the auto-play blocked events in Live Event Broadcasting and Live Video Broadcasting to 'blocked'.
New support for marker callback events.
Fixed some issues.

TCplayer 4.5.3 @ 2022.06.15

New support for commercial-grade VOD DRM.

TCplayer 4.5.2 @ 2022.04.15

Fixed a vulnerability where videos could be stolen by hijacking MSE to bypass private encryption schemes.
Fixed some data reporting issues.

TCplayer 4.5.0 @ 2022.01.14

New support for data reporting features in Live Video Broadcasting and Live Event Broadcasting.
New support for playing MP3 audio formats.
New support for frame synchronization capabilities under poor network conditions.
Optimized watermark feature, supporting more configuration options.
Fixed several issues.

TCplayer 4.4.0 @ 2021.12.14

New support for playing Live Event Broadcasting.
New data reporting feature for on-demand scenarios.
New support for playing original and transcoded media files via the v4 interface.

TCplayer 4.1 @ 2020.07.10

Updated the default version of hls.js to 0.13.2.
Support for enabling the Key Hotlink Protection feature.
Fixes for other known issues have been implemented.

TCplayer 4.0 @ 2020.06.17

Fix preview video to keep displaying the original duration.
Enable background quality configuration.
Fixes for other known issues have been implemented.

TCplayerLite update log

TCplayerLite 2.4.1 @ 2021.06.25

New support for WebRTC stream addresses with v1 signaling.
Added webrtcConfig parameter.
Added events for WebRTC stutter, stutter end, and stream end.

TCplayerLite 2.4.0 @ 2021.06.03

Added support for the Live Event Broadcasting feature.
Fixes for other known issues have been implemented.

TCplayerLite 2.3.3 @ 2020.07.01

Fixed an issue where switching to fullscreen in the X5 environment causes abnormal event dispatch.
Avoided the issue where switching HLS sources causes related events to trigger slowly, leading to abnormal cover display.

TCplayerLite 2.3.2 @ 2019.08.20

Updated the default HLS version to 0.12.4.
Fixes for other known issues have been implemented.

TCplayerLite 2.3.1 @ 2019.04.26

Added flvConfig parameter.
Load flv.1.5.js by default.
Fixes for other known issues have been implemented.

TCplayerLite 2.3.0 @ 2019.04.19

Added some feature parameter options.
Changed the parameter 'coverpic' to 'poster'.
Destroy the flv.js instance.
Fixes for other known issues have been implemented.

TCplayerLite 2.2.3 @ 2018.12.17

Optimized playback logic.
Resolved the issue where loading animation appears on iOS WeChat without triggering play events.
Fixes for other known issues have been implemented.

TCplayerLite 2.2.2 @ 2018.05.03

Optimized the loading component.
Optimized the Flash destroy method.
Use H5 playback by default.
Fixed known issues.

TCplayerLite 2.2.1 @ 2017.12.20

Added configurable definition text feature.
Set default definition.
Supports method to switch definition.

TCplayerLite 2.2.1 @ 2017.12.07

Added systemFullscreen parameter.
Added flashUrl parameter.
Fixed the UI issue when toggling mute at maximum volume.
Fixed the issue where two clicks are required to play on iOS 11 WeChat.
Fixed the issue where system styles are obscured in Safari 11.
Adapted to the situation where seeking is triggered in the X5 kernel, but seeked is not.
Fixed the problem where dragging the progress bar to the starting position and setting currentTime fails.
Switching definition keeps the volume unchanged.
Fixed the issue where the page width is 0, causing the player width determination to fail.
The destroy method now completely removes the player node.

TCplayerLite 2.2.0 @ 2017.06.30

Added parameters to control the playback environment: Flash, h5_flv, x5_player.
Adjusted player initialization logic to optimize error message effects.
Added support for flv.js, allowing FLV playback with flv.js under the right conditions.
Supports the x5-video-orientation attribute.
Enhanced the logic for determining the playback environment. Parameters can now adjust the priority between H5 and Flash, and whether to enable TBS playback.
Implemented version numbering for releases to avoid impacting users of older versions.
Optimized the timestamp for event triggers, standardizing it to universal time.
Bug fixes.

TCplayerLite 2.1.0 @ 2017.03.04

As of 2017.06.30, after several iterations of development, it has gradually become stable. Currently, unless otherwise specified in the documentation, all feature descriptions are based on this version.

Adapter plugin update log

Adapter plugin released @ 2021.07.16

First release of the player Adapter version

Release Notes(iOS & Android)

Last updated：2025-06-20 15:38:56

Player SDK

Player SDK 12.6 @ 2025.06.18

Function optimization:

Android&iOS: Optimize the playback retry mechanism in weak network environment to improve the playback continuity when the network is unstable.

Bug fixes:

Android&iOS: Fix the application crash caused by the bit rate dropping to 0 in extreme cases.

Android&iOS: Fix the problem of insensitive resolution adjustment when switching adaptive bit rate.

Player SDK 12.5 @ 2025.05.07

Function optimization:

Android&iOS：The standard version supports FileId pre-download of encrypted videos.

Android&iOS：Optimize playback memory usage.

Bug fixes:

iOS：Fixed the playback error issue of picture-in-picture during loop playback and lock screen playback.

Player SDK 12.4 @ 2025.03.24

New features:

Android&iOS: DRM playback supports license headers and encrypted playback in cbcs mode (supported by the premium version).

Android & iOS: Preloading allows setting preferred audio tracks (supported by the premium version).

Function optimization:

Android & iOS: If switching audio tracks fails, it supports rolling back to the previous playback audio track.

Android & iOS: Optimization during the playback process in case of network timeout

Bug fixes:

iOS: Fix subtitle font color incorrect settings issue.

iOS: Fix an issue that the rendering of missing frames at the end of some audio causes incomplete audio playback.

Android: Fix potential memory leak issue after `setPlayerView` is set to null.

Player SDK 12.3 @ 2025.01.17

New features:

Android&iOS: Subtitles support text callback (supported by the premium version).

Android&iOS: Support setting the priority start audio track before starting (supported by the advanced version).
Android&iOS: Fixed the problem of incomplete pre-download of encrypted nested HLS.

Bug fixes:

Android & iOS: Fixed the issue that trackIndex is not called back after multiple subtitles are selected.
Android: Fixed the issue that occasional multi-threaded crashes occurred in the TRTC playback scene.

Player SDK 12.2 @ 2024.11.20

New features:

Android&iOS: When the player pushes the video auxiliary stream to TRTC, it supports pausing and continuing to push video frames.
Android&iOS: Pre-download supports encrypted MP4, and supports offline playback of encrypted MP4 videos.
Android&iOS: Pre-download supports setting httpHeader.

Bug fixes:

Android&iOS: Fixed the problem of playback failure caused by case sensitivity of httpHeader when switching links.
Android&iOS: Fixed the problem of Loading event triggering for too long in network abnormal scenarios.

Function optimization:

Android&iOS: Optimize the startup speed and potential memory leaks.
Android&iOS: Support the playback of videos with poor audio and video interleaving (supported by the premium version).

Player SDK 12.1 @ 2024.09.26

Bug fixes:

Android&iOS: Fixed the issue that the loading time of playback is too long under abnormal network conditions.
Android: Fixed the issue that the HEVC downgrade playback event is thrown before the Prepare event.
iOS: Fixed the issue that the progress callback is abnormal when the video duration is 0.

Player SDK 12.0 @ 2024.08.01

New features:

Android&iOS: When HEVC is downgraded, the corresponding event VOD_PLAY_EVT_HEVC_DOWNGRADE_PLAYBACK (2031) is thrown externally.
Android&iOS: Add the event VOD_PLAY_EVT_VOD_PLAY_FIRST_VIDEO_PACKET (2017) for receiving the first frame packet.

Bug fixes:

Android&iOS: Fixed the cache reuse error caused by the playback link with exper when MP4 preview video switches to the main film playback scene.
Android&iOS: Fixed the abnormal preloading and adaptive playback of HLS encrypted video.
Android&iOS: Fixed the memory leak problem that may occur in high-concurrency playback scenarios.
iOS: Fixed the problem that multiple pre-downloads initiated by fileId may cause crashes.

iOS: Fixed the problem that the app_version field is missing in the playback quality monitoring report.
iOS: Fixed the problem that the error code is lost after selecting the subtitle track.
iOS: Fixed the issue that the subtitle data is lost after restarting the playback after adding subtitles after calling TXVodPlayer#stopPlay.
iOS: Fixed the issue that the playback starts from the beginning after returning to the foreground after going back to the background for a while during playback.
iOS: Fixed the issue that the memory is not released after the pre-download fails.

Player SDK 11.9 @ 2024.06.03

Bug fixes:

Android&iOS: Optimize the first frame time statistics to make the statistics more accurate.
Android&iOS: Fix the crash problem of the player downloader module in multi-threaded scenarios.
Android: Remove the call to `NetworkInfo#getExtraInfo` to avoid being mistakenly detected as reading ssid.

Player SDK 11.8 @ 2024.05.06

New features:

Android&iOS: Support HLS EVENT live source (supported by the advanced version).
Android: Support playing local videos with `content://` and `asset://` URIs.
Android&iOS: Support precise and non-precise Seek.

Bug fixes:

Android: Fix the problem of callback progress after receiving the PLAY_EVT_VOD_PLAY_PREPARE event when the TXVodPlayer#setAutoPlay parameter is false.
iOS: Fix the abnormal problem caused by directly stopPlay without calling exitPictureInPicture during picture-in-picture playback.

Player SDK 11.7 @ 2024.03.04

New features:

Android&iOS: Added playback volume equalization control function (supported by premium version).
Android&iOS: HECV adaptive downgrade playback (supported by premium version).
Android&iOS: Support HLS EVENT live broadcast source (supported by premium version).

Player SDK 11.6 @ 2024.01.10

New features:

Android & iOS: Upgraded network kernel for advanced player version, providing better performance (supported by premium version).
Android & iOS: Preloading supports FileId encrypted videos (supported by premium version).
Android & iOS: FileId video playback supports ghost watermark.

Android & iOS: Supports SEI information callback (supported by premium version).
Android & iOS: Supports real-time acquisition and seek ability of Program Date Time for HLS video format (supported by premium version).

Function optimization:

Android & iOS: First frame event carries additional first frame duration information.
Android & iOS: Fixed built-in subtitle parsing exception for advanced player version.
Android & iOS: No need to set http proxy to bypass localhost when using packet capture tool.

Bug fixes:

Android & iOS: Fixed issue where playback retry count is invalid after network disconnection.
Android & iOS: Fixed slow seek issue for some mp3 files.
Android & iOS: Fixed issue where AES-128 encrypted m3u8 files cannot be played offline.
Android: Fixed issue where only the first seek is effective when seeking multiple times in a short period.
iOS: Fixed issue where built-in subtitle causes abnormal playback in Picture-in-Picture mode.

Player SDK 11.4 @ 2023.08.29

New Features:

iOS: Added advanced picture-in-picture capabilities, supporting picture-in-picture playback of encrypted videos, offline playback of picture-in-picture, and automatic picture-in-picture when switching to the background.

Function Optimizations:

Android & iOS: Cache-related interfaces support KB granularity control.
Android & iOS: Optimized the network scheduling strategy of the player.

Bug Fixes:

Android & iOS: Fixed the issue where network traffic consumption could not be stopped in a timely manner after playback stopped.

Player SDK 11.3 @ 2023.07.07

New Features:

Android&iOS: Video preloading now supports specifying media types (TXPlayInfoParams#mMediaType) to reduce type detection and improve download efficiency.
iOS: Added network exception retry mechanism during playback.

Function Optimizations:

Android&iOS: Optimized the issue of excessive memory allocation during playback in the event of network disconnection.

Bug Fixes:

Android&iOS: Fixed issue where playing HLS videos after network disconnection caused repeated playback of a certain segment.
Android&iOS: Fixed issue where small preloadSize settings caused occasional video playback failures.
iOS: Fixed issue where httpDns service occasionally caused crashes in weak networks.

Player SDK 11.2 @ 2023.06.01

New features:

Android&iOS: Added callback for VOD_PLAY_EVT_HIT_CACHE event during VOD playback.

Android: DRM playback can now be configured with COM or CN certificate providers using TXPlayerGlobalSetting#setDrmProvisionEnv.

Function optimizations:

Android&iOS: Media type can now be specified using TXVodPlayConfig#setMediaType to reduce underlying type detection and improve startup speed.

Android&iOS: Added callback for audio bitrate (VIDEO_BITRATE&AUDIO_BITRATE) during MP4 playback.

Bug fixes:

Android&iOS: Fixed missing VIDEO_CACHE value in onNetStatus callback.

Android&iOS: Fixed playback failure issue for offline downloaded resources in certain scenarios.

Android&iOS: Fixed probability of playback failure during fast video switching.

Player SDK 11.1 @ 2023.04.07

Function optimizations:

Android&iOS: Added interface for checking if downloaded video resources exist (TXVodDownloadMediaInfo#isResourceBroken).

Android&iOS: Video downloading now supports sharing cache for private encrypted videos before the anti-theft link expires.

Bug fixes:

Android&iOS: Fixed issue where CPU usage increased after audio frame decoding failure.

Android&iOS: Fixed issue where progress was lost when the download process exited abnormally.

Android&iOS: Fixed playback issues for some MP4 files.

Player SDK 11.0 @ 2023.02.24

Function optimizations:

Android&iOS: Improved compatibility for video playback when audio and video are not well interleaved.

iOS: Optimized memory release for video frame data.

Bug fixes:

iOS: Fixed abnormal event of not calling back VOD_PLAY_EVT_LOOP_ONCE_COMPLETE (single loop playback completed) when playing in a loop.

Android&iOS: Fixed inaccurate size of sub-stream files obtained through video downloading (TXVodDownloadMediaInfo#getSize).

Android&iOS: Fixed abnormal download issue for nested m3u8 single-stream downloads.

Android&iOS: Fixed probability of playback issues for nested m3u8 private encrypted videos downloaded through video downloading.

Player SDK 10.9 @ 2022.12.30

New features:

Android&iOS: Support external HttpDns to solve the problem of playback failure caused by domain hijacking during playback.

Android&iOS: V2TXLivePlayer now supports WebRTC playback.

Function optimizations:

Android: Added "speed" field to TXVodDownloadMediaInfo for obtaining network download speed in video downloading.

Bug fixes:

iOS: Fixed issue where picture-in-picture switching repeatedly caused switching to fail.

iOS: Fixed issue where interface still showed video downloading as incomplete after download was completed.

iOS: Fixed issue where video playback was stuck on iOS 16.

Android&iOS: Fixed issue where playback failed on some devices due to high video frame rate. Android&iOS: Reduced the time it takes for playback to fail in case of network exceptions.

Player SDK 10.8 @ 2022.10.27

Function optimizations:

Android&iOS: Added VOD_PLAY_EVT_LOOP_ONCE_COMPLETE event for single loop playback completion.

Android: Optimized startup by reducing the number of calls to NetworkInfo.getExtraInfo to comply with regulations.

Bug fixes:

Android&iOS: Fixed issue where private encrypted videos failed to play in certain scenarios.

Android&iOS: Fixed issue where some videos failed to play when transmitted through gzip.

Android&iOS: Fixed issue where progress bar duration did not match actual video duration after playback ended.

iOS: Fixed issue where v2 protocol failed to retrieve video source address for appid&fileid playback.

Player SDK 10.7 @ 2022.09.20

Function optimizations:

Android&iOS: Changed startPlay interface for VOD playback to startVodPlay.

Android&iOS: Changed startPlay interface for live playback to startLivePlay.

iOS: Fixed issue where playback could not be resumed after being in background for a long time.

Android: Fixed issue where some videos failed to play on older Android systems.

Player SDK 10.6 @ 2022.08.31

Function optimizations:

Android&iOS: Added sprite map and URL information callback for fileid playback.

Android&iOS: Optimized package size.

Bug fixes:

iOS: Fixed issue where offline downloaded private encrypted videos failed to play in certain scenarios.

Player SDK 10.5 @ 2022.08.12

Bug fixes:

Android&iOS: Fixed issue where short links without video format suffixes caused playback failure.

Player SDK 10.4.0 @ 2022.07.21

Function optimizations:

Android&iOS: Added support for adaptive playback for HLS live streaming.

Bug fixes:

Android: Fixed abnormal interval between onNetStatus and progress callbacks.

Android: Fixed null pointer exception caused by failure to call setConfig on the player.

iOS: Fixed issue where replaying caused stuttering in certain scenarios.

Player SDK 10.3.0 @ 2022.07.06

New features:

iOS: Added support for picture-in-picture mode during video playback.

Bug fixes:

Android: Fixed issue where continuous playback of video lists in the background using hardware decoding was interrupted.

Android&iOS: Fixed issue where seek completion event was not called back.

Player SDK 10.2.0 @ 2022.06.23

Function optimizations:

Android&iOS: Optimized callback parameters such as cachedBytes and IP address during playback.

Bug fixes:

Android&iOS: Fixed issue where hardware decoding failed for H265 format videos.

Android&iOS: Fixed issue with playing HLS live streaming.

iOS: Fixed issue with abnormal retrieval of supportedBitrates in certain scenarios.

Player SDK 10.1.0 @ 2022.05.31

Android&iOS: Optimized video super-resolution effect.

Android&iOS: Fixed issue with nested m3u8 refer header sub-stream transmission.

iOS: Resolved conflict with third-party SDK ffmpeg.

Android&iOS: Optimized player kernel performance.

Player SDK 9.5.29040 @ 2022.05.13

Android&iOS: Fixed issue where playing mp3 with cover image failed.

Player SDK 9.5.29036 @ 2022.05.06

Android: Fixed issue where SurfaceView caused black screen due to repeated add and remove.

Player SDK Android 9.5.29035, iOS 9.5.29036 @ 2022.04.28

Android&iOS: Added video preloading function.
Android&iOS: Added ability to pause player before onPrepared event.
Android&iOS: Added ability to maintain pause state when switching stream under pause state. Android&iOS: Optimized playback performance.

Player SDK 9.5.29016 @ 2022.03.30

Android&iOS: Added support for fine-grained control of cached traffic, preloading buffer and startup buffer can be controlled separately.
Android&iOS: Added ability to specify preferred resolution before startup and find the most suitable resolution to start playback.

Player SDK 9.5.29015 @ 2022.03.25

Android: Optimized playback performance.

Player SDK 9.5.29011 @ 2022.03.10

iOS: Optimized version compatibility issues.

Player SDK 9.5.29009 @ 2022.03.03

Android&iOS: Added support for terminal ultra-high definition, can be accessed through plugins.
Android&iOS: Optimized private encrypted video playback.
Android&iOS: Optimized accurate seeking to frames.
Android&iOS: Added support for EXT-X-DISCONTINUITY tag in HLS.
Android&iOS: Optimized player kernel and improved performance.
Android&iOS: Player component provides demo for immersive short video, feed video stream, video preview, video cover and video dynamic watermark functions.

Player SDK 9.5 @ 2022.01.11

Android: Fixed issue where switching resolution twice after playing any video in the video list to the end caused replay.
Android&iOS: Fixed issue where playback time point was inaccurate when playing back at different time points.

Player SDK 9.4 @ 2021.12.09

iOS: Fixed issue where switching HLS stream caused black screen.
iOS: Fixed issue where frequent seeking during playback with VOD player caused noise.
Android: Fixed issue where anti-theft chain sprite failed to retrieve.
Android: Fixed issue where VOD player occasionally reported errors during HLS offline download.

Android&iOS: Fixed issue where accurate seeking with player was inaccurate.

Player SDK 9.3 @ 2021.11.04

Android&iOS: Fixed issue where enabling preloading with VOD player and calling startPlay caused abnormal sound.
Android&iOS: Fixed issue where hardware decoding with VOD player caused callback resolution to be 0 for HEVC videos.

Player SDK 9.2 @ 2021.09.26

Android&iOS: VOD player supports HLS reinforcement encryption playback.
Android: Fixed issue where playing addresses with special characters failed.
Android: Fixed issue where frequent switching between foreground and background caused occasional sound without picture.

Player SDK 9.1 @ 2021.09.02

Android: Fixed issue where playback crashed on specific Android 5.x devices.
Android: Optimized live playback to prevent overexposure under specific conditions.

Player SDK 9.0 @ 2021.08.06

iOS: Fixed issue where enabling smoothSwitchBitrate caused crash when switching resolutions repeatedly.
iOS: Optimized VOD player to prevent playback progress from being abnormal after network recovery.

Player SDK 8.9 @ 2021.07.15

Android: Fixed issue where callback event logic was incorrect after VOD player lost network connection.

Player SDK 8.8 @ 2021.06.21

iOS: Fixed issue where starting and stopping playback with VOD player multiple times caused memory leaks.
Android: Fixed issue where playing HLS files on Android 11 caused errors.
Android: Fixed issue where default live playback was jerky and other live streams occasionally had accelerated sound and picture.
Android&iOS: Fixed issue where seeking with VOD player for specific videos was slow.
Android&iOS: Fixed issue where VOD player displayed slowly after pausing playback and setting progress with seek.

Player Adapter

Player Adapter 1.4.0 @ 2023.04.18

Android&iOS: Added support for decrypting VOD CDN encryption.

Player Adapter 1.2.0 @ 2022.03.10

Android&iOS: Added support for playing adaptive bitrate, transcoded, and original videos through FileId.

Player Adapter Release @ 2021.07.22

First release of iOS & Android player component Adapter.

Release Notes(Flutter)

Last updated：2025-06-23 11:43:04

Player SDK Flutter Version V12.6.0 @ 2025.06.20

Upgraded the native SDK version.
Added `setDrmProvisionEnv` interface in `SuperPlayerPlugin` to configure the DRM certificate environment.
Added `reDraw` interface in `TXVodPlayerController` to force a screen redraw.
The player UI component will no longer forcibly expand to fill the screen after entering full-screen mode to avoid unforeseen issues. Now, after entering full-screen, the business side needs to hide unnecessary components and remove height restrictions on the player.
Fixed known issues.

Player SDK Flutter Version V12.5.0 @ 2025.05.08

Upgraded the native SDK version.
Added `setRenderMode` interface in both VOD and live player controllers to directly set the current video's tiling mode.
Optimized the logic of the UI-based player component `super_player_widget` , making full-screen transitions smoother.
Fixed video jittering issues when scaling the PiP (Picture-in-Picture) window during live streaming on iOS.
After calling `stopPlay` on iOS, the previously set `startTime` parameter is no longer reset, aligning with Android behavior.
Fixed known issues.

Player SDK Flutter Version V12.4.2 @ 2025.04.30

Fixed known issues.

Player SDK Flutter Version V12.4.1 @ 2025.04.02

`TXPlayerVideo` no longer supports controller-based parameter passing.

Player SDK Flutter Version V12.4.0 @ 2025.03.31

Upgraded the native SDK version.
On Android, PiP (Picture-in-Picture) can hide the PiP window button by passing an empty string to the icon.
Deprecated the texture-binding method in `TXPlayerVideo` 's controller parameter passing, recommending the use of `onRenderViewCreatedListener` instead.
Fixed known issues.

Player SDK Flutter Version V12.3.1 @ 2025.03.18

Upgraded the native SDK version.
The PlatformView texture rendering version now supports both `SurfaceView` and `TextureView` modes on Android.
Added `onRenderViewCreatedListener` callback in `TXPlayerVideo` to obtain `viewId` , allowing controllers to bind textures via `setPlayerView` .
Added support for private DRM video playback.
Fixed known issues.

Player SDK Flutter Version V12.3.0 @ 2025.01.21

Upgrade the native SDK version.
Add pub dependency method.
This version is the last version of FlutterTexture rendering method.

Player SDK Flutter Version V12.2.1 @ 2024.12.30

Upgrade the native SDK version.
Add DRM playback API.
Fix the problem that events are not called back in some pre - download situations.
Fix the problem of abnormal startup of picture - in - picture on some Android models.
Fix other known issues.

Player SDK Flutter Version V12.2.0 @ 2024.12.04

Upgrade the native SDK version.
Pre - download supports httpHeader.
Support MP4 encrypted playback.
Add HEVC playback degradation - related parameters.
Fix other known issues.

Player SDK for Flutter V12.1.0 @ 2024.11.20

Upgrades the native SDK version.
Fix the issue of reversed logic in the live streaming mute method.
IOS adds support for Picture-in-Picture for live streaming, which requires premium permission to use.
Fixed other known issues.

Player SDK for Flutter V12.0.1 @ 2024.09.25

Upgrades the native SDK version.
Optimized the native message interaction architecture.
The SDK plugin feature module is now loaded lazily instead of during plugin initialization.
Fix the issue that the player texture does not refresh under some circumstances.
Fixed the issue where the Android Picture-in-Picture UI was displaced during updates in certain situation.
The demo and player component no longer require mandatory language settings. The default language is English UI if not set.
A placeholder image is added when the player component loads the network cover.
Add VOD_PLAY_EVT_VOD_PLAY_FIRST_VIDEO_PACKET event for receiving the first frame data packet.
Fixed other known issues.

Player SDK for Flutter V12.0.0 @ 2024.08.21

Upgrades the native SDK version.
Live streaming replaces the new kernel.
Due to the new kernel replacement, the live config currently only retains the maxAutoAdjustCacheTime, minAutoAdjustCacheTime, connectRetryCount, and connectRetryInterval attributes, while other parameters are marked as deprecated.
New live streaming interfaces: enableReceiveSeiMessage, showDebugView, setProperty, getSupportedBitrate, setCacheParams.

When playing a live stream, it is no longer necessary to pass the playType parameter, as this parameter is now deprecated.
Added logic to the live streaming and on-demand demo pages to wait for the license to load successfully before playing.
Fixed other known issues.

Player SDK for Flutter V11.9.1 @ 2024.06.20

Upgrades the native SDK version.
Fixed the issue where video playback fails in the picture-in-picture window after recovering from lock screen.
TXVodPlayerController introduced a new setStringOption interface for configuration expansion.
Player operations on the Flutter side can now affect the play and pause UI updates in the picture-in-picture window.
Fixed potential memory leak issues.
Optimized the logic of the superPlayer component.
Fixed other known issues.

Player SDK for Flutter V11.9.0 @ 2024.06.05

Upgrades the native SDK version.
Android compatibility with higher versions of Gradle.
The position of superPlayerWidget has changed. Integrating superPlayer will no longer include the source code of superPlayerWidget.
Android Picture-in-Picture feature logic optimization, compatible with more models.

Player SDK for Flutter V11.8.1 @ 2024.05.22

Upgrades the native SDK version.
Fixed known issues.

Player SDK for Flutter V11.8.0 @ 2024.05.06

Upgrades the native SDK version.
Added premium version.
Fixed known issues.

Player SDK for Flutter V11.7.0 @ 2024.04.07

Upgrades the native SDK version.
SuperPlayerPlugin added setSDKListener for monitoring License loading results
Added Player_Premium branch, supports external subtitles, multiple tracks, and other value-added features.

Player SDK for Flutter V11.6.1 @ 2024.01.29

Upgrades the native SDK version.
New `stopPlay` method and Tile Mode configuration added to the Player component.
The `dispose` method for on-demand playback and live streaming components now supports `await` .
Fixed known issues in the Player component and Player.

Player SDK for Flutter V11.6.0 @ 2024.01.11

Upgrades the native SDK version.
Adapted to the latest version of flutter development environment.
Fixed known issues in the Player component and Player.

Player SDK for Flutter V11.4.1 @ 2023.12.20

Upgrades the native SDK version.
Supports fileId pre-download capability.
Fixed known issues.

Player SDK for Flutter V11.4.0 @ 2023.08.30

Flutter Player SDK supports internationalization.
Fixed known issues.

Player SDK Flutter version 11.3.0 @ 2023.07.07

Removed deprecated live time-shifting interface.
Fixed known issues.

Player SDK for Flutter V11.2.0 @ 2023.06.05

Minimum requirement version for Flutter development environment has been raised.
Fixed known issues.

Player SDK for Flutter V11.1.1 @ 2023.05.08

Removed unnecessary third-party dependencies.
Fixed known issues.

Player SDK for Flutter V11.1.0 @ 2023.04.10

Refactored native bridging layer.
Fixed known issues.

Player SDK for Flutter V11.0.0 @ 2023.02.28

Added offline download demo.
Fixed known issues including sprite map disorder and failure to open picture-in-picture on iOS.

Player SDK for Flutter V10.9.0 @ 2023.01.03

Fixed the issue of picture-in-picture not supporting playback of encrypted videos on Android.
Fixed the issue of automatic playback after returning to the application interface from the background when live streaming or VOD is paused.
Optimized log output for easier issue tracking.

Player SDK for Flutter V10.8.0 @ 2022.10.20

Adjusted the picture-in-picture interaction of the player component, supporting returning to the previous page after enabling picture-in-picture.

Player SDK for Flutter V10.7.0 @ 2022.09.20

Changed the startPlay interface for VOD playback to startVodPlay.
Changed the startPlay interface for live streaming playback to startLivePlay.
Changed the playWithModel interface for the player component to playWithModelNeedLicence.

Player SDK for Flutter V1.0.3 @ 2022.07.05

Added picture-in-picture (PIP) function for Android and iOS.

Player SDK for Flutter V1.0.2 @ 2022.06.24

Added the SuperPlayer component with integrated UI solution.
Improved the SDK interfaces for live streaming and VOD.
Fixed the issue of playback failure when using fileId and psign.

Player SDK for Flutter released on 2021.07.16

Release Flutter Player SDK.

Licenses
Adding and Renewing a License

Last updated：2025-02-12 19:52:51

Overview

The Player SDK provides video playback capabilities for live streaming and VOD. Mobile and Web platforms are separately authorized and billed. You need to access the corresponding License before you can use the respective features.

The access and usage of License are as follows:

1. **Purchasing License**

You can refer to the product features and purchase guide to confirm and purchase the License you need. If you want to apply for a trial License for testing, please see the free trial guidelines.

2. **Bind License**

After purchasing a License, you need to bind the newly purchased License with the application/domain you want to authorize to implement authorization for the corresponding application/domain. Mobile platforms are authorized by application package name, and Web platforms by domain name. The guidance on mobile platform binding operations and Web platform binding operations are below.

3. **Configure License**

After binding, you will obtain the authorization credential **License URL** and **License Key** in the console. During the SDK integration process, you need to enter the corresponding information. Please keep it safe. For the specific method of entry, refer to the integration documentation for each platform.

After purchasing or renewing a new license, the terminal application will automatically complete the update when it is connected to the Internet. If it is not effective, it may be affected by the local cache. You can solve this problem by restarting the application.

Note:

MLVB License and UGSV License include all the capabilities of the Player Mobile Basic Version License. Therefore, they can also be used to unlock the basic features of the Player Mobile SDK. For more information on MLVB License, see Adding and Renewing a License and Adding and Renewing a License. For more information on UGSV License, see License Pricing Overview and Adding and Renewing a License.

By default, only one set of License URL and Key is generated under a single Tencent Cloud account, regardless of License types and quantities. The License URL and Key are unique and unchangeable. (This means under one account, regardless of how many SDK Licenses are added, how many package names are added, or whether Licenses expire/renew/are newly purchased, the License URL and Key remain constant).

Add and Renew a License (Mobile)

Purchase an official License

Before binding the License, you can access the Player License in the following manner:

Types of License	Acquisition Method	Price (USD)	Validity Period
Player Mobile Basic Version License	Apply for free	0	1 year
Player Mobile Premium Version License	Purchase now	499	1 month

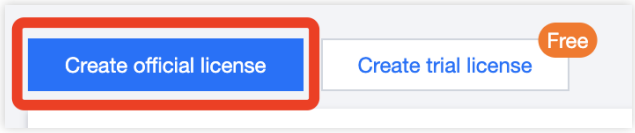
Bind an official License

After purchasing, you need to go to the VOD console **or** CSS console to bind the License to the application to activate it. You can choose to **Method 1: create a new official application and select the player License** or **Method 2: unlock the player feature in an existing application and bind the License** as two ways to officially bind the License.

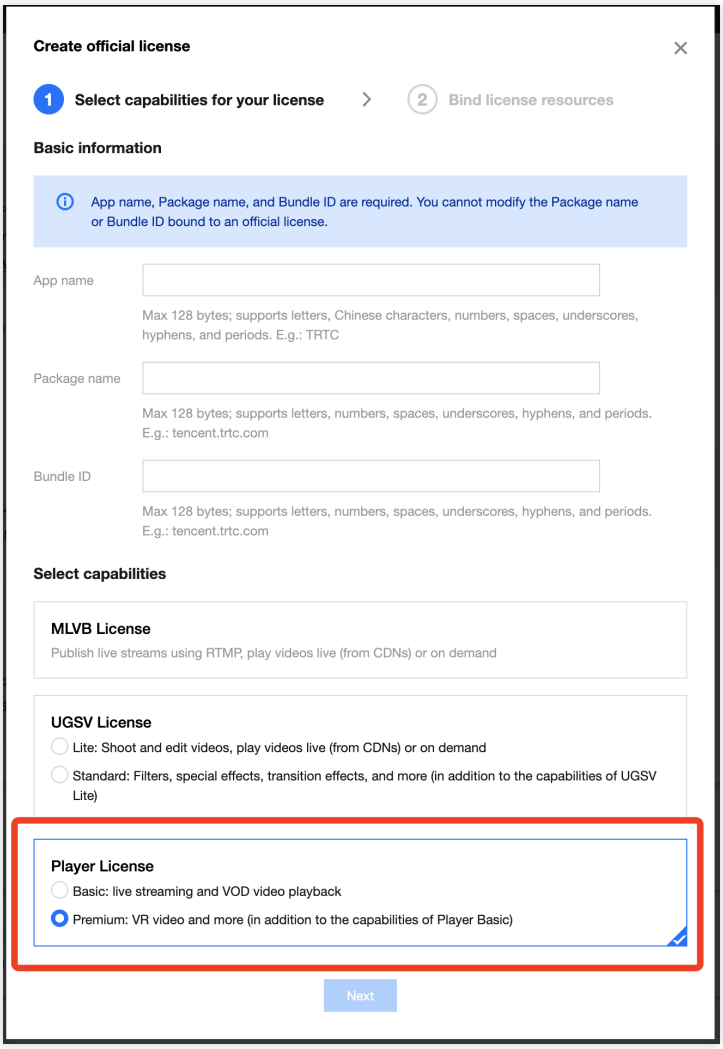
Method 1

Method 2

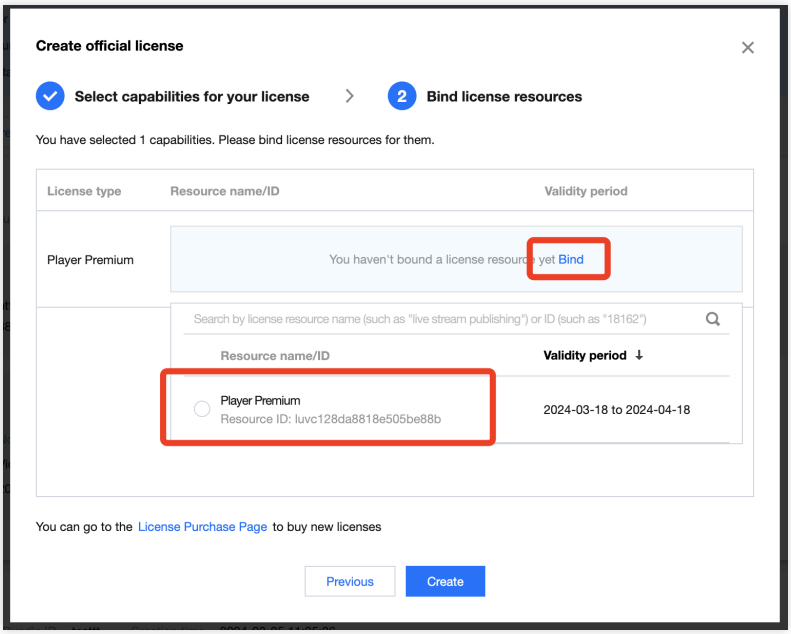
1. Go to the **VOD console or CSS console > License Management > Mobile License**, and click **Create official license**.



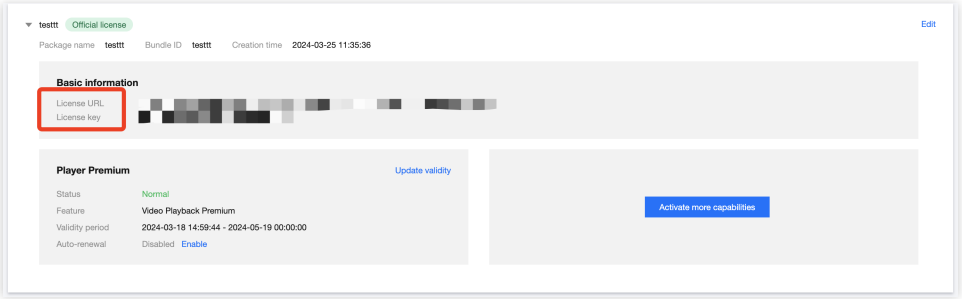
2. Enter an app name, a package name, and a bundle ID, select **Player License** , choose **Basic Version** or **Premium Version**, and click **Next**.



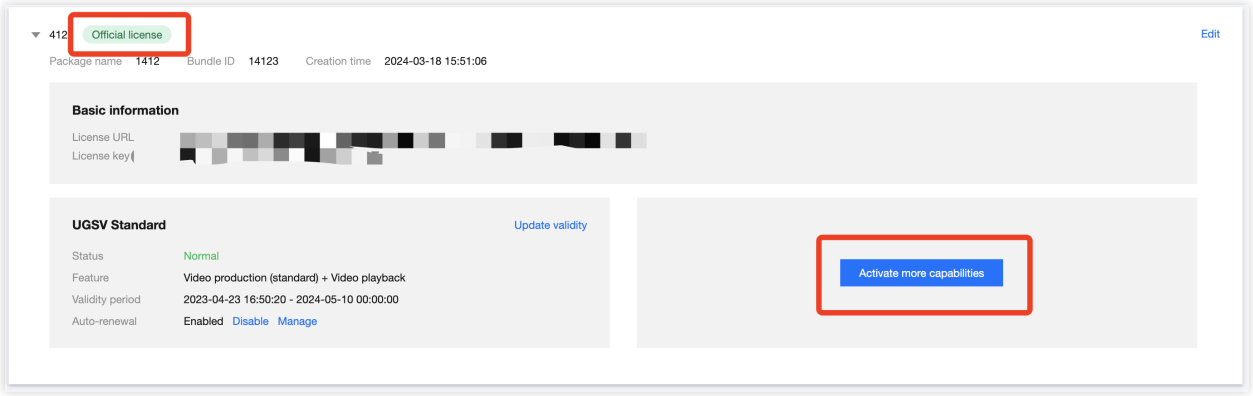
3. Go to the **Choose License resources and bind** page, click **Bind now**, select the **Unbound** player License (if there are no bindable License resources, refer to Purchase Player License), and click **Create** to create the application and generate an official License.



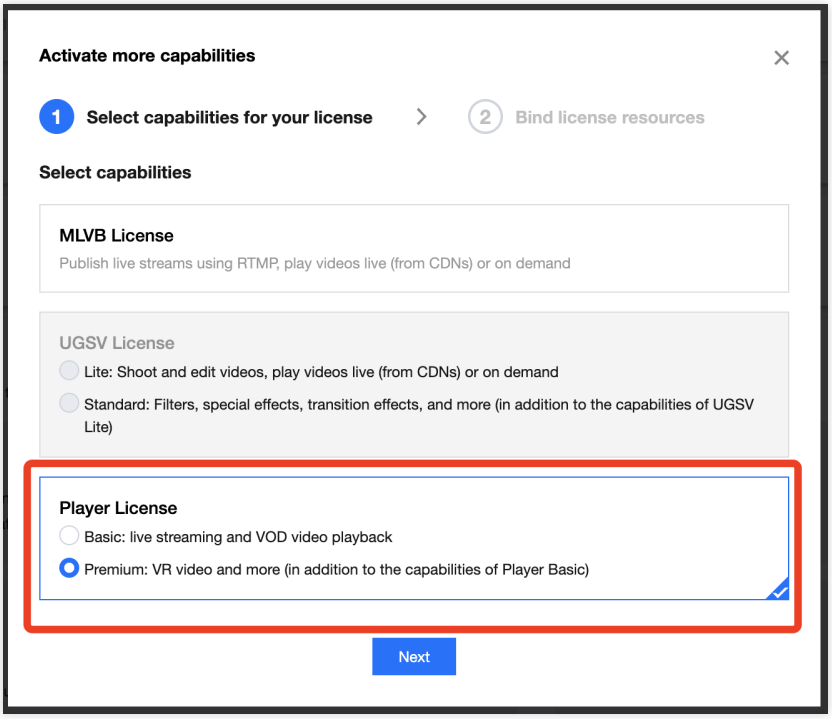
4. After the official License is successfully created, the page will display the generated official License information. When initializing SDK configuration, you need to input License URL and License Key. Please save the following information properly. Refer to Configuring and Reviewing License guide to input your License URL and License Key in the SDK to complete License authorization.



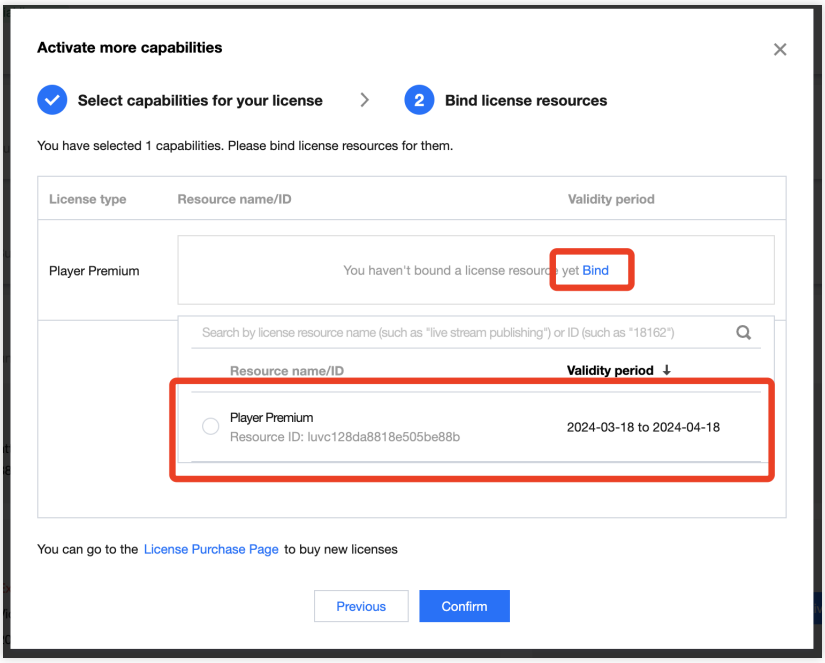
1. Go to the **VOD console** or **CSS console**> **License Management** > **Mobile License**.
2. Select the Official license that you want to add the player feature, and click **Activate more capabilities**.



3. Select **Player License**, then click **Next**.



4. Enter the **select and bind License** page, click **Bind now**, select the **unbound** player License (if there are no bindable License resources, refer to purchasing a player License), and click **Confirm** to generate the official version of the player feature under the application.



Note:

Before clicking **Confirm**, double-check the bundle ID and package name and make sure they are identical to what you submit to app stores. **The information cannot be modified after submission.**

Update official License valid period

You can log in to **the VOD console** or **the CSS console** > **License Management** > **Mobile License** page to view the valid period of the Player Mobile official license. You can also subscribe to the SDK in Message Subscription, set up **Message Center**/**Email**/**SMS** and other message receiving channels to receive official license expires reminders. The player official license will send you an expires reminder 32 days, 7 days, 3 days, and 1 day before the current time. If the license is renewed after it expires, it may be affected by the local cache and may not take effect in time. Therefore, it is recommended that you renew it in advance to ensure business continuity.

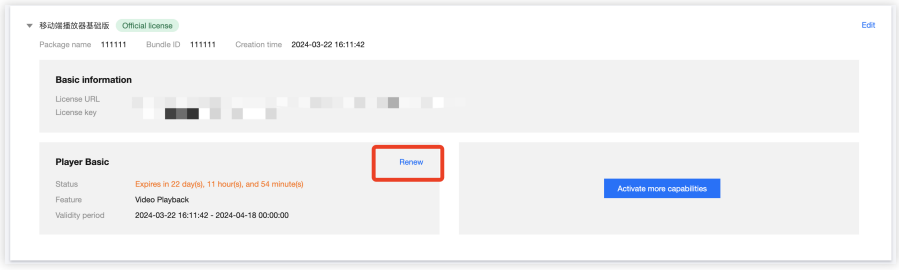
If your Player Mobile official license has expired, you can apply to renew the validity period **within one month before expiration**. Please refer to the following operations to renew:

Update Basic Version License valid period

The Mobile **Basic Version** is free to use. Choose the license you need to update the valid period. If the remaining valid period is within 30 days, you can click renew to extend the valid period for free.

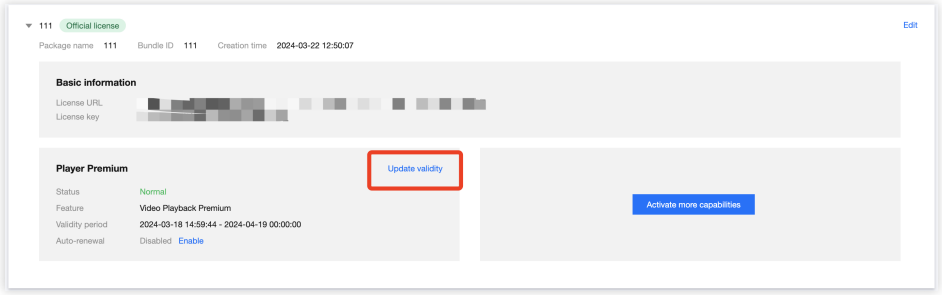
Note:

The Player Mobile Basic Version License does not support enabling auto-renewal.



Update Premium Version License valid period

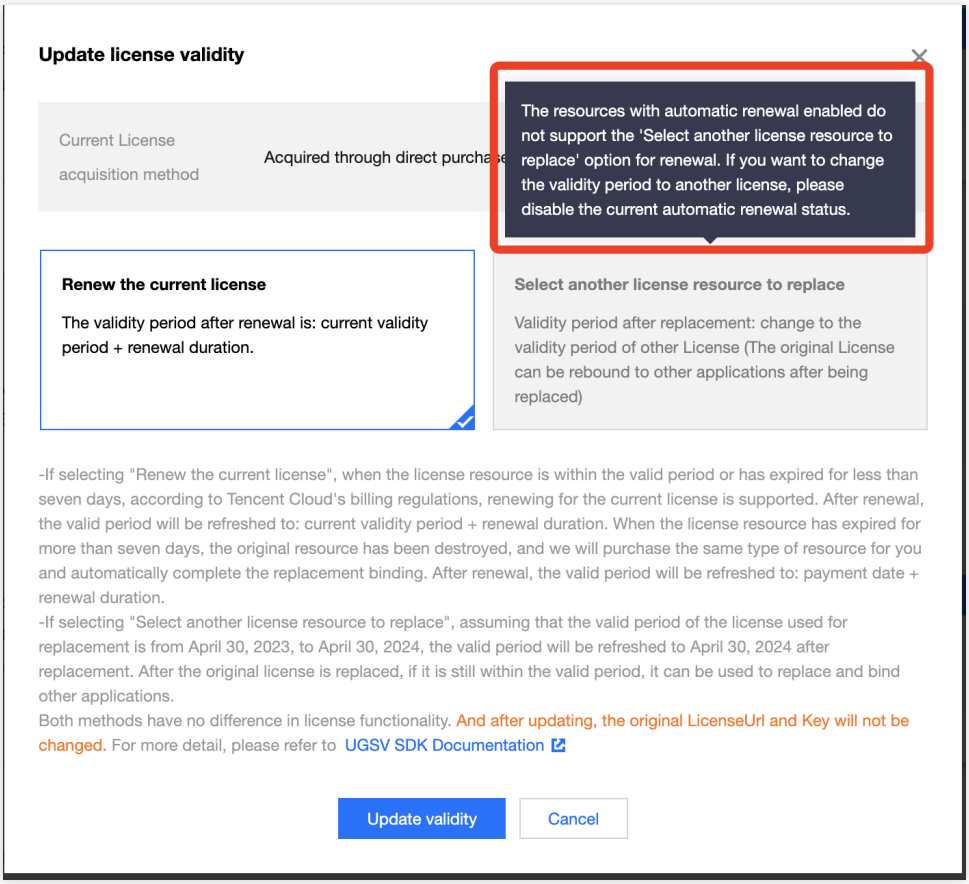
1. Select the target license and click **Update validity** in the **Player** features.



2. You can choose two methods to update the license valid: **Renew the current license** or **Select another license resource to replace**. Specific as follows.

Note：

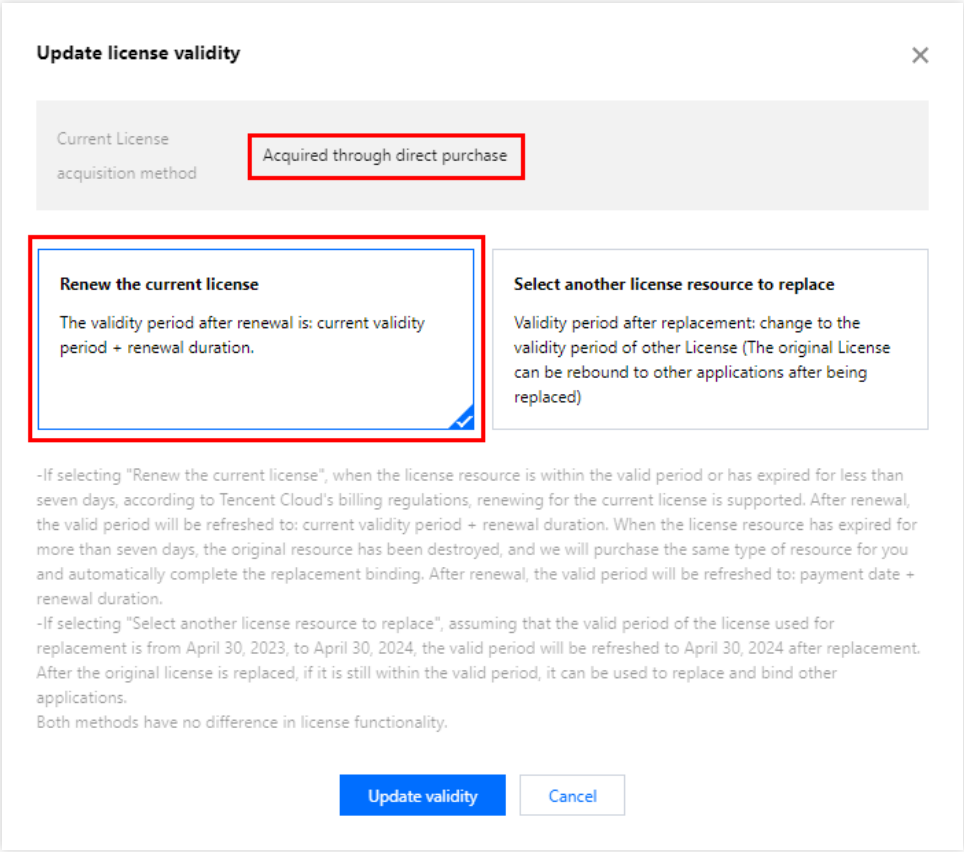
Resources with auto-renewal enabled do not support the license resource replacement method for renewal. If you want to change the validity period to that of another license, disable the auto-renewal function.



Renew the current license

Select another license resource to replace

1. Click **Renew the current license**, and click **Update validity**.



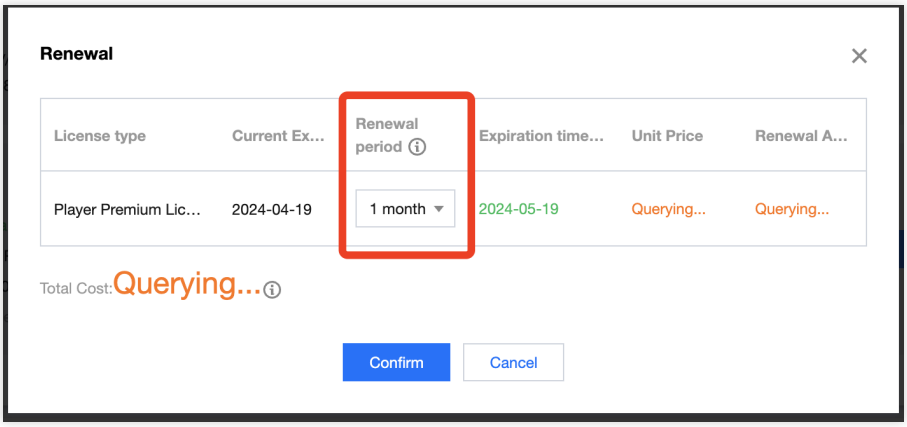
Note:

If you select **Renew the current license**:

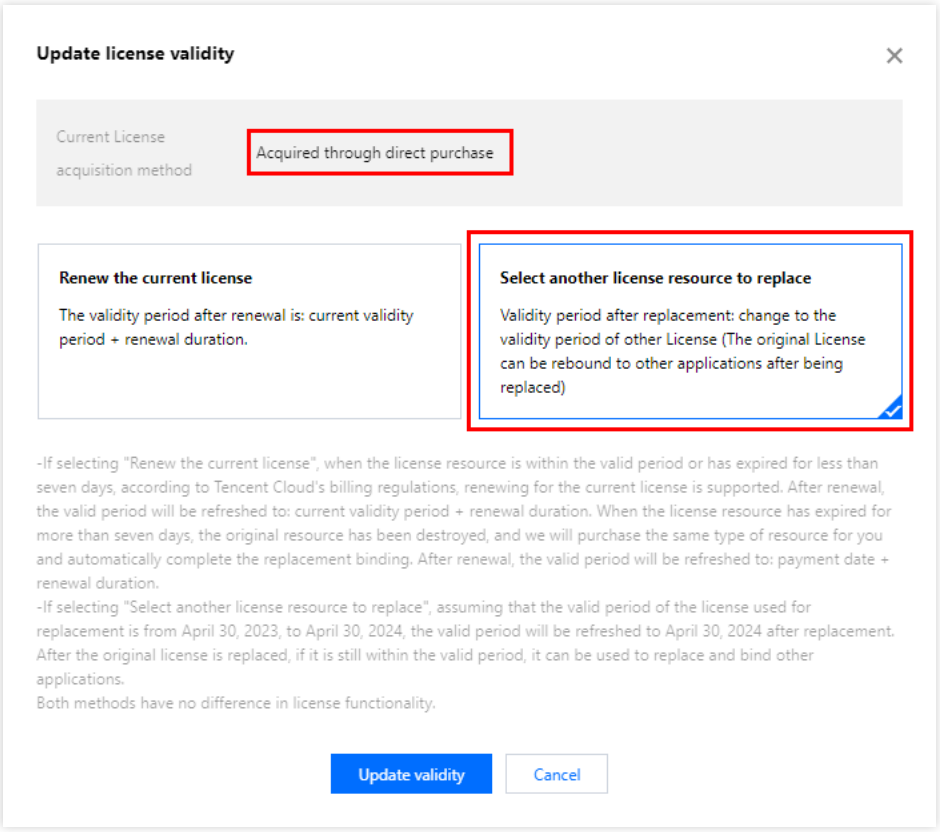
When the license resource **is within the valid period or has expired for less than seven days**, according to Tencent Cloud's billing regulations, renewing for the current license is supported. After renewal, the valid period wil be refreshed to: **current validity period + renewal duration**.

When the license resource **has expired for more than seven days**, the original resource has been destroyed, and we wil purchase the same type of resource for you and automatically complete the replacement binding, After renewal, the valid period will be refreshed to: **payment date + renewal duration**.

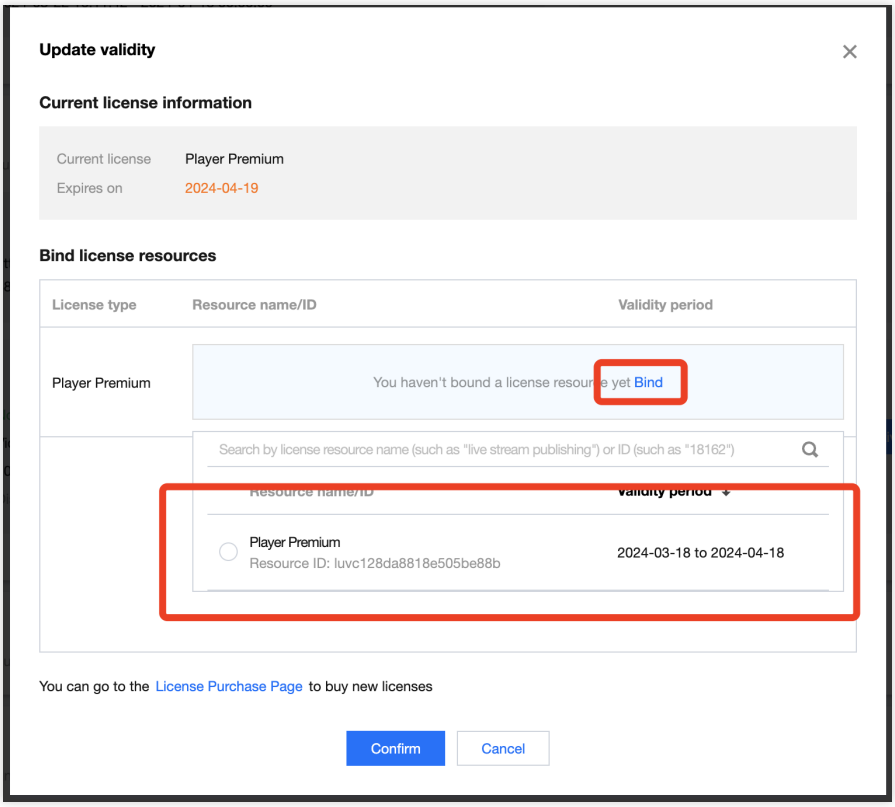
2. In the **Renewal** interface, select the **Renewal period**. The Player Premium Version License renews **monthly**. Click **Confirm** to extend the license valid period.



1. Click **Select another license resource to replace**, and click **Update validity**.



2. In the **Update validity** interface, click **Bind**. Select the unbound Player Advanced license (if there is no available resource pack to bind, you can go to License Purchase Page to buy), and click **Confirm**.



3. Check the renewed validity period.

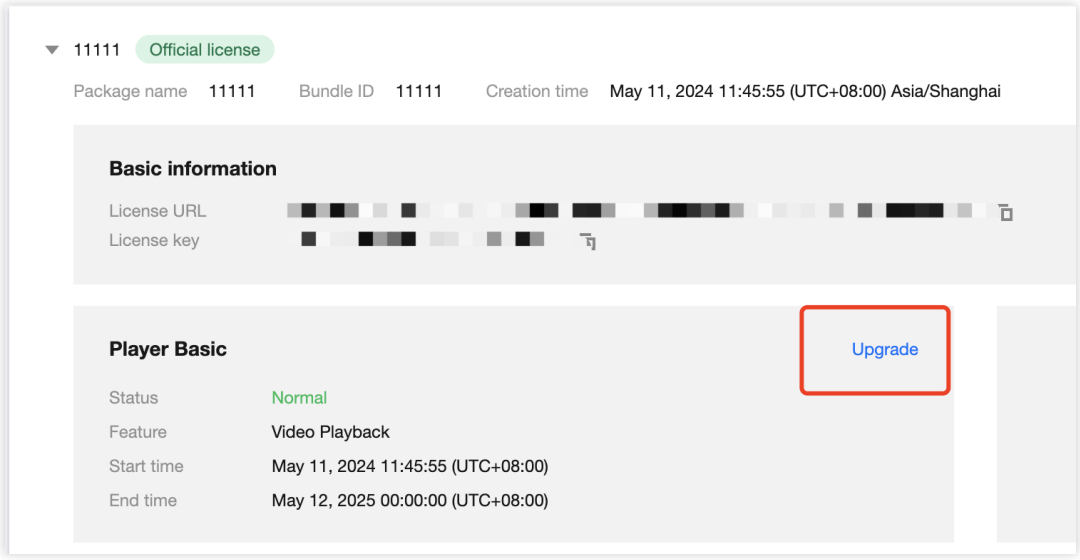
Note:

The official license of the player does not support information modifications. If you need to make modifications to the license information, after purchasing the resource package, please do not use it for the update of the valid license period. Click to **Create official license and bound the license** to recreate the application, add a new license, and bound the new bundle ID information.

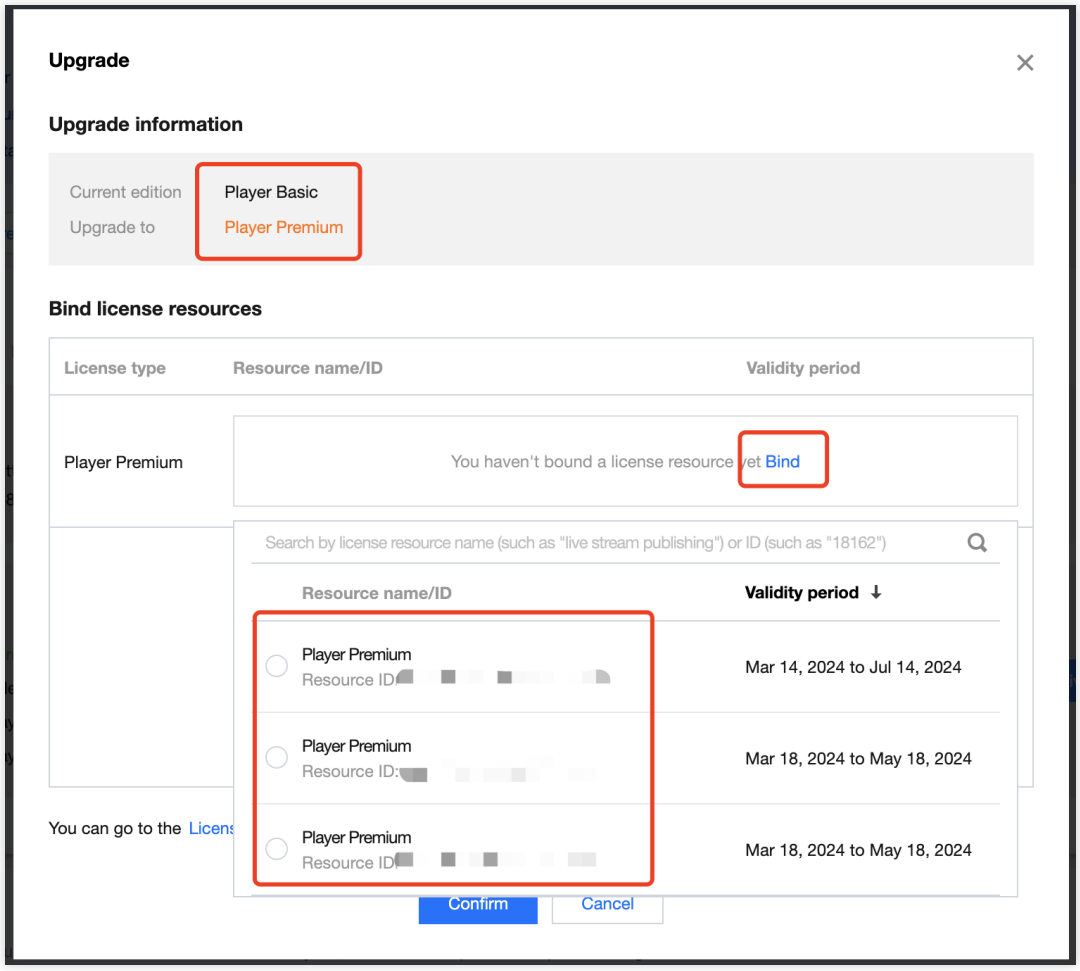
Upgrade Basic to Premium Version License

If you have Activated the Mobile Player **Basic Version License** and require External Subtitles, Advanced Picture-in-Picture Components, and other premium features, you can follow the instructions below to upgrade to the Mobile Player **Premium Version License**, unlocking more features:

1. Select the official Mobile Player **Basic Version License** you want to upgrade and click on the **Upgrade**.



2. Enter the upgrade interface and click **Bind**. Then select the Player Premium License you want to bind and click **Confirm** to upgrade to the Mobile Player **Basic Version License**.



Add and Renew a License (Web)

Purchase an official License

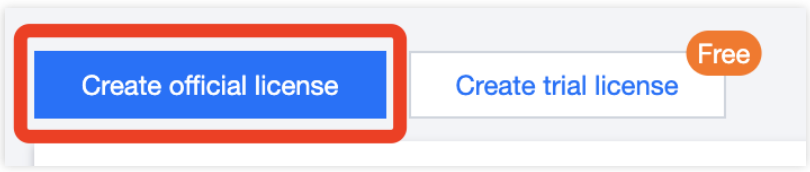
Before binding the license, you can refer to the following methods to obtain the player license:

Types of License	Authorization Unit	Acquisition Method	Price (USD)	Validity Period
Player Web Basic Version License	Precise Domain (1 license can authorize up to 10 precise domains)	Apply for free	0	1 year
Player Web Premium Version License	Wildcard Domain (1 License can authorize 1 wildcard domain)	Purchase now	99	1 month

Bind an official license

After purchase, you need to bind the license to the application in **VOD console** or **CSS console** > **License Management** > **Web License** to activate it.

1. Enter the **Web License**, and click on **Create Official License**.



2. Select the version. The player's Web official license includes Basic and Premium Versions.

The Basic Version can be applied for free, with a valid period of one year. It only supports **precise domain** bound, and can be associated with up to **10**.

The Premium Version is for monthly paid use. It only supports **wildcard domains** bound, and can be associated with **1**.

Add Basic Version License

Add Premium Version License

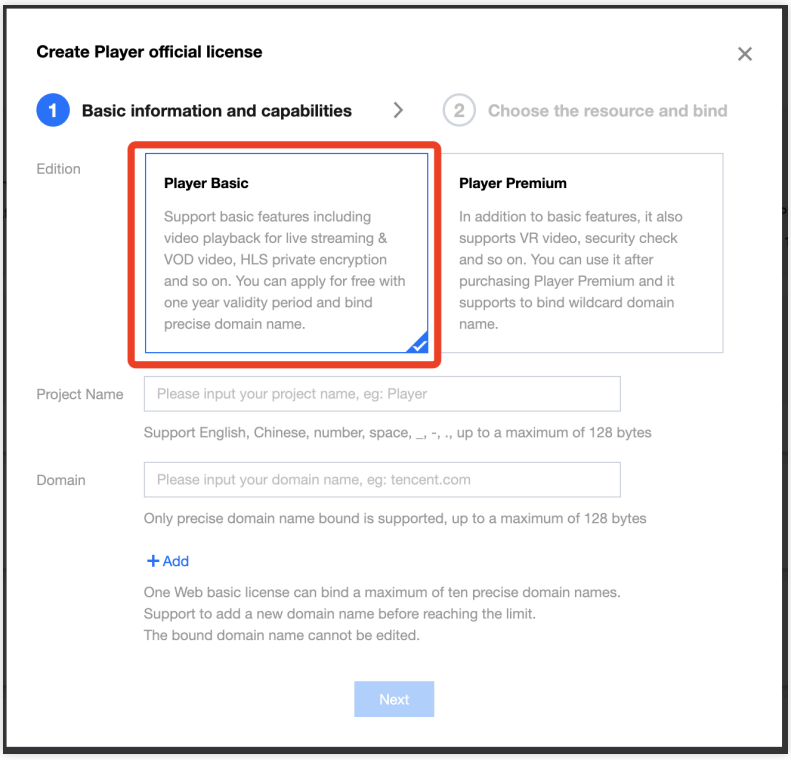
Note：

1 Web Basic Version License can be bound to up to 10 precise domains.

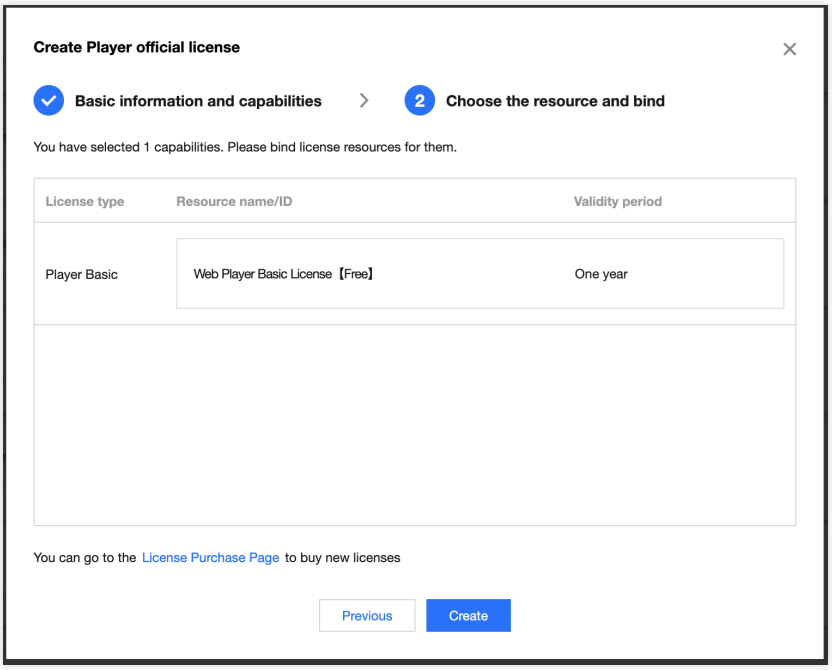
You can add domains at any time before reaching the limit.

Bound domains cannot be modified.

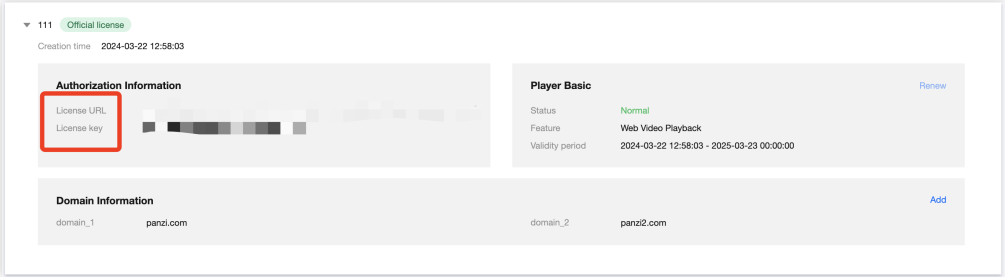
1. Select **Player Basic**, fill in the `Project name` and `Domain`, and click **Next**.



2. Automatically select the **1-year free Web Player Basic Version License** resource, and click create to create the application and generate the license.



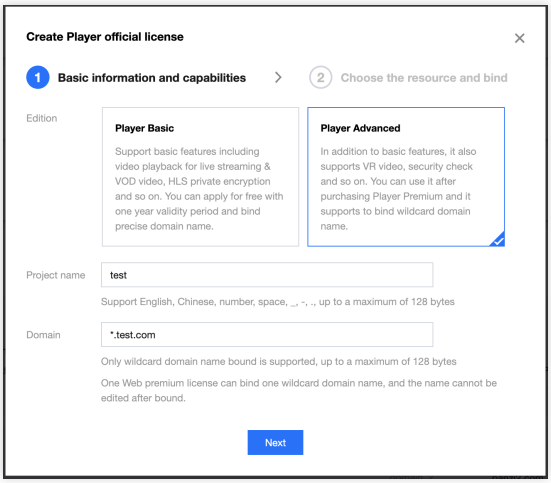
3. After the official license is successfully created, the page will display the generated official license information. When initializing the SDK configuration, you need to input the License URL and License Key parameters, please properly save the following information.



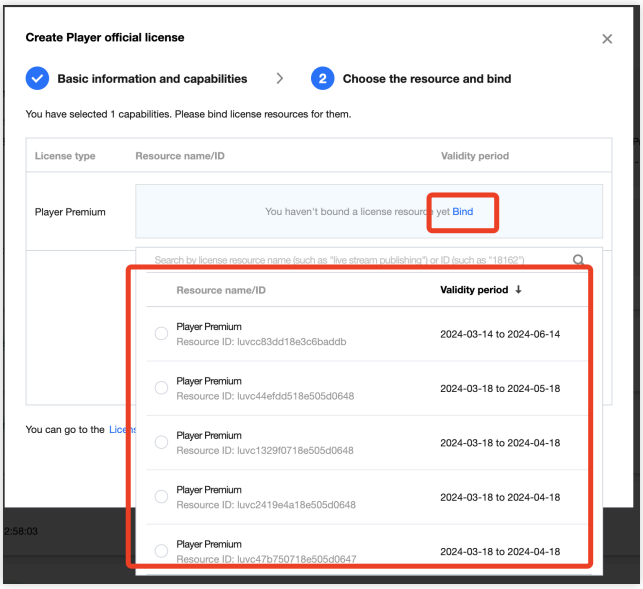
Note：

1 Web Premium Version License can be associated with 1 wildcard domain and the domain cannot be modified once bound.

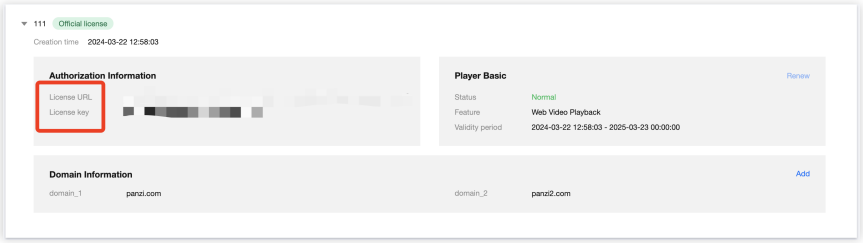
1. Select **Player Premium**, fill in the `Project name` and `Domain`, and click **Next**.



2. Enter **the selection of license resources and binding page**, click **Bind**, select the unbound player license (if there is no bindable license resource, you can refer to Purchase Player License), and click **Create** to create the application and generate the Advanced license.



3. Upon the successful creation of the Official License, the page will display the produced Official License information. This information, including the License URL and License Key, is required when configuring the SDK initialization. Please make sure to securely store this information.



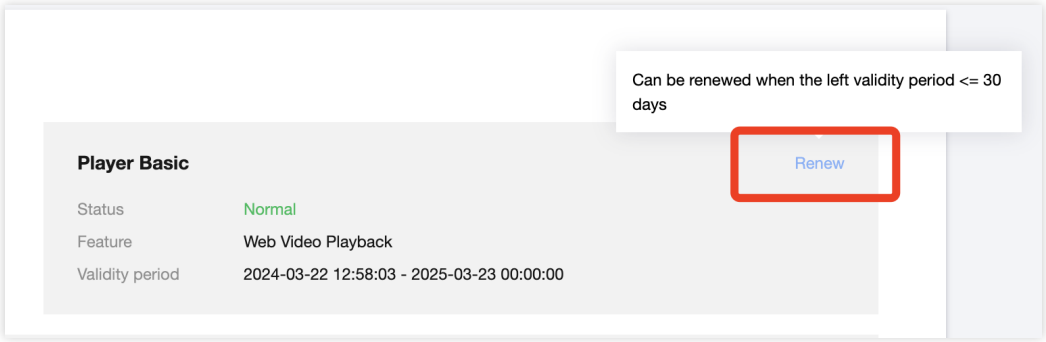
Update official License valid period

You can log in to **VOD console** or **CSS console** > **License Management** > **Mobile License** page to view the valid period of the Player Web official license. You can also subscribe to the SDK in Message Subscription, set up **Message Center**/**Email**/**SMS** and other message receiving channels to receive official license expires reminders. The player official license will send you an expires reminder 32 days, 7 days, 3 days, and 1 day before the current time, reminding you to renew in a timely manner to avoid affecting the normal business operation.
If your Player Web official license has expired, you can apply to renew the validity period **within one month before expiration**. Please refer to the following operations to renew:

Update Basic Version License valid period

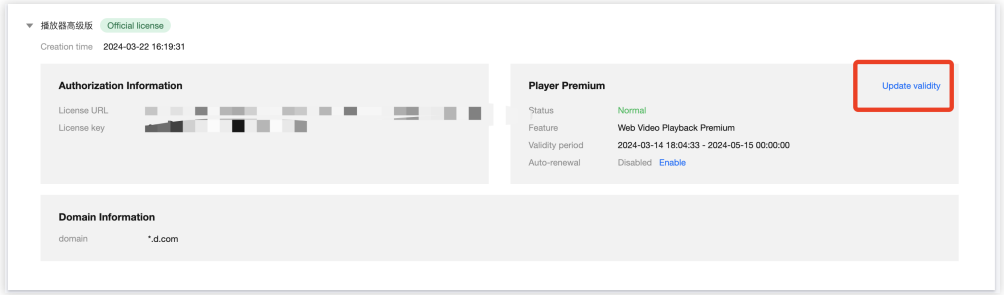
The Web **Basic Version** is free to use. Choose the license you need to update the valid period. If the remaining valid period is within 30 days, you can click **Renew** to extend the valid period for free.
Note:

The Player Web Basic Version License does not support enabling auto-renewal.



Update Premium Version License valid period

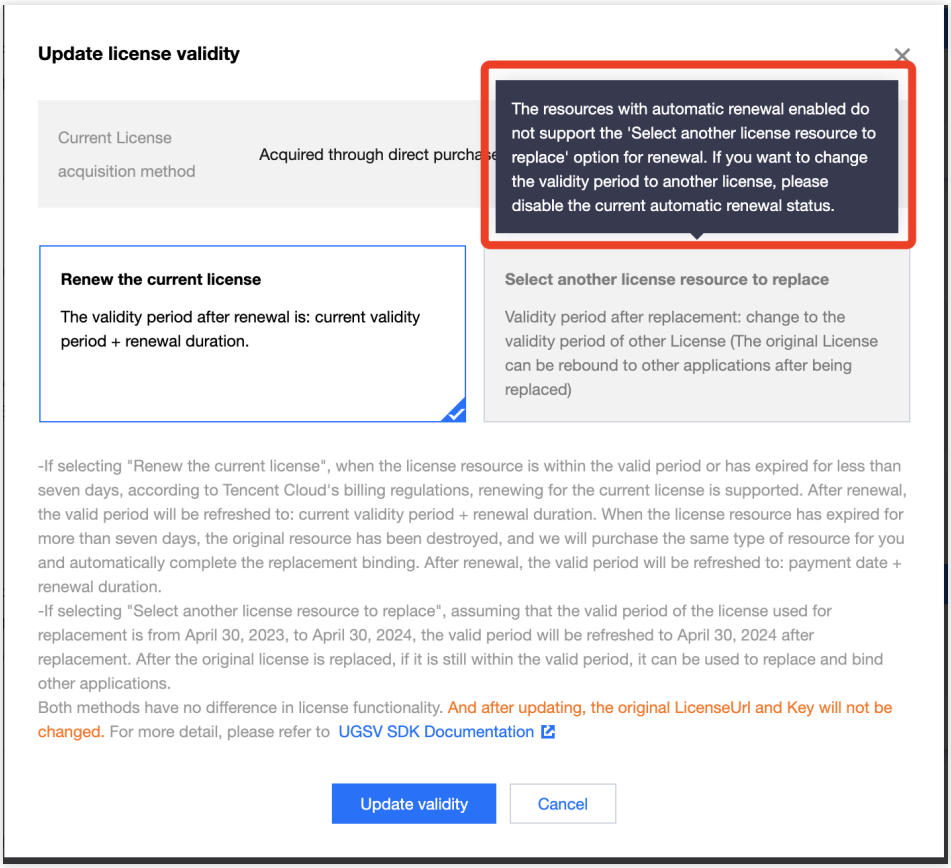
1. Select the target license and click **Update validity** in the **Player** features.



2. The Player Premium Version License supports 2 methods of updating the valid period: **Renew the current license** or **Select another license resource to replace**. The specific methods are as follows.

Note：

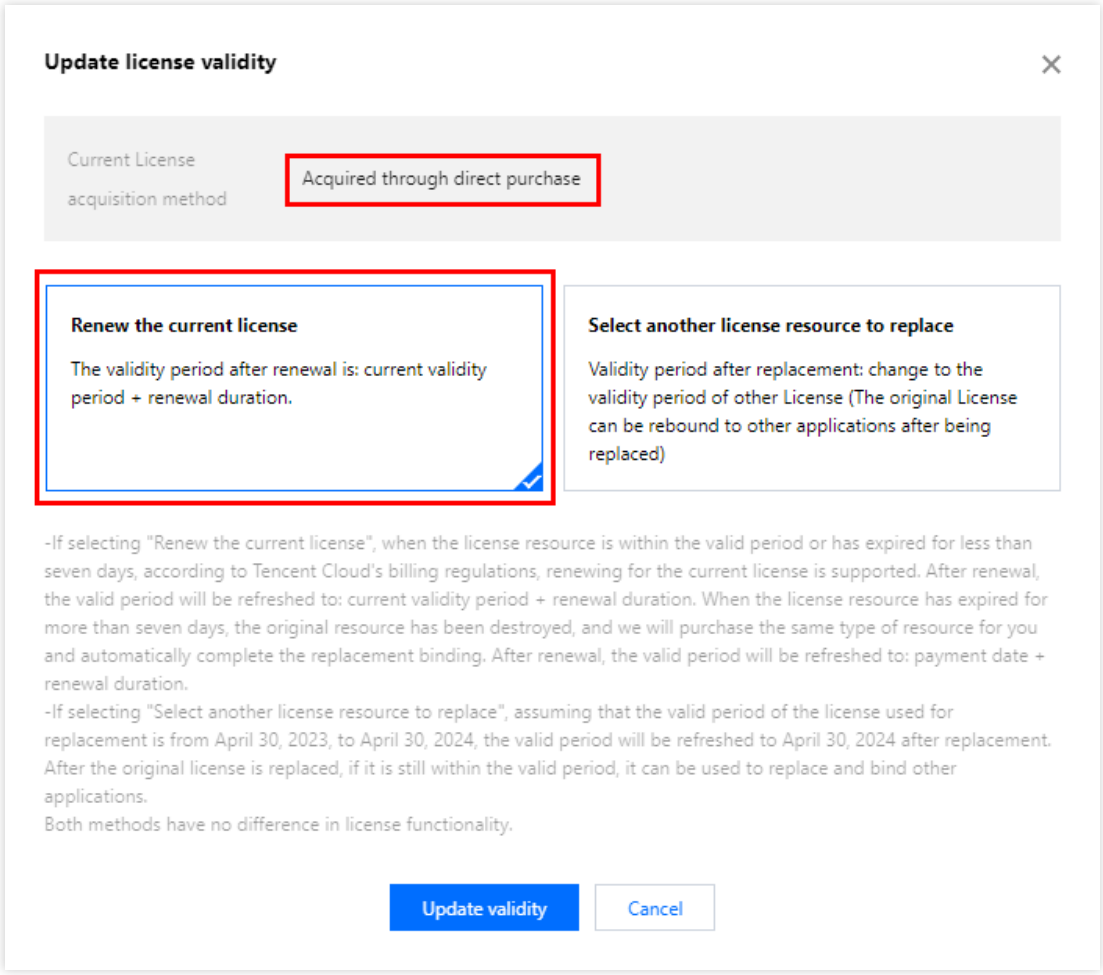
Resources with auto-renewal enabled do not support the license resource replacement method for renewal. If you want to change the validity period to that of another license, disable the auto-renewal function.



Renew the current license

Select another license resource to replace

1. Click **Renew the current license**, and click **Update validity**.



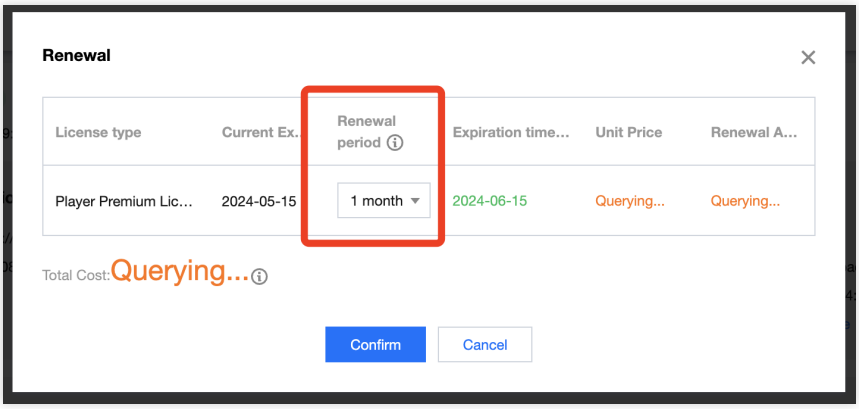
Note:

If you select **Renew the current license**:

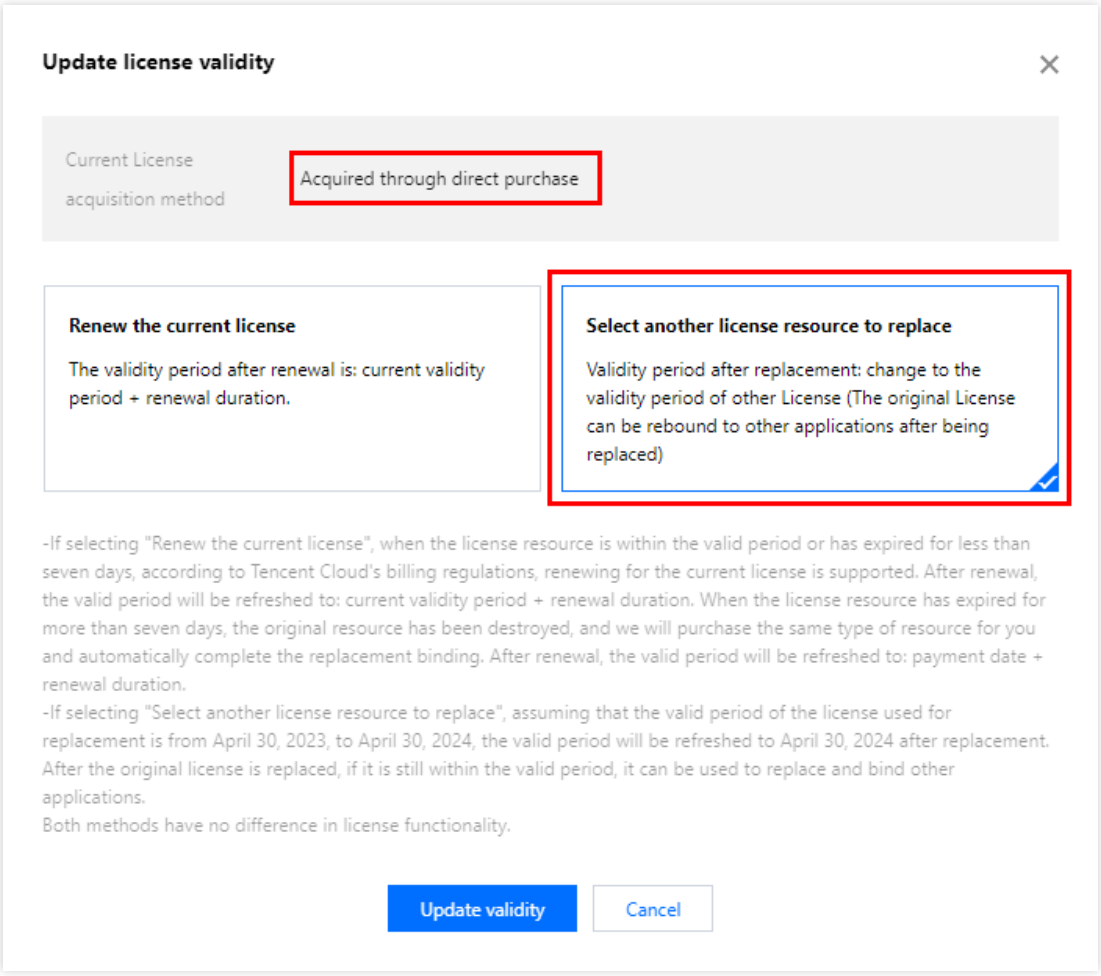
When the license resource **is within the valid period or has expired for less than seven days**, according to Tencent Cloud's billing regulations, renewing for the current license is supported. After renewal, the valid period wil be refreshed to: **current validity period + renewal duration**.

When the license resource **has expired for more than seven days**, the original resource has been destroyed, and we wil purchase the same type of resource for you and automatically complete the replacement binding, After renewal, the valid period will be refreshed to: **payment date + renewal duration**.

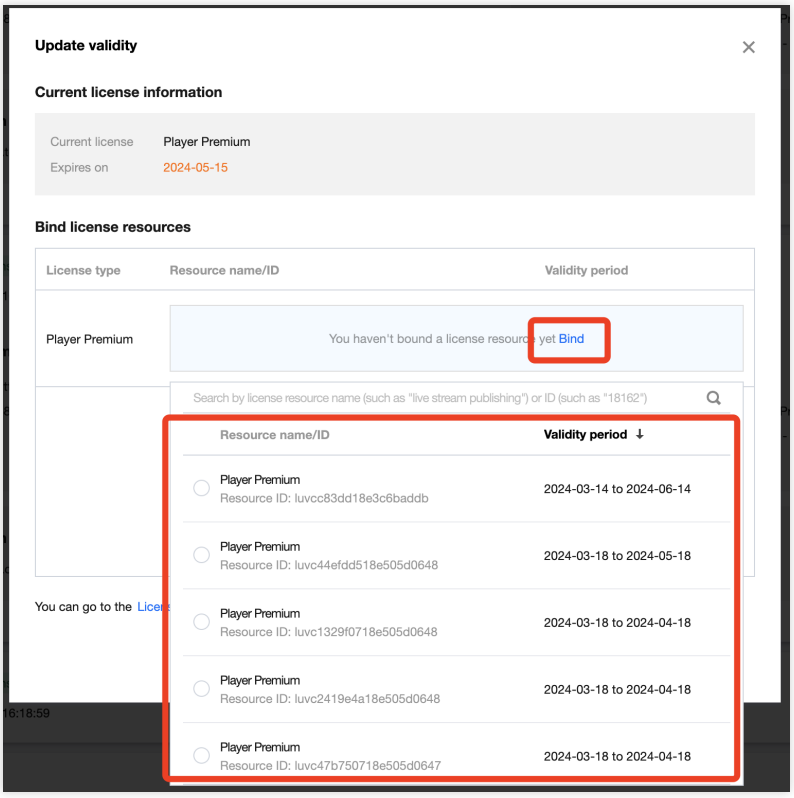
2. In the **Renewal** interface, select the **Renewal period**. The Player Premium Version License renews **monthly**. Click **Confirm** to extend the license valid period.



1. Click **Select another license resource to replace**, and click **Update validity**.



2. In the **Update validity** interface, click **Bind**. Select the unbound Player Advanced license (if there is no available resource pack to bind, you can go to License Purchase Page to buy), and click **Confirm**.



3. Check the renewed validity period.

Note:

The official license of the player does not support information modifications. If you need to make modifications to the license information, after purchasing the resource package, please do not use it for the update of the valid license period. Click to **Create official license and bound the license** to recreate the application, add a new license, and bound the new bundle ID information.

Auto-renewal

You can manage auto-renewal through **Console** and **Billing center** in 2 methods. The details are as follows.

Console

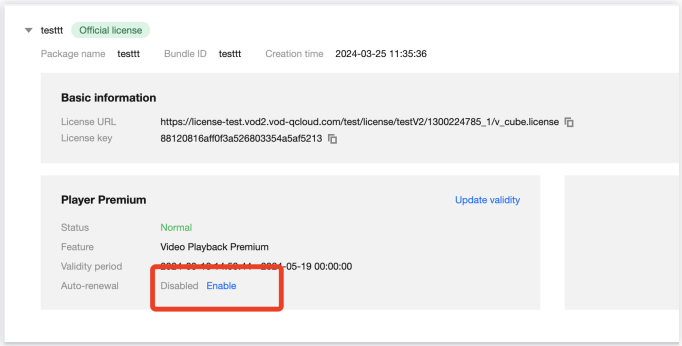
Billing center

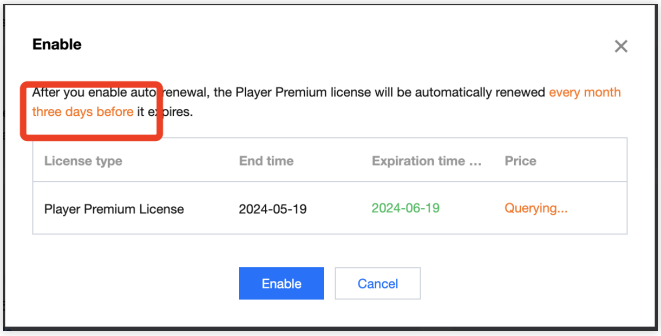
License supports the enabling of automatic renewal. License resources with automatic renewal enabled will be **automatically renewed monthly 3 days before expiration**. Make sure your account has sufficient available balance before enabling automatic renewal. Otherwise, it may lead to a renewal failure and affect your usage.

Log in to the CSS or VOD console **> License Management**, locate the license you wish to manage for automatic renewal:

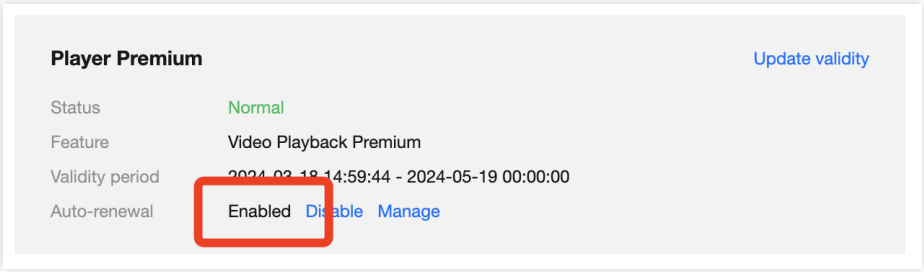
1. Enable Auto-renewal.

1.1 In the **Disabled** status of the license Auto-renewal, click to **Enable** auto-renewal, and it will be automatically deducted and renewed **monthly** three days before expiration.

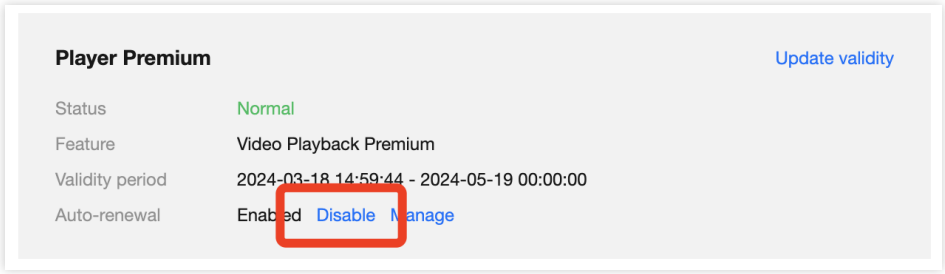


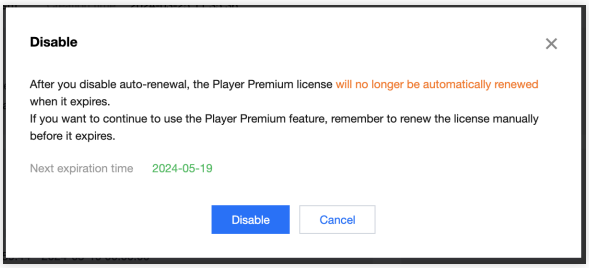


1.2 **Auto-renewal** status changed to **Enabled**.



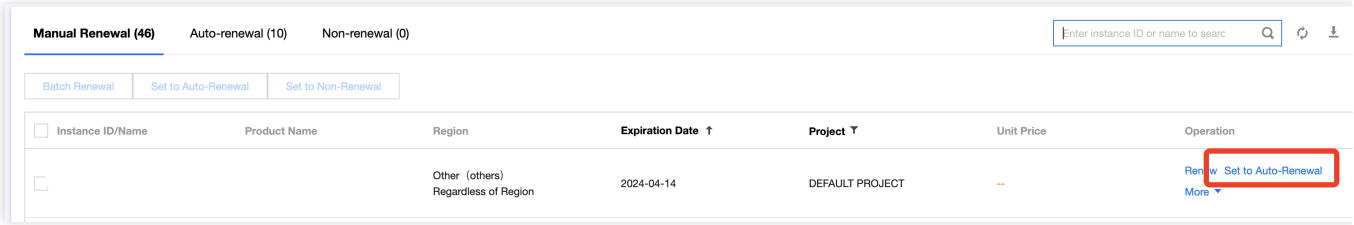
2. Disable Auto-renewal. The **Auto-renewal** of the license can be turned off in the **Enabled** status by click **Disable**. After it expires, it will no longer be automatically renewed.





You can navigate to Renewal Management to set resources to automatic renewal.

In the search box on the right, search for a **Player**, locate the target resource, and click **Set to Auto-Renewal**.



Configuring and Viewing a License

Last updated：2024-10-09 16:55:14

Video Playback License

Configuration

Before you call the APIs of the Player SDK, follow the steps below to configure the license:

iOS

Add the code below in `[AppDelegate application:didFinishLaunchingWithOptions:]`:

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    NSString * const licenseURL = @"<The license URL obtained>";
    NSString * const licenseKey = @"<The key obtained>";

    //TXLiveBase can be found in the "TXLiveBase.h" header file
    [TXLiveBase setLicence:licenseURL key:licenseKey];
    [TXLiveBase sharedInstance].delegate = self;
    NSLog(@"SDK Version = %@", [TXLiveBase getSDKVersionStr]);
    return YES;
}

#pragma mark - TXLiveBaseDelegate
- (void)onLicenceLoaded:(int)result Reason:(NSString *)reason {
    NSLog(@"onLicenceLoaded: result:%d reason:%@", result, reason);
}
@end
```

Android

Add the code below in `application`:

```
public class MApplication extends Application {

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

```
                }
            });
        }
    }
```

Note：

1. License is a strong online verification logic. When calling TXLiveBase#setLicence after the application is started for the first time, the network must be ensured. When the App is launched for the first time, the Internet access permission may not have been authorized. You need to wait for the Internet access permission to be granted before calling TXLiveBase#setLicence again.
2. Monitor the TXLiveBase#setLicence loading result: onLicenceLoaded interface. If it fails, retry and guide accordingly according to the actual situation. If it fails multiple times, the frequency can be limited, and the service can be supplemented with product pop-up windows and other guidance to allow users to check the network situation.
3. TXLiveBase#setLicence can be called multiple times. It is recommended to call TXLiveBase#setLicence when entering the main interface of the App to ensure successful loading.
4. For multi-process apps, ensure that TXLiveBase#setLicence is called when each process using the player starts. For example: For an Android app that uses an independent process to play videos, if the process is killed by the system and restarted during background playback, TXLiveBase#setLicence must also be called.

Viewing license information

After the license is successfully configured, you can call the API below to view the license information. Please note that it may take a while for the configuration to take effect. The exact time needed depends on your network conditions.

iOS:

```
NSLog(@"%@", [TXLiveBase getLicenceInfo]);
```

Android:

```
TXLiveBase.getInstance().getLicenceInfo();
```

Player Guide
Stage 1. Play back a source video

Last updated：2023-06-19 16:19:01

Overview

This document describes how to upload a video to VOD and play it using VOD's player.

Prerequisites

Before you start, do the following:

Activating VOD

Follow the steps below to activate VOD:

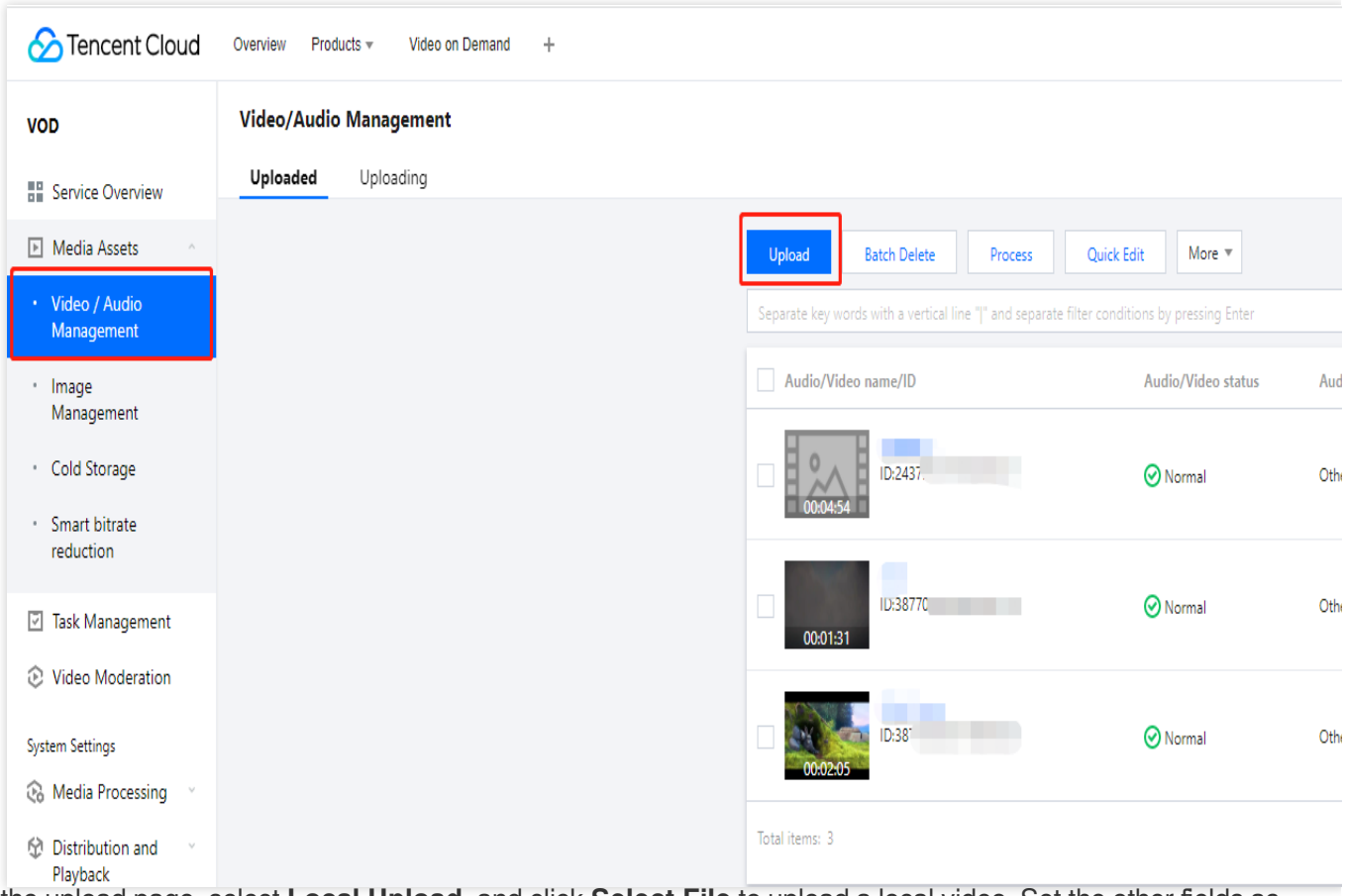
1. Sign up for a Tencent Cloud account.
2. Purchase VOD services. For details, see Billing Overview.
3. Go to the **VOD console**.

At this point, you have activated VOD.

Step 1. Upload a Video

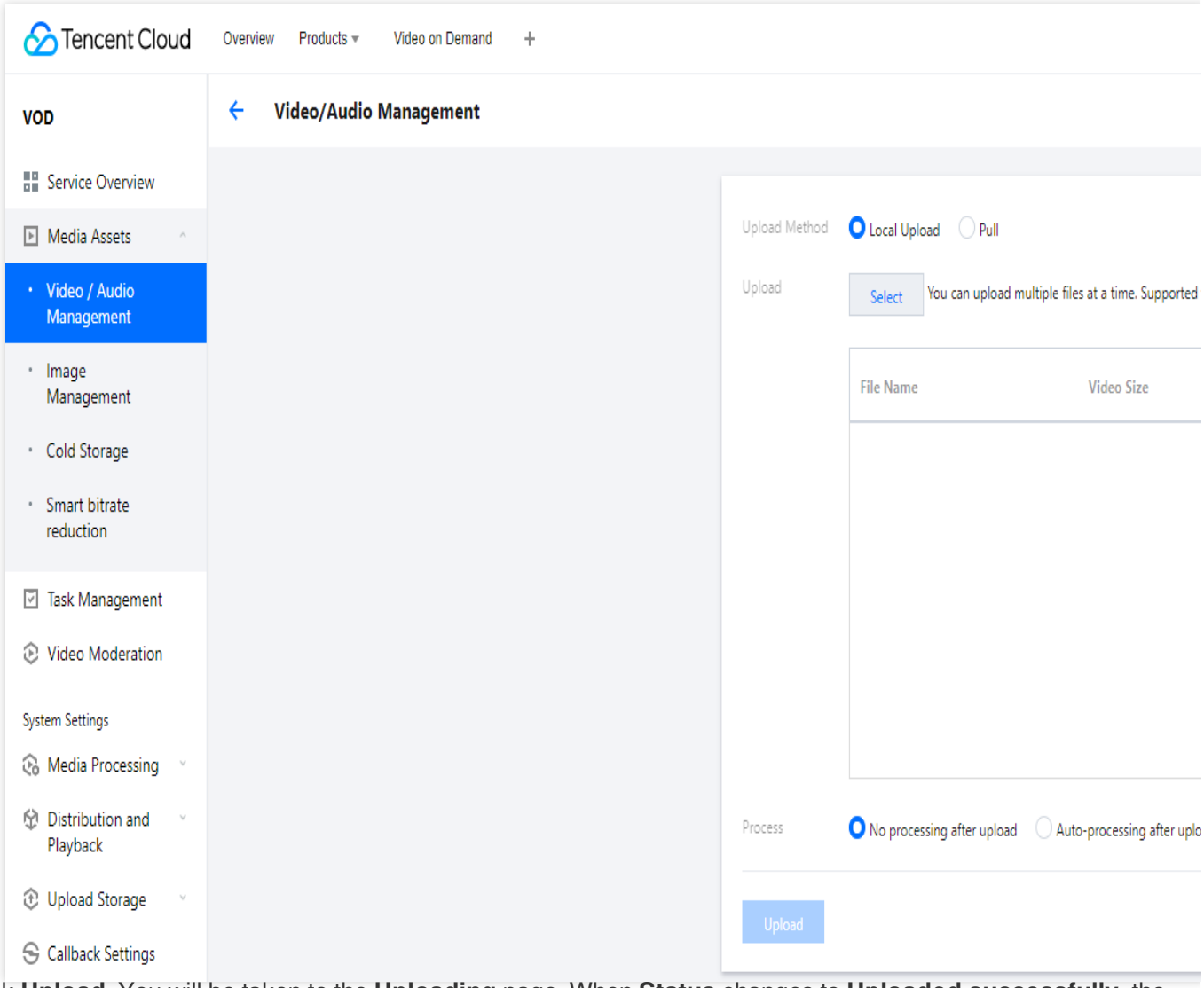
This step shows you how to upload a video.

1. In the VOD console, select Application Management on the left sidebar and select the target application. On the **Media Assets > Video/Audio Management** page, click **Upload**.

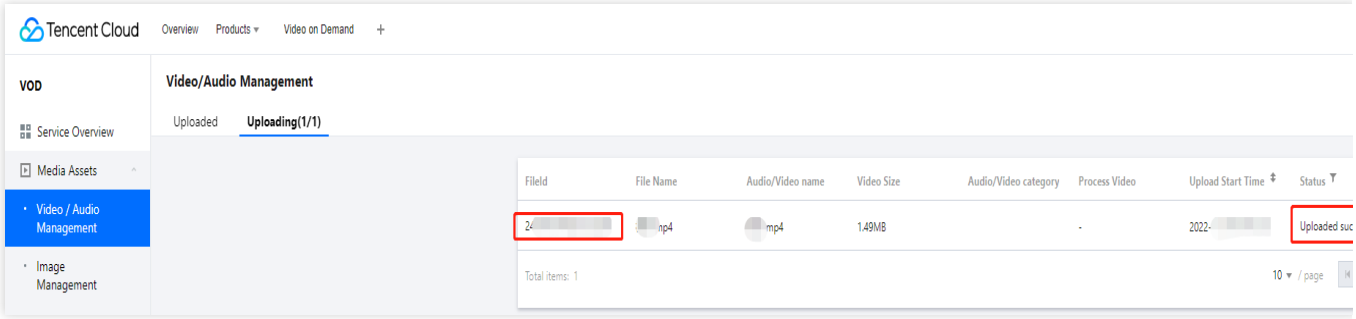


2. On the upload page, select **Local Upload**, and click **Select File** to upload a local video. Set the other fields as follows:

Select **No processing after upload** for **Media processing**.



3. Click **Upload**. You will be taken to the **Uploading** page. When **Status** changes to **Uploaded successfully**, the upload is completed, and you can view the **file ID** of the file (for example, 387xxxxx8142975036).



Step 2. Generate a Player Signature

In this step, you use the signature tool to quickly generate a player signature to play the video.

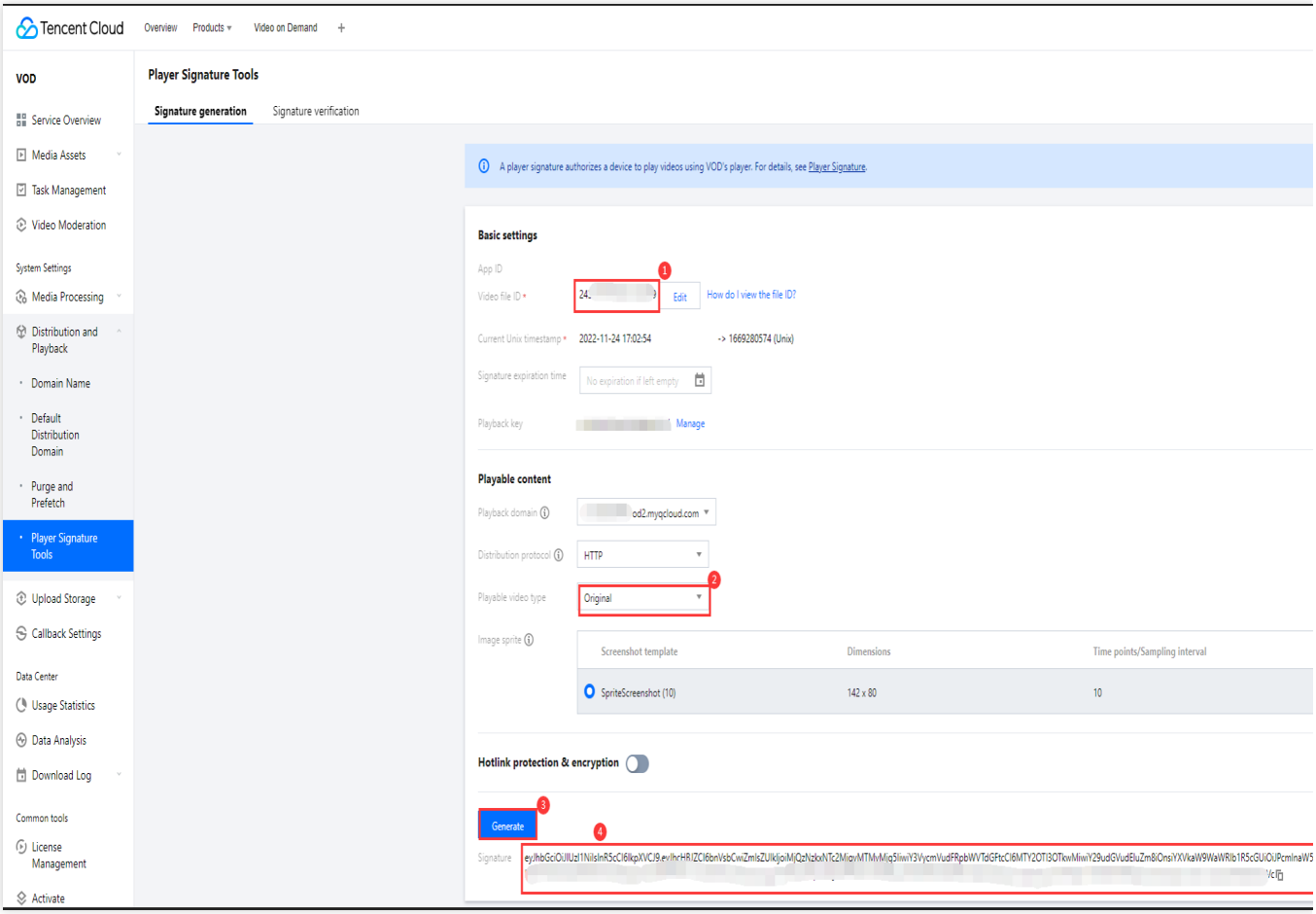
1. Select **Distribution and Playback > Player Signature Tools** on the left sidebar and complete the following settings:

Video file ID: Enter the file ID generated in Step 1 (387xxxxx8142975036).

Signature expiration time: Enter the player signature expiration time. If you leave it empty, the signature will never expire.

Playable video type: Select **Original**.

2. Click **Generate** to get the signature string.



Step 3. Play the Video

After step 2, you have obtained the three parameters needed for video playback: `appId`, `fileId` and `psign` (player signature). The following describes how to play the video on the web.

Playback on the web

1. Open the web player demo.

Select **Video playback**.

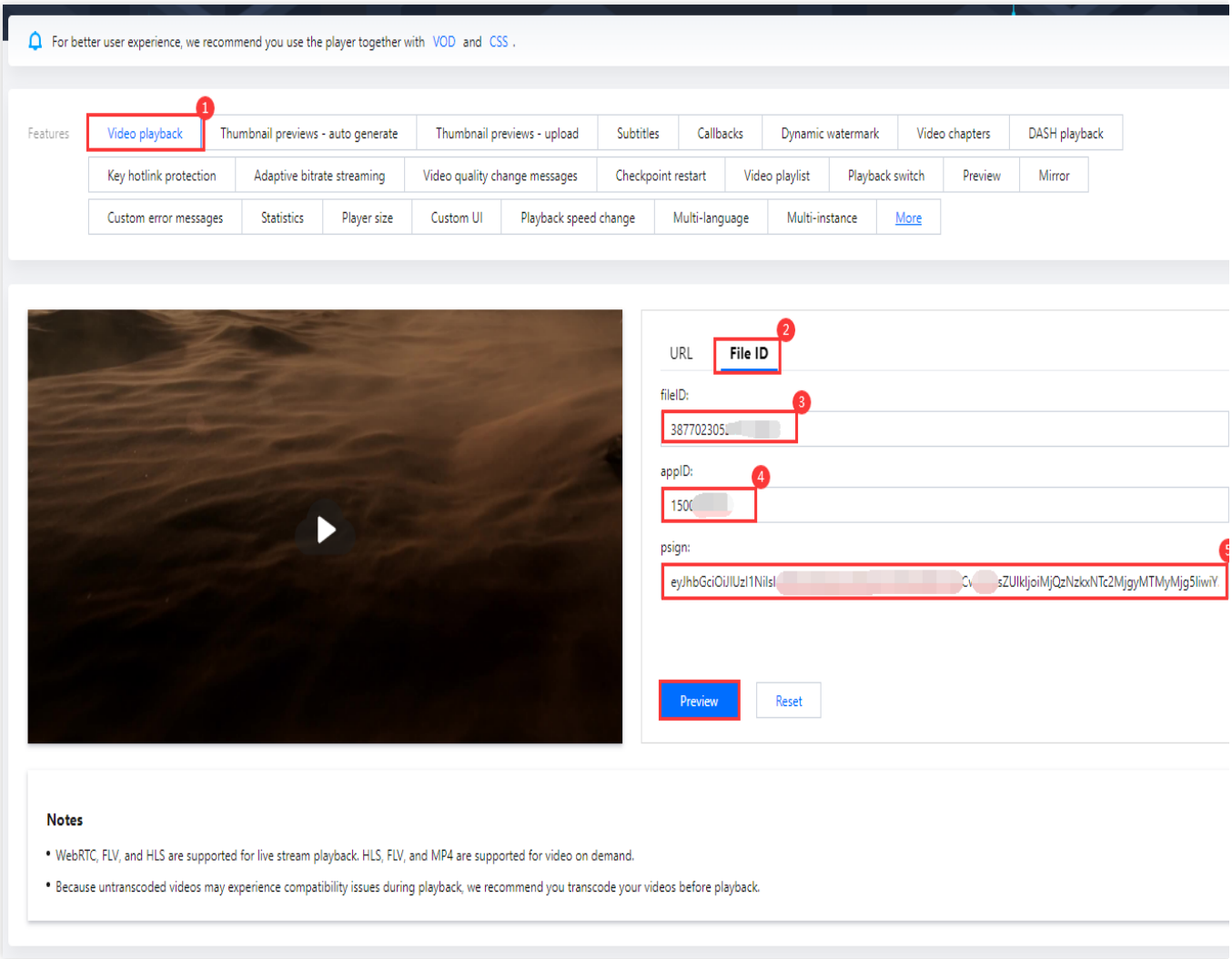
Click the **File ID** tab.

fileID: Enter the same file ID (387xxxxx8142975036) in the previous step.

appID: Enter the ID of the VOD application to which the file belongs (which is also the App ID displayed on the signature generation page in the previous step).

psign: Enter the signature string generated in the previous step.

2. Click **Preview** to play the video.



Multi-platform player demos

After generating the player signature, you can use our player demos for web, Android, and iOS to play the video. For details, see the source code for the demos.

Summary

At this point, you have understood how to upload a video to VOD and play it back in the player.

See also:

Stage 2. Play back a transcoded video.

Stage 3. Play back an adaptive bitrate streaming video.

Stage 4. Play back an encrypted video

Stage 5. Play back a long video

Stage 2. Play back a transcoded video

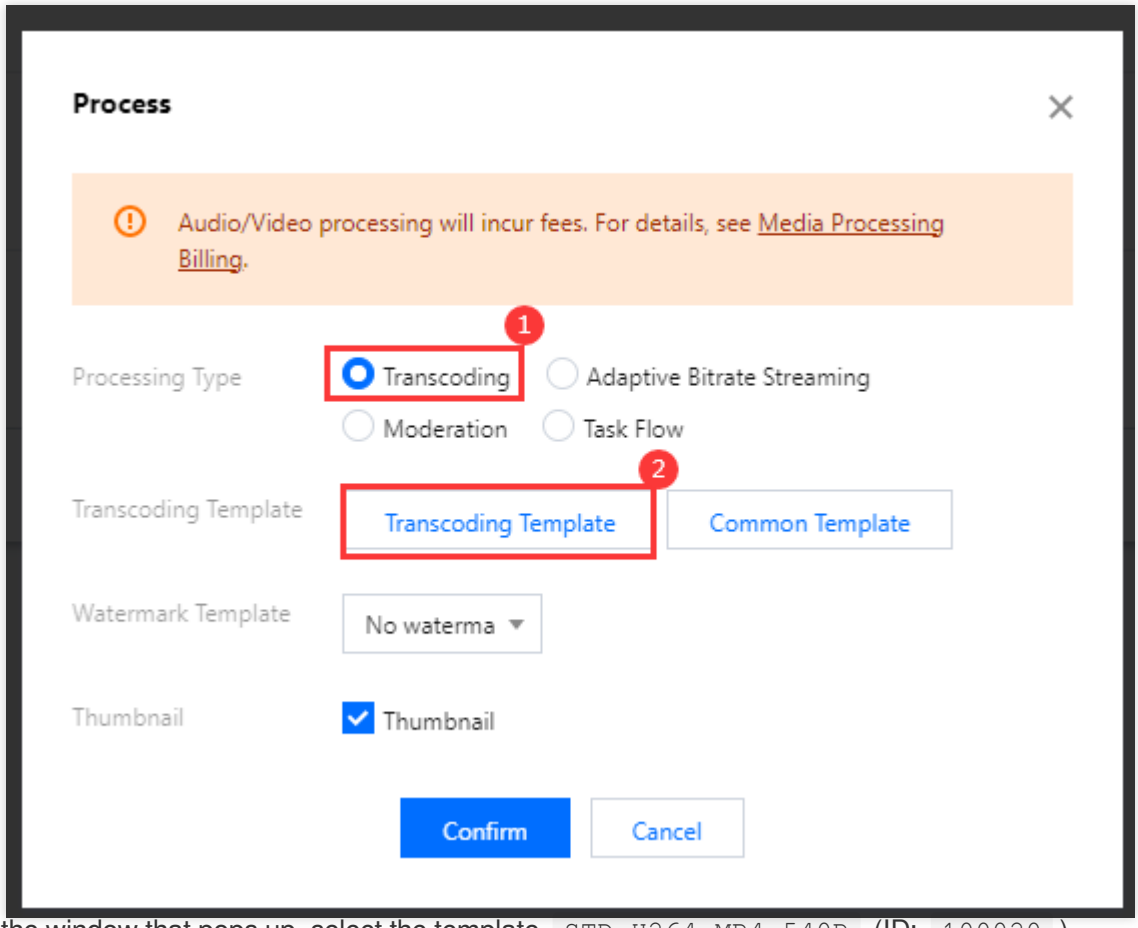
Last updated：2023-05-15 17:35:16

Overview

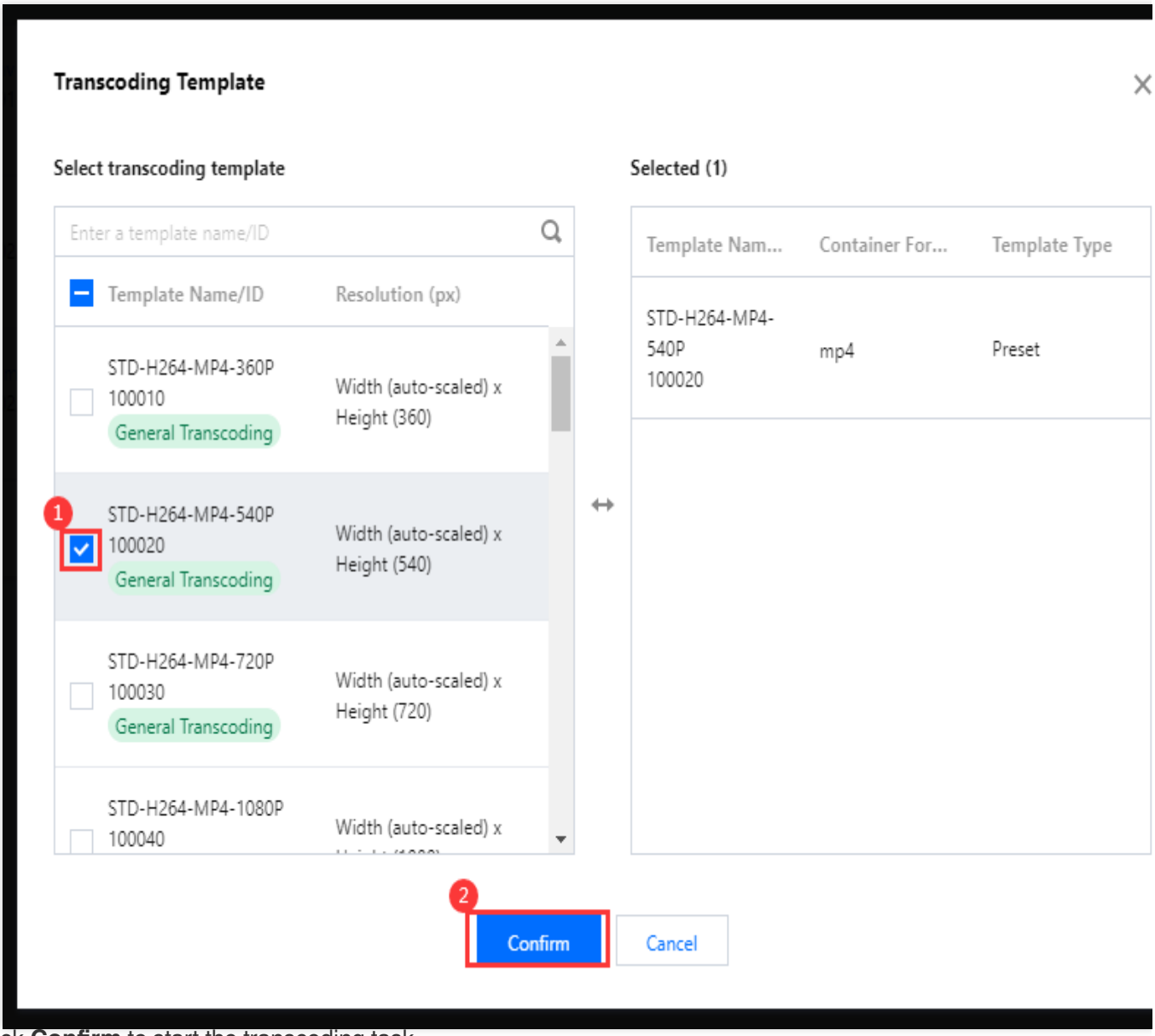
This document describes how to transcode a video and use the player to play the transcoded video.
Before you go on, make sure you have read Stage 1. Play back a source video. You will be using the account registered and the video uploaded in stage 1.

Step 1. Encrypt a Video

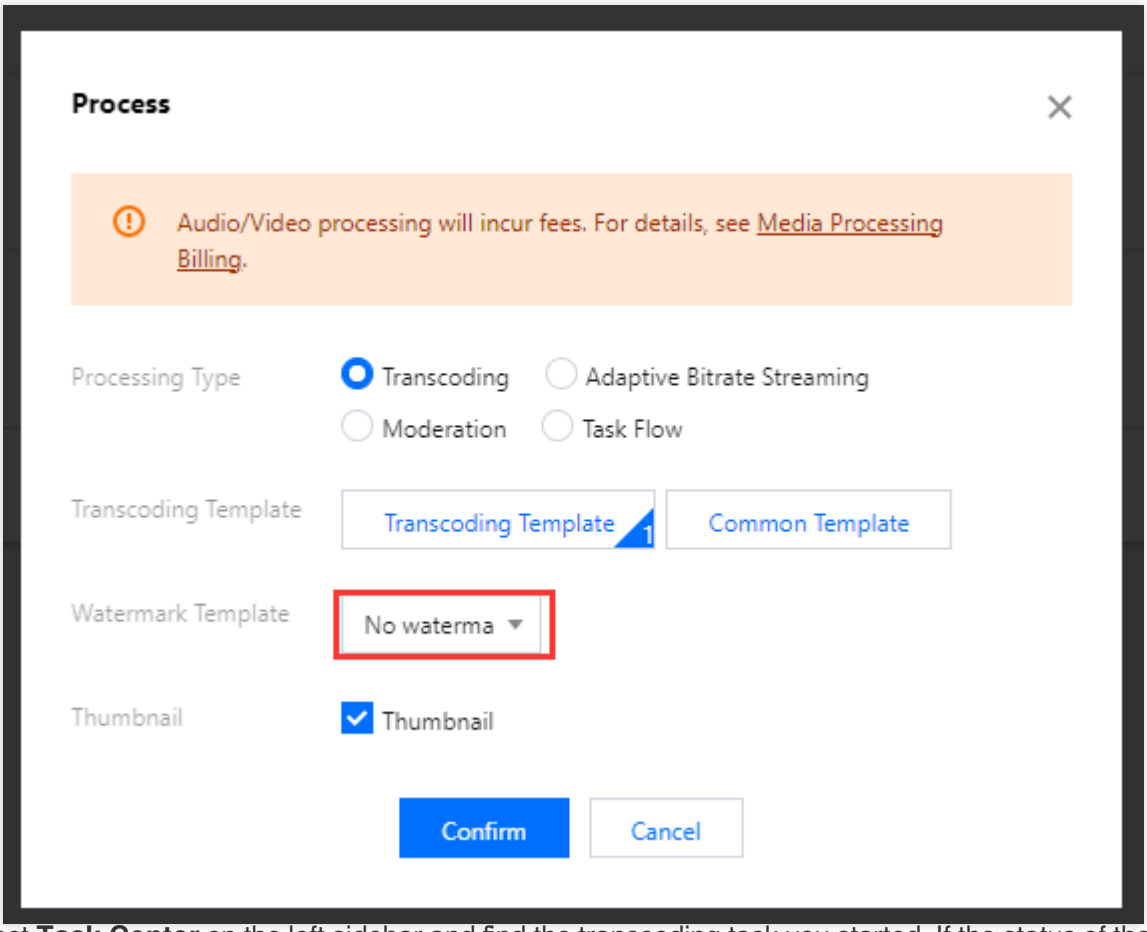
1. In the VOD console, select **Application Management** on the left sidebar and select the target application. On the **Media Assets > Video/Audio Management** page, select the target video (file ID 387xxxxx8142975036), and click **Transcoding**.
2. Complete the following settings:
2.1 Select **Transcoding** for **Processing Type**.
2.2 Click **Select template** to select a transcoding template.



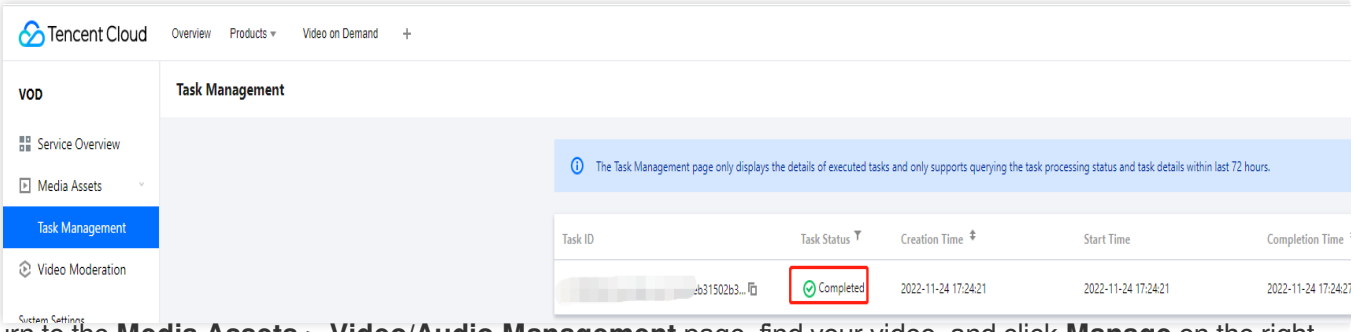
2.3 In the window that pops up, select the template `STD-H264-MP4-540P` (ID: `100020`).



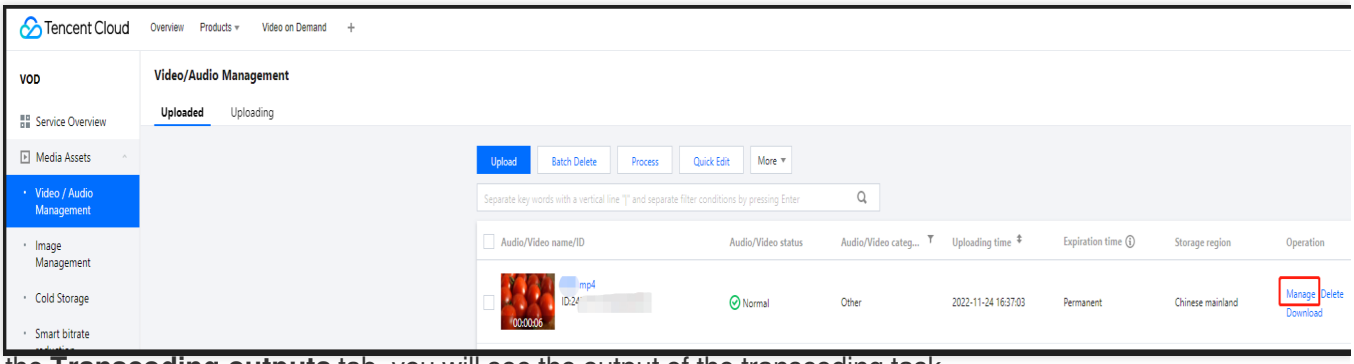
2.4 Click **Confirm** to start the transcoding task.



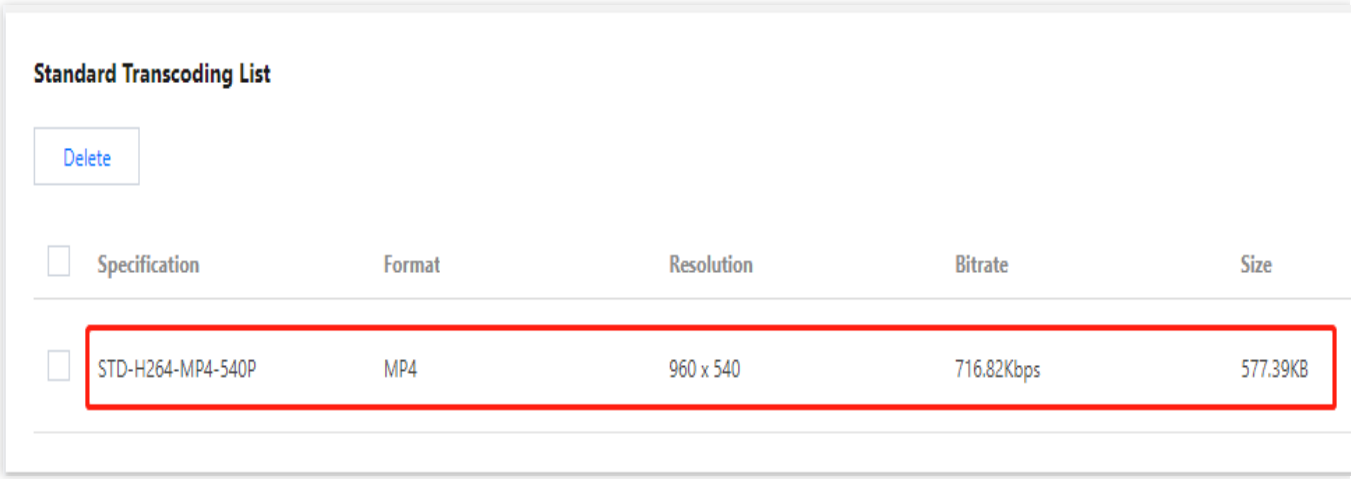
3. Select **Task Center** on the left sidebar and find the transcoding task you started. If the status of the task has changed from "Processing" to "Completed", the video has been transcoded.



4. Return to the **Media Assets > Video/Audio Management** page, find your video, and click **Manage** on the right.



Under the **Transcoding outputs** tab, you will see the output of the transcoding task.



Step 2. Generate a Player Signature

In this step, you use the signature tool to quickly generate a player signature to play the video.

1. Select **Distribution and Playback** > **Player Signature Tools** on the left sidebar and complete the following settings:

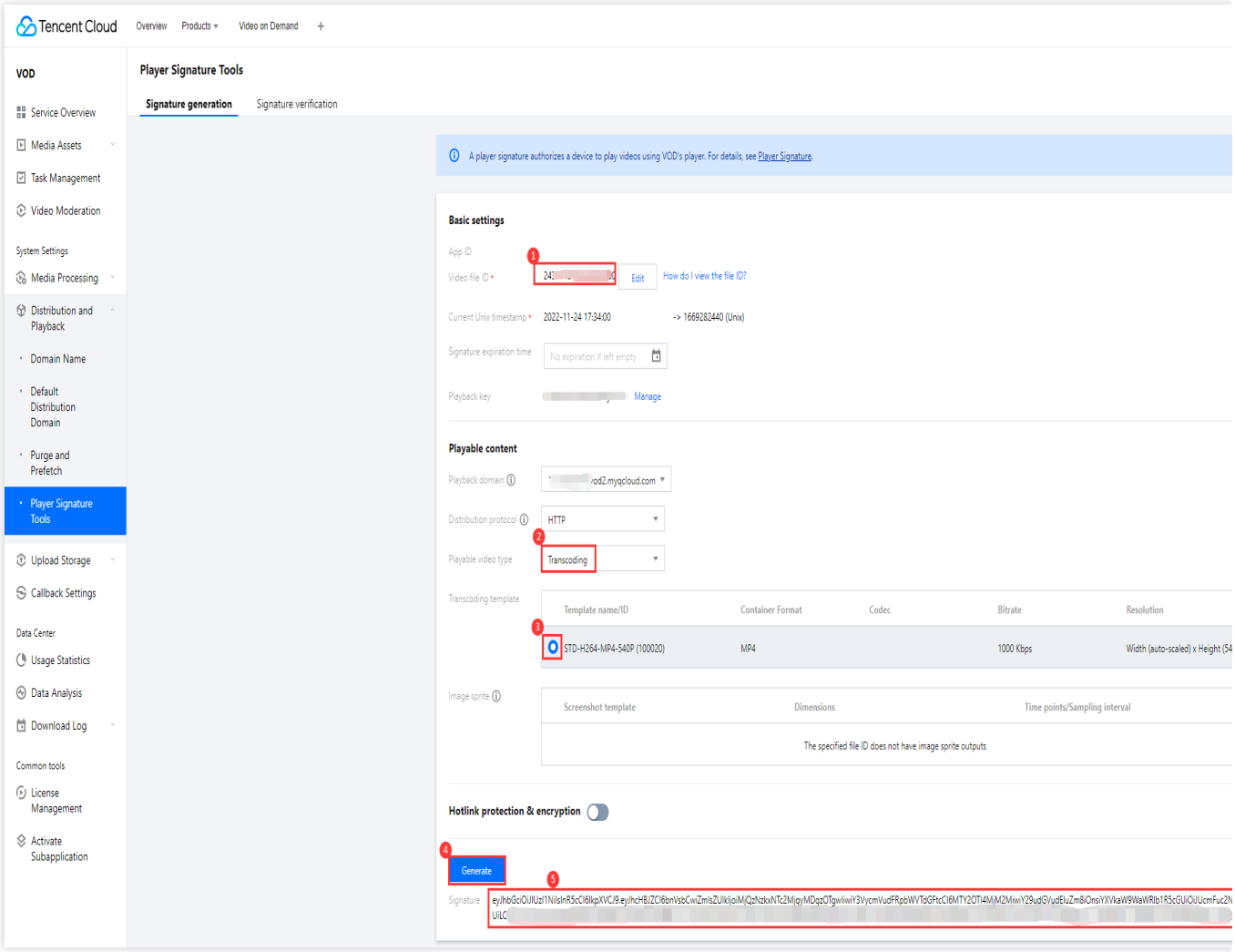
Video file ID: Enter the file ID generated in Step 1 (387xxxxx8142975036).

Signature expiration time: Enter the player signature expiration time. If you leave it empty, the signature will never expire.

Playable video type: Select **Transcoding**.

Transcoding template: Select `STD-H264-MP4-540P (100020)` .

2. Click **Generate** to get the signature string.



Step 3. Play the Video

After step 2, you have obtained the three parameters needed for video playback: `appId` , `fileId` and `psign` (player signature). The following describes how to play the video on the web.

Playback on the web

1. Open the web player demo.

Select **Video playback**.

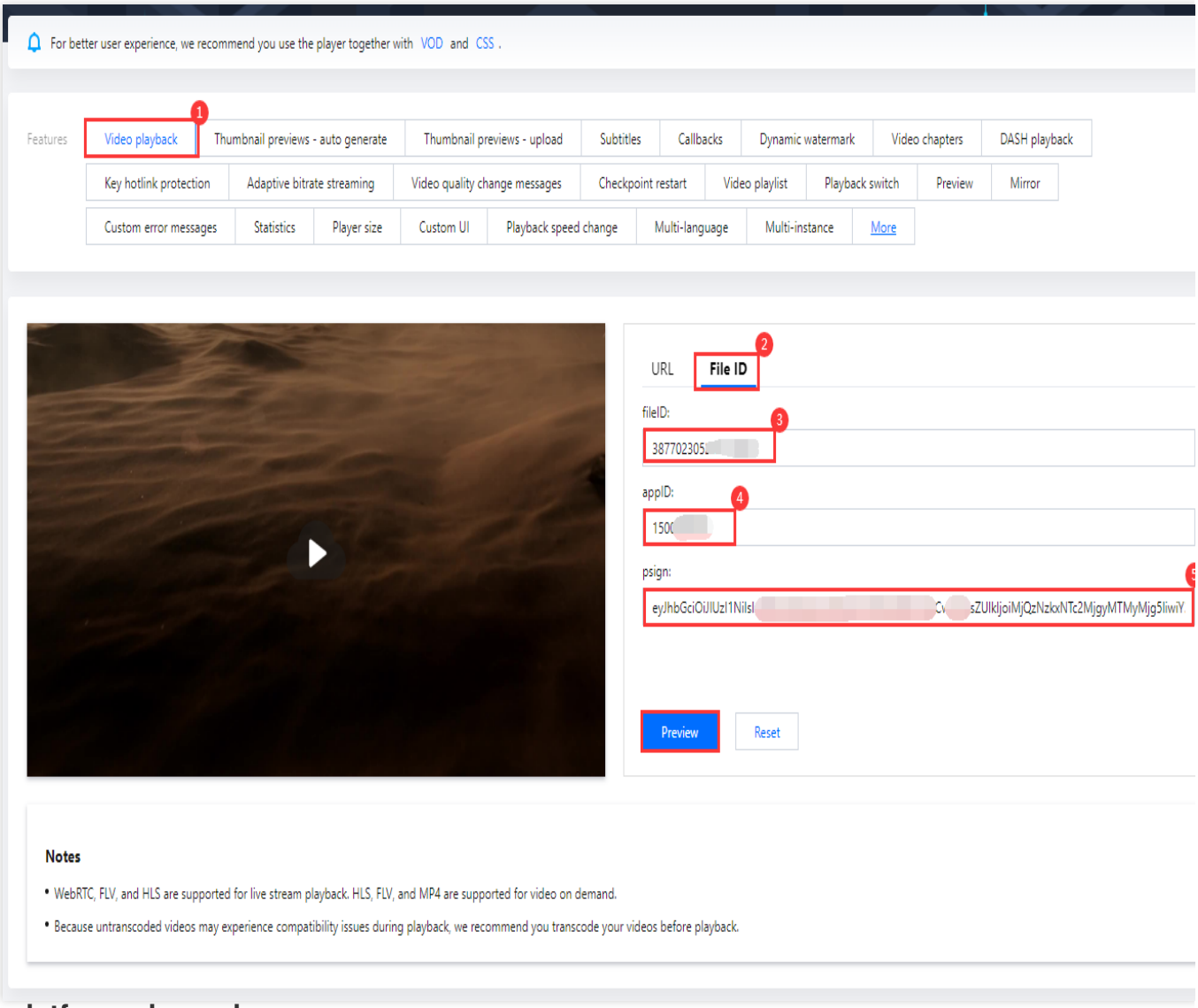
Click the **File ID** tab.

fileID: Enter the same file ID in the previous step.

appID: Enter the ID of the VOD application to which the file belongs (which is also the App ID displayed on the signature generation page in the previous step).

psign: Enter the signature string generated in the previous step.

2. Click **Preview** to play the video.



Multi-platform player demos

After generating the player signature, you can use our player demos for web, Android, and iOS to play the video. For details, see the source code for the demos.

Summary

At this point, you have understood how to transcode a video and use the player to play it.

Stage 3. Play back an adaptive bitrate streaming video

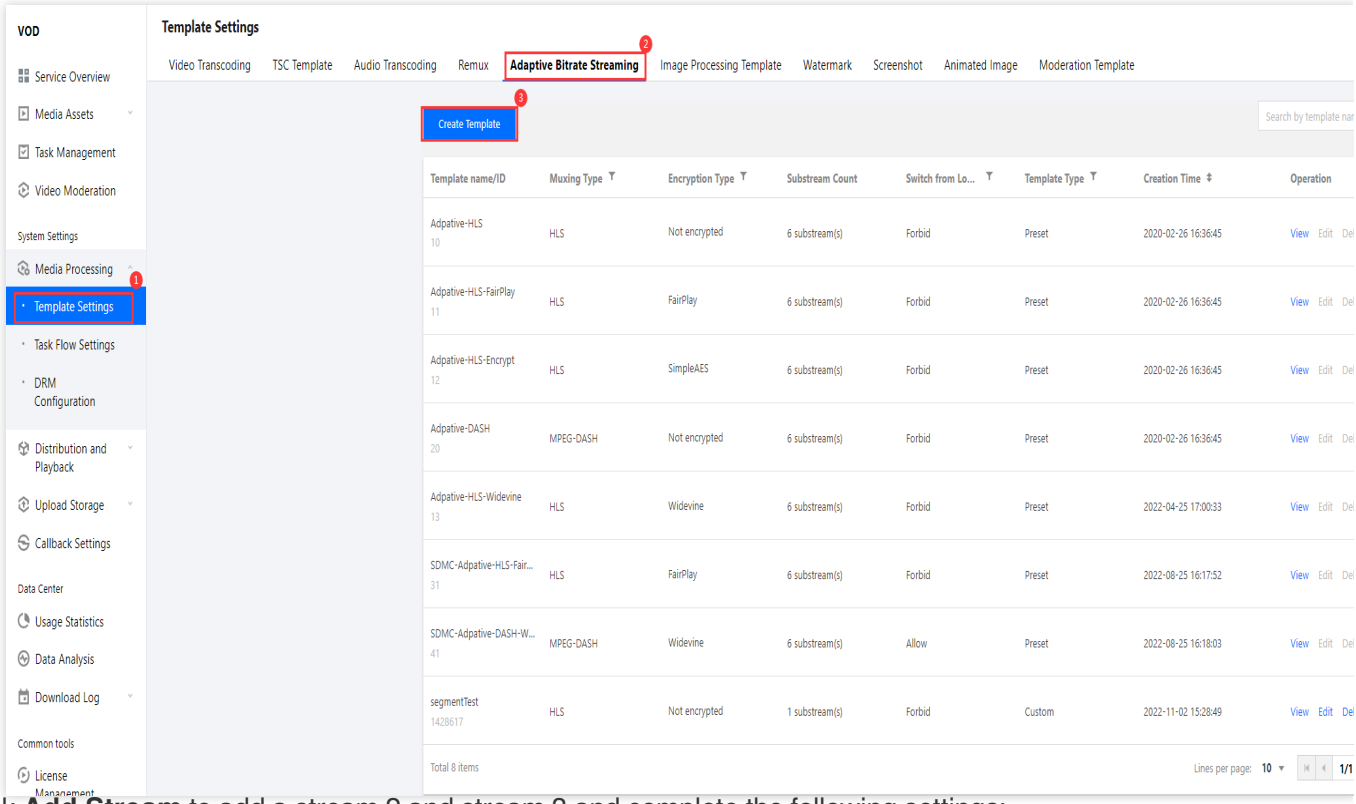
Last updated：2023-05-15 17:35:41

Overview

This document describes how to play an adaptive bitrate streaming video. Specifically, the following will be performed:
Streams of different resolutions will be generated. The lowest will be 480p and the highest will be 1080p.
A screenshot will be taken from the video and will be used as the thumbnail.
Multiple screenshots will be taken at an interval of 20% and will be used as thumbnail previews.
Before you go on, make sure you have read Stage 1. Play back a source video. You will be using the account registered and the video uploaded in stage 1.

Step 1. Create an Adaptive Bitrate Streaming Template

1. Log in to the VOD console. Select Application Management on the left sidebar. Select the target application and go to **Media Processing > Template Settings**. Under the **Adaptive Bitrate streaming** tab, click **Create Template**.



2. Click **Add Stream** to add a stream 2 and stream 3 and complete the following settings:

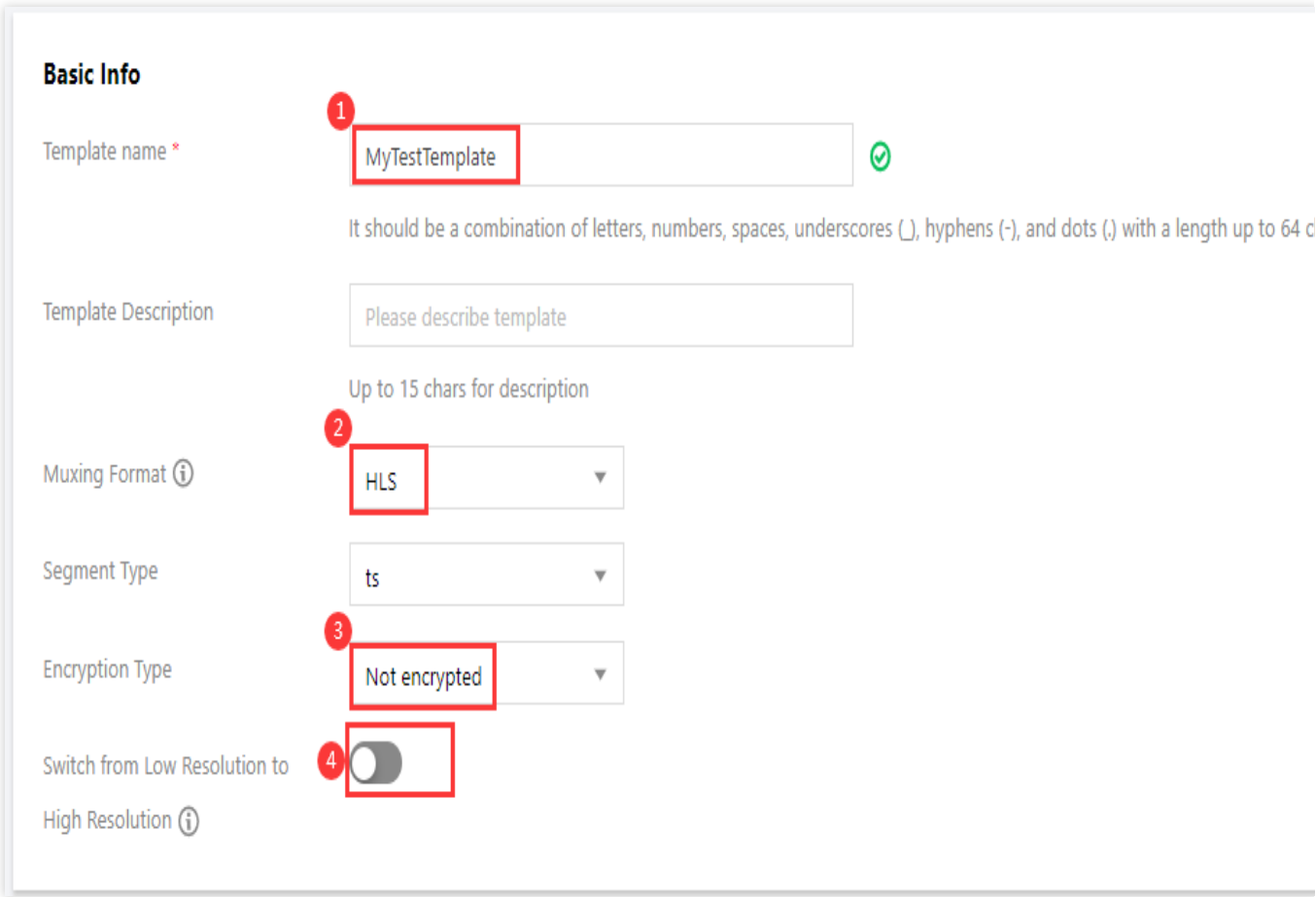
Basic Info:

Template name: Type `MyTestTemplate` .

Muxing format: Select "HLS".

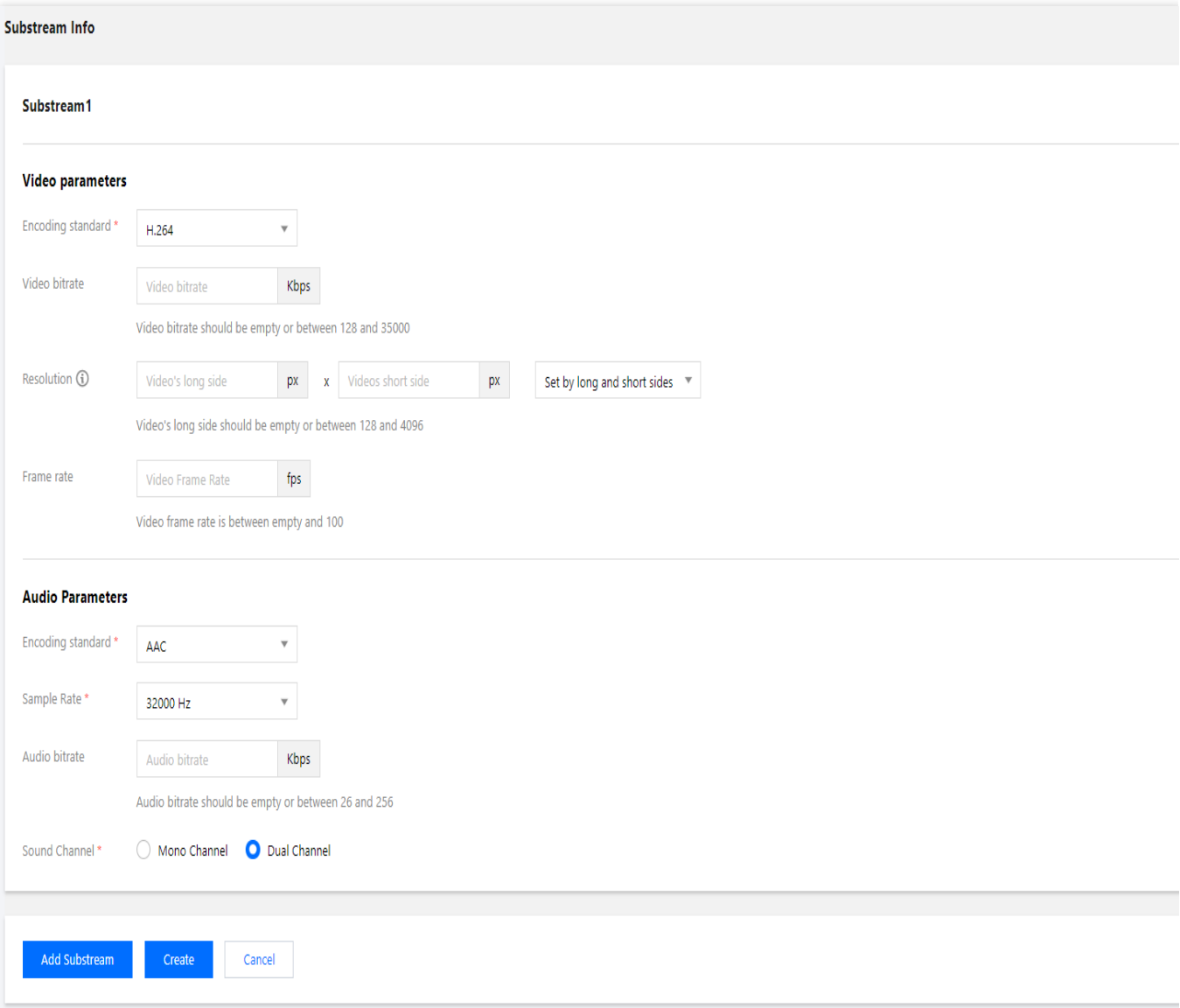
Encryption type: Select "Not encrypted".

Switch from Low Resolution to High Resolution: Disable

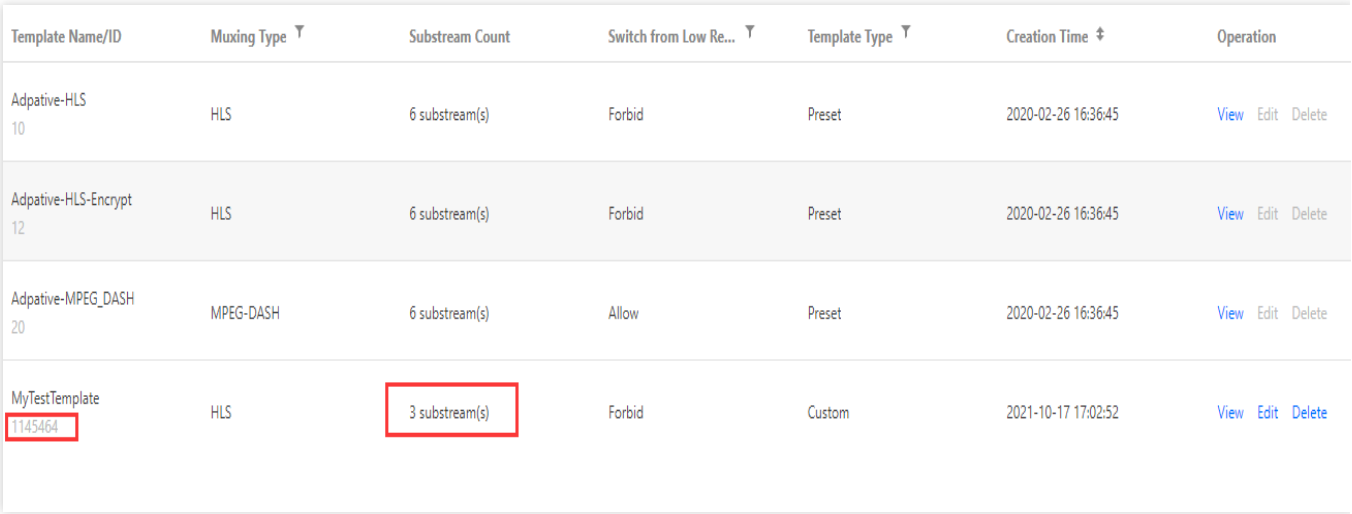


Streams

Stream No.	Video Bitrate	Resolution	Frame Rate	Audio Bitrate	Sound Channels
Stream 1	512 Kbps	Long side: 0 px, short side: 480 px.	24 fps	48 Kbps	Dual
Stream 2	512 Kbps	Long side: 0 px, short side: 720 px.	24 fps	48 Kbps	Dual
Stream 3	1,024 Kbps	Long side: 0 px, short side: 1080 px.	24 fps	48 Kbps	Dual

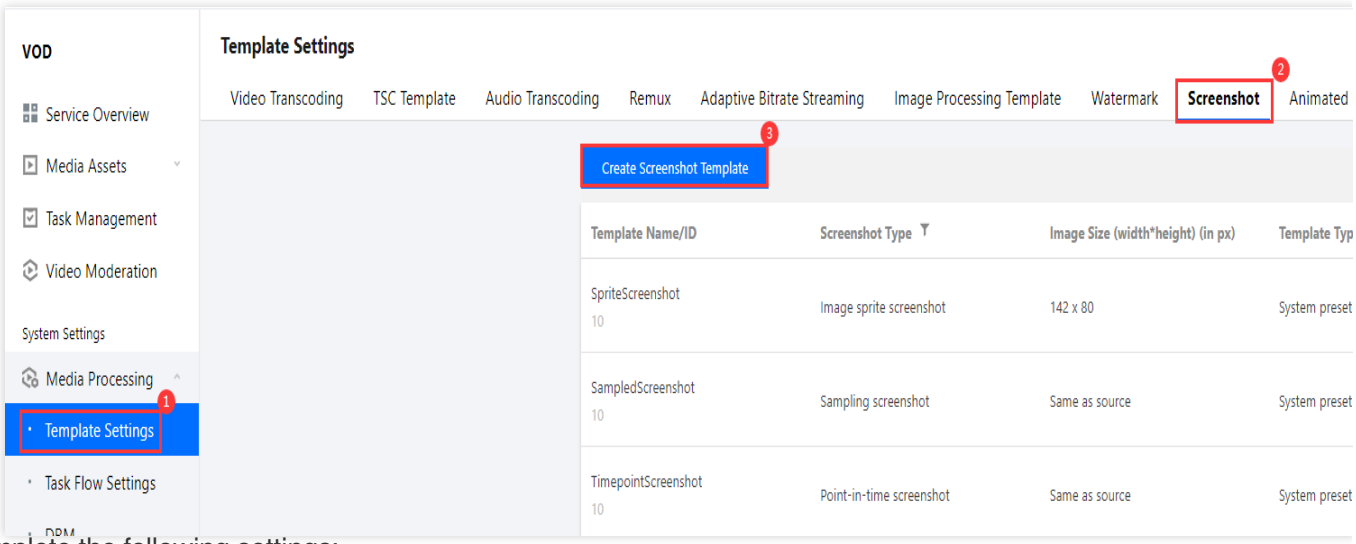


3. Click **Create**. An adaptive streaming template that converts a video into three streams is created. The ID of the template is `1430219`.



Step 2. Create an Image Sprite Template

1. Go to **Media Processing** > **Template Settings**. Select the **Screenshot** tab and click **Create Screenshot Template**.



2. Complete the following settings:

Template name: Type `MyTestTemplate` .

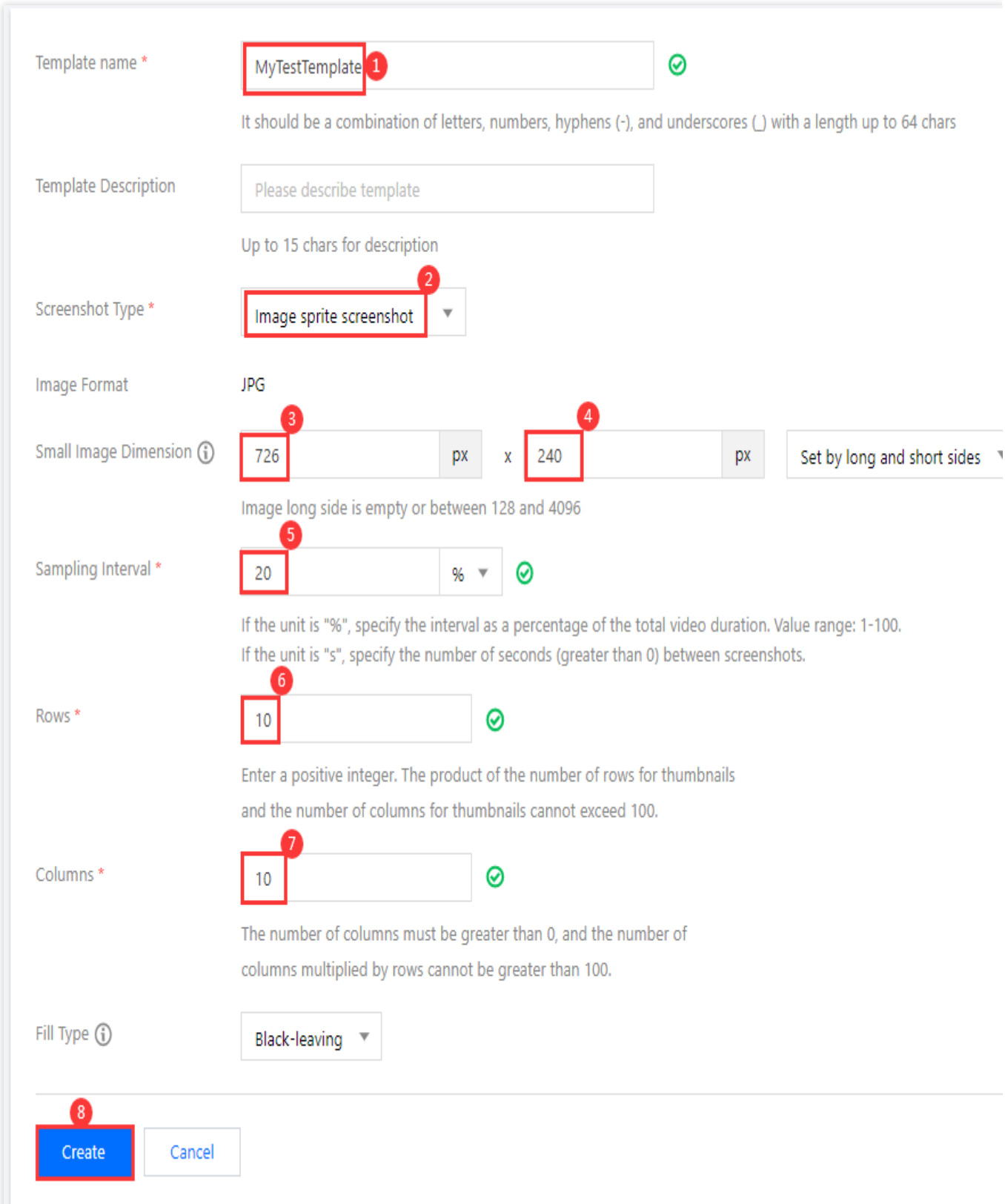
Screenshot type: Select "Image sprite screenshot".

Small image dimension: 726 px x 240 px.

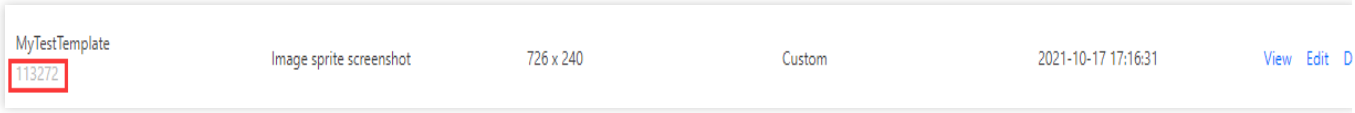
Sampling interval: 20%.

Rows: 10.

Columns: 10.



3. Click **Create**. An image sprite template with the ID 131864 is created.



Step 3. Create and Start a Task Flow

Now you have an adaptive bitrate streaming template (ID 1430219) and an image sprite template (ID 131864). You also need to create a task flow.

1. Go to **Media Processing > Task Flow Settings** and click **Create Task Flow**.

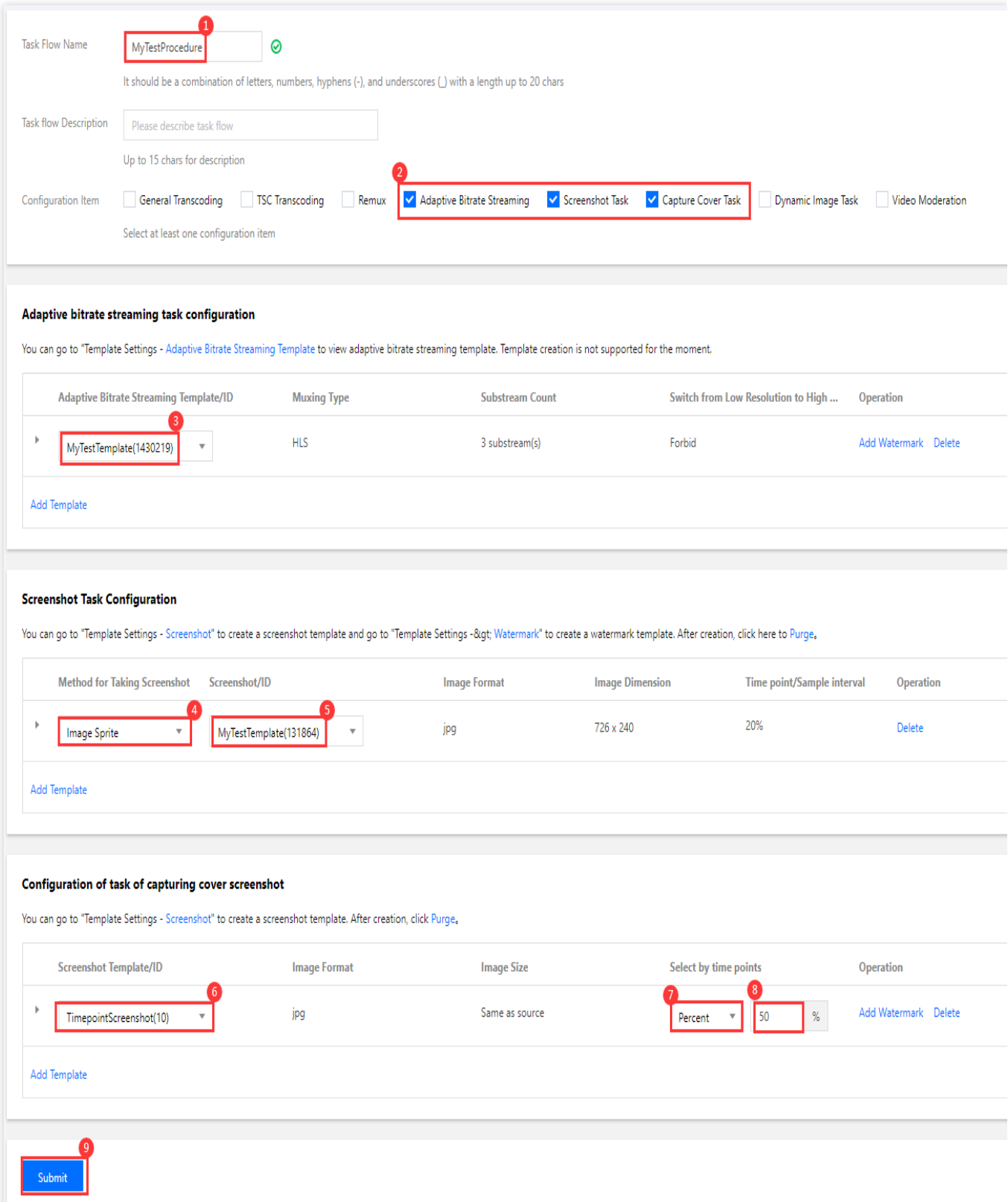
Task flow name: Enter `MyTestProcedure` .

Configuration item: Select "Adaptive bitrate streaming", "Screenshot", and "Thumbnail generation".

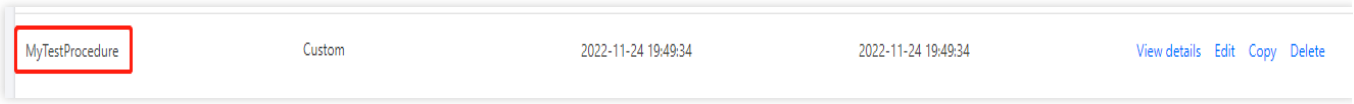
In the **Adaptive bitrate streaming task configuration** area, click **Add Template**, and select the `MyTestTemplate` template (ID 1430219) created in **step 1**.

In the **Screenshot task configuration** area, click **Add Template**. Select **Image sprite** for **Method for Taking Screenshot** and then select the `MyTestTemplate` template (ID 131864) created in **step 2**.

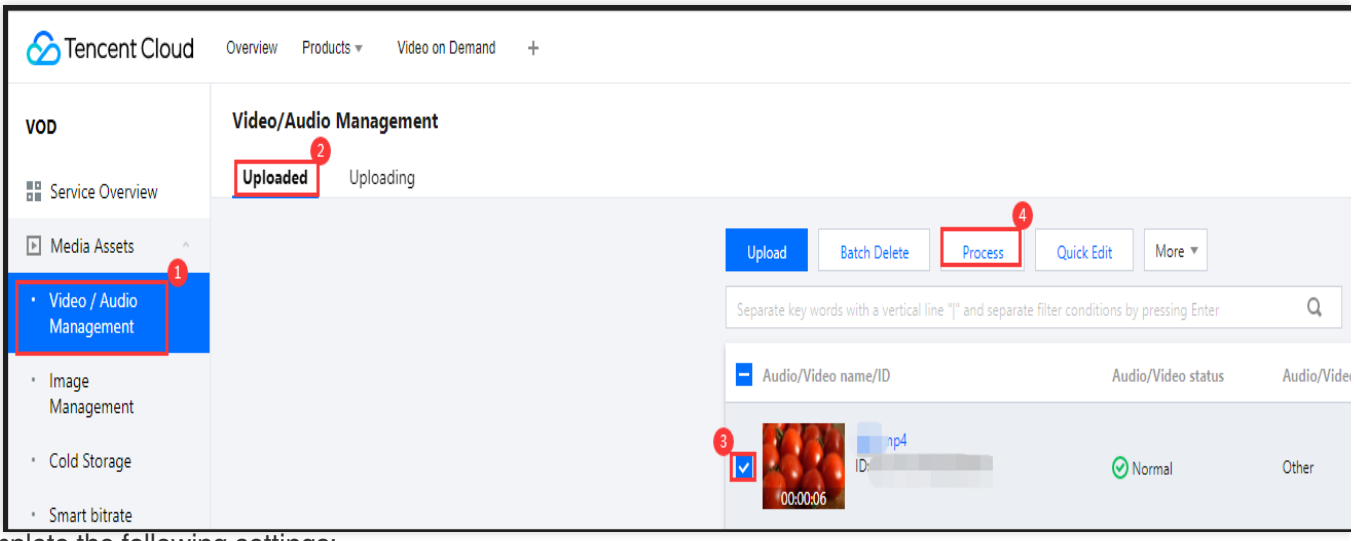
In the **Configuration of task of capturing cover screenshot** area, click **Add Template**. For **Screenshot Template/ID**, select "TimepointScreenshot". For **Select by time points**, select "Percent", and enter "50".



2. Click **Submit**. A task flow named `MyTestProcedure` will be generated.



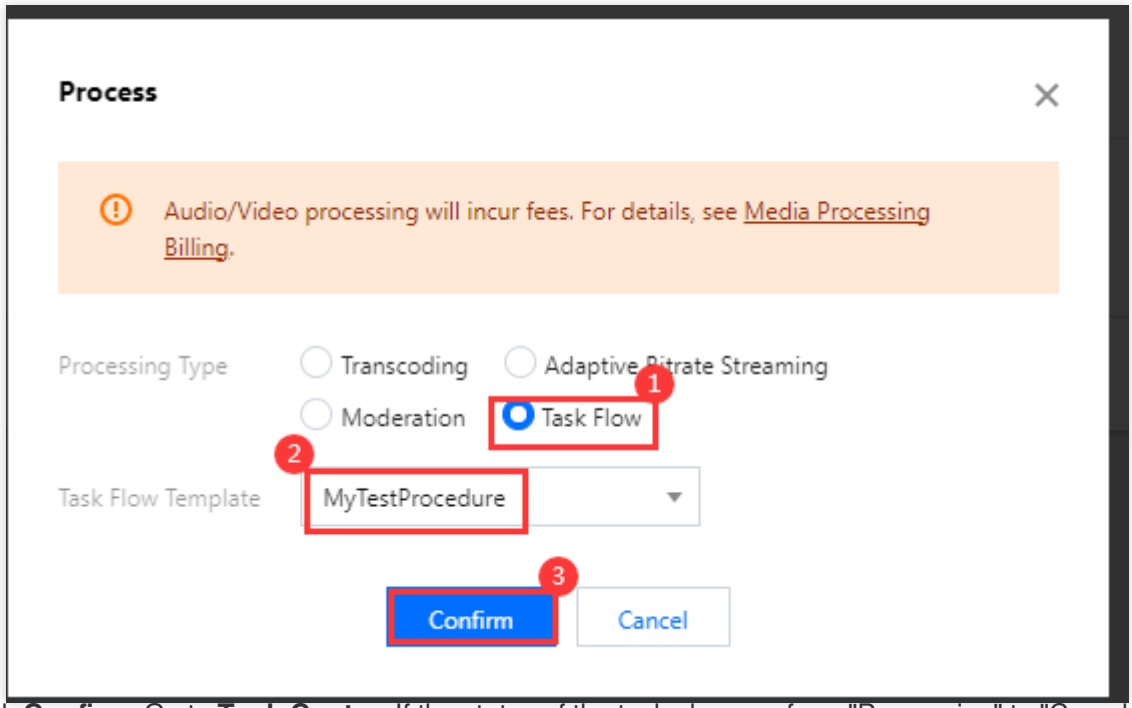
3. Go to **Media Assets > Video/Audio Management**, select the target video (file ID 243xxx814xxxxx416), and click **Task Flow**.



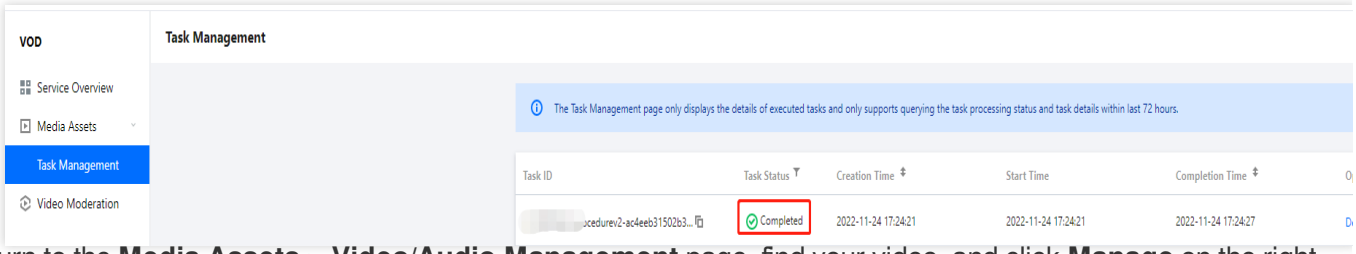
4. Complete the following settings:

Select **Task Flow** as the **Processing Type**.

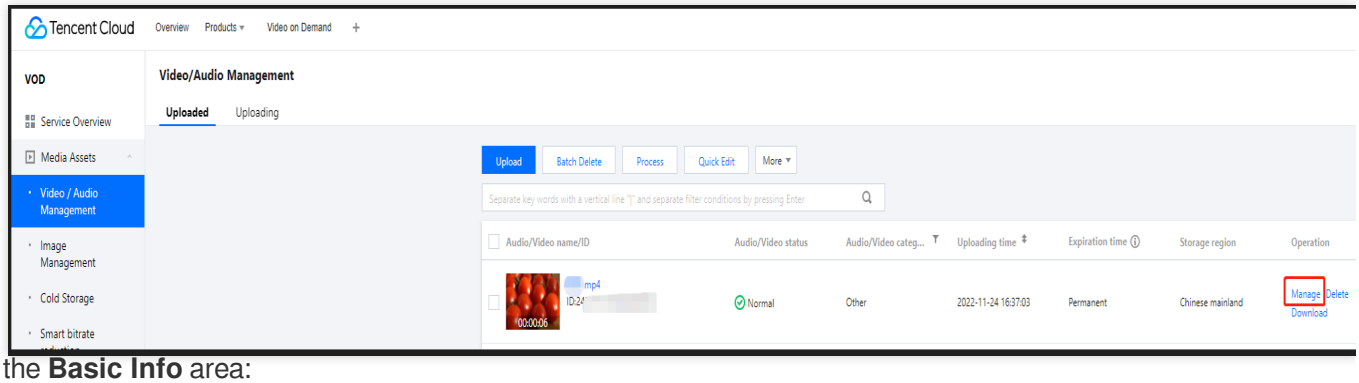
Select the "MyTestProcedure" task flow template.



5. Click **Confirm**. Go to **Task Center**. If the status of the task changes from "Processing" to "Completed", the processing of the video is finished.

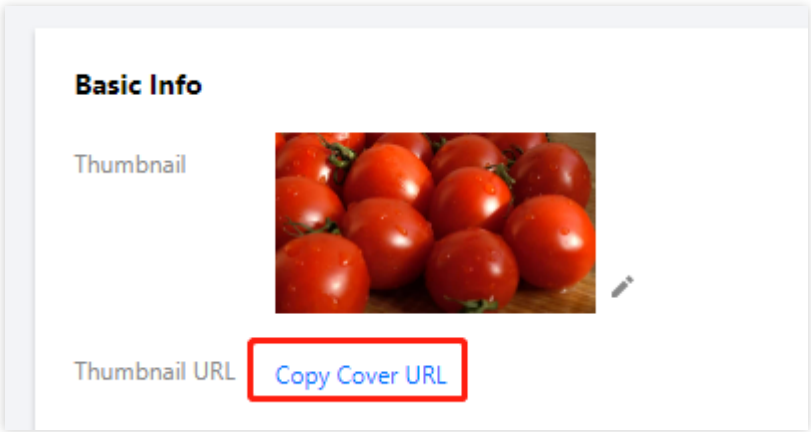


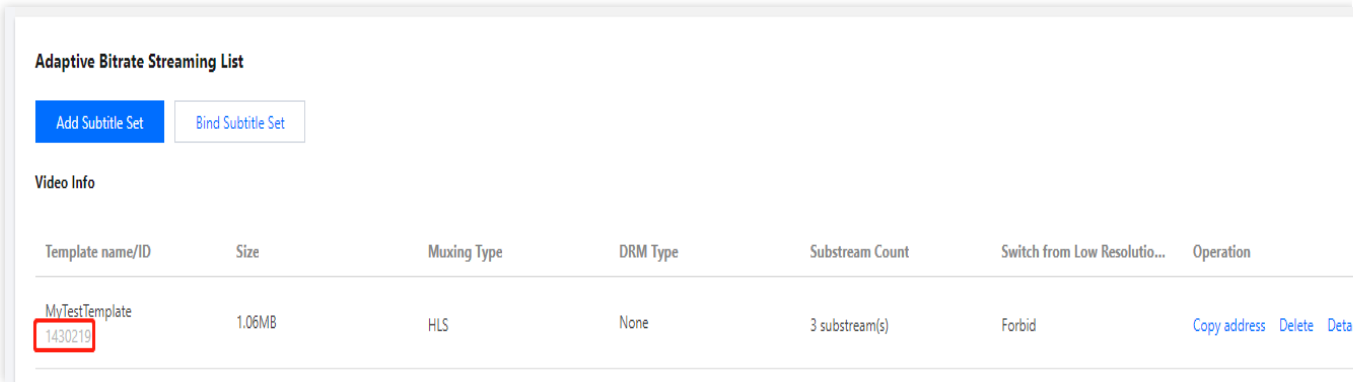
6. Return to the **Media Assets > Video/Audio Management** page, find your video, and click **Manage** on the right.



6.1 In the **Basic Info** area:

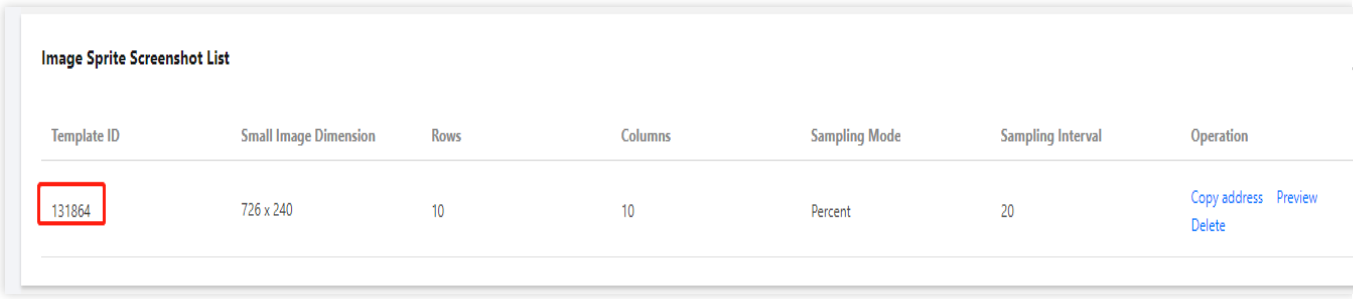
You can view the thumbnail generated and the outputs of adaptive bitrate streaming (template ID: 1430219).





6.2 Select the **Screenshots** tab:

You can view the image sprite generated (template ID: 131864).



Step 4. Generate a Player Signature

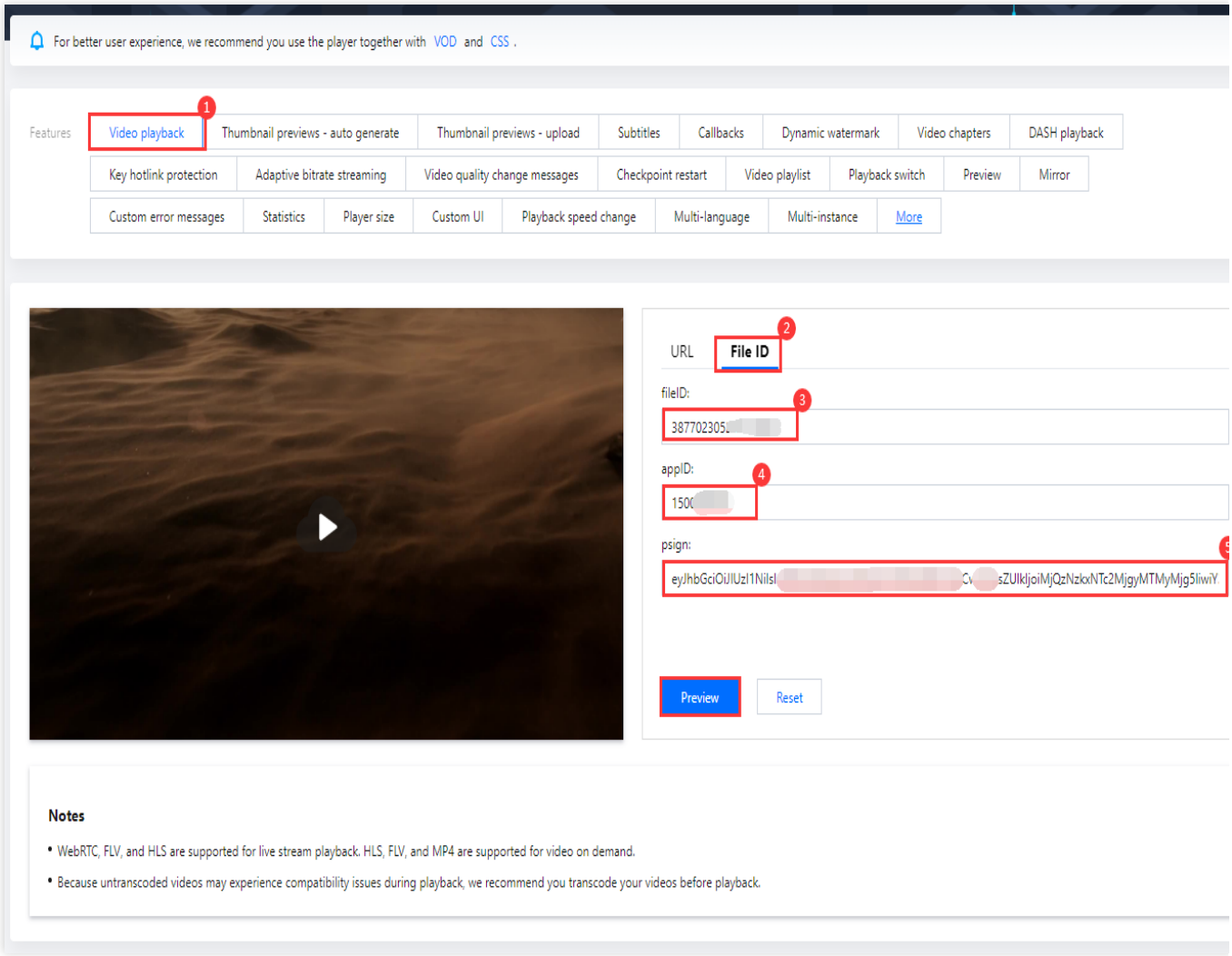
In this step, you can use the signature tool to quickly generate a signature for the player to play back the video.
Select **Distribution and Playback** > **Player Signature Tools** on the left sidebar and complete the following settings:
Video file ID: Enter the file ID (243xxx814xxxxx416) in step 3.
Signature expiration time: Enter the player signature expiration time. If you leave it empty, the signature will never expire.
Playable video type: Select **Unencrypted adaptive bitrate**.
Adaptive bitrate template: Select `MyTestTemplate (1430219)` .
Image sprite: Select `MyTestTemplate (131864)` .
Click **Generate** to get the signature string.

Step 5. Play the Video

After step 4, you have obtained the three parameters needed for video playback: `appId` , `fileId` and `psign` (player signature). The following describes how to play the video on the web.

Playback on the web

Open the web player demo.
Select **Video playback**.
Click the **File ID** tab.
fileID: Enter the same file ID (243xxx814xxxxx416) in the previous step.
appID: Enter the ID of the VOD application to which the file belongs (which is also the App ID displayed on the signature generation page in the previous step).
psign: Enter the signature string generated in the previous step.
Click **Preview** to play the video.



Multi-platform player demos

After generating the player signature, you can use our player demos for web, Android, and iOS to play the video. For details, see the source code for the demos.

Summary

At this point, you have understood how to play an adaptive bitrate streaming video.

To learn how to play an encrypted video, see Stage 4. Play back an encrypted video.

Stage 4. Play back an encrypted video

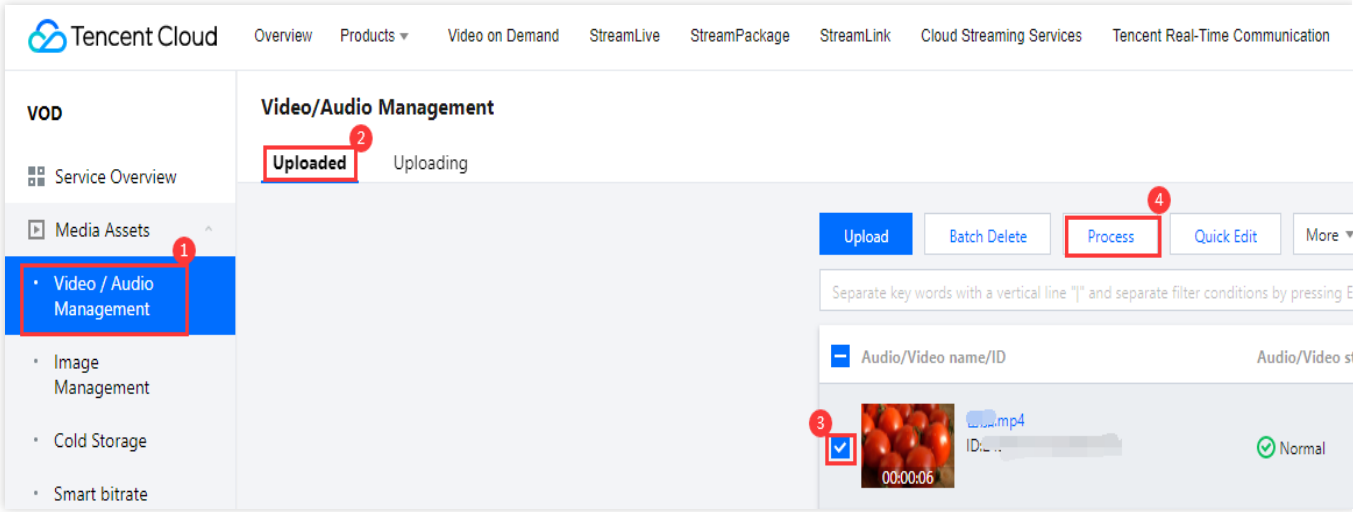
Last updated：2025-03-18 14:34:19

Overview

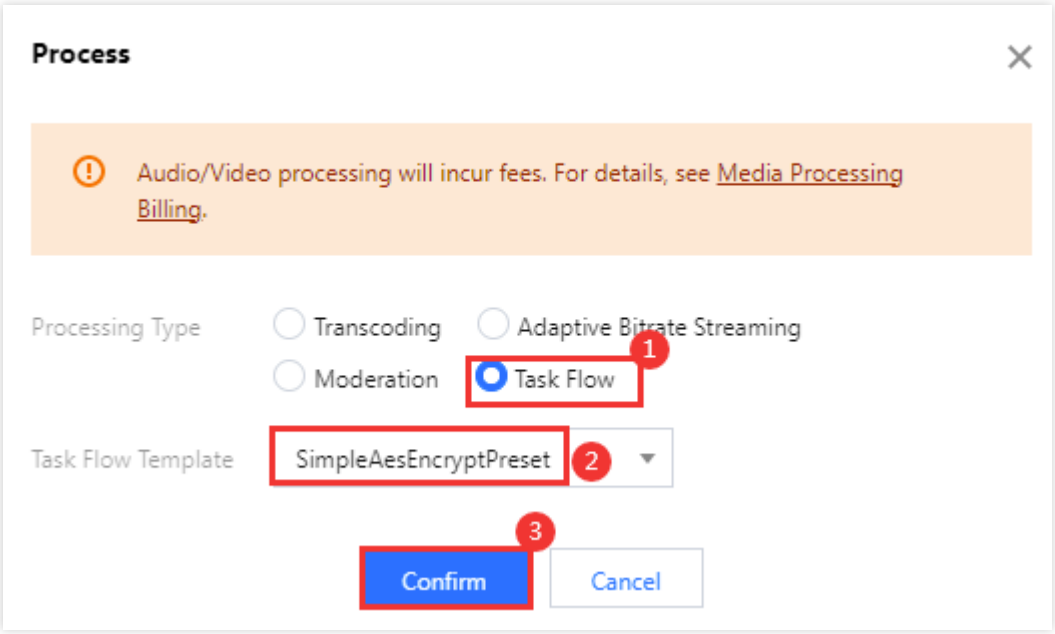
This document describes how to encrypt a video and use VOD's player to play the encrypted video.
Before you go on, make sure you have read Stage 1. Play back a source video. You will be using the account registered and the video uploaded in stage 1.

Step 1. Encrypt a Video

1. In the VOD console, select Application Management on the left sidebar and select the target application. On the **Media Assets > Video/Audio Management** page, select the target video, and click **Task Flow**.



2. Complete the following settings:
Select **Task Flow** as the **Processing Type**.
Select the **SimpleAesEncryptPreset** task flow template.



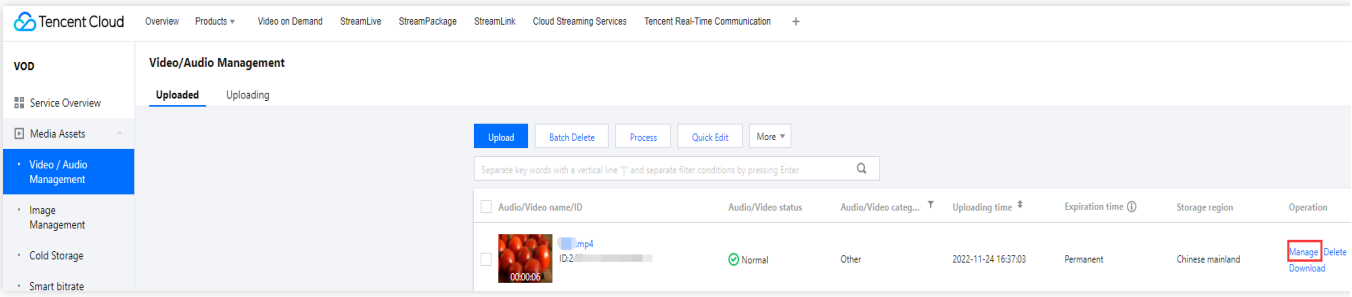
Note:

`SimpleAesEncryptPreset` is a preset task flow, which uses the adaptive bitrate streaming template 12, thumbnail generation template 10, and image sprite generation template 10.

The adaptive bitrate streaming template 12 outputs encrypted multi-bitrate streams.

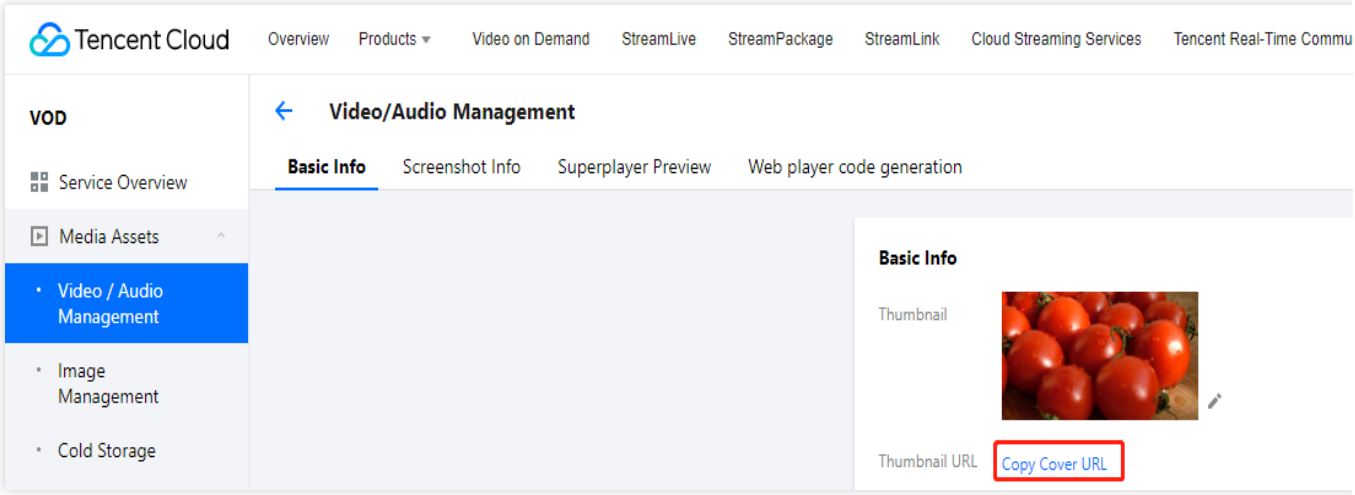
3. Click **Confirm**. Go to **Task Center**. If the status of the task changes from "Processing" to "Completed", the processing of the video is finished.

4. Go to **Media Assets > Video/Audio Management**, find your video, and click **Manage** on the right.



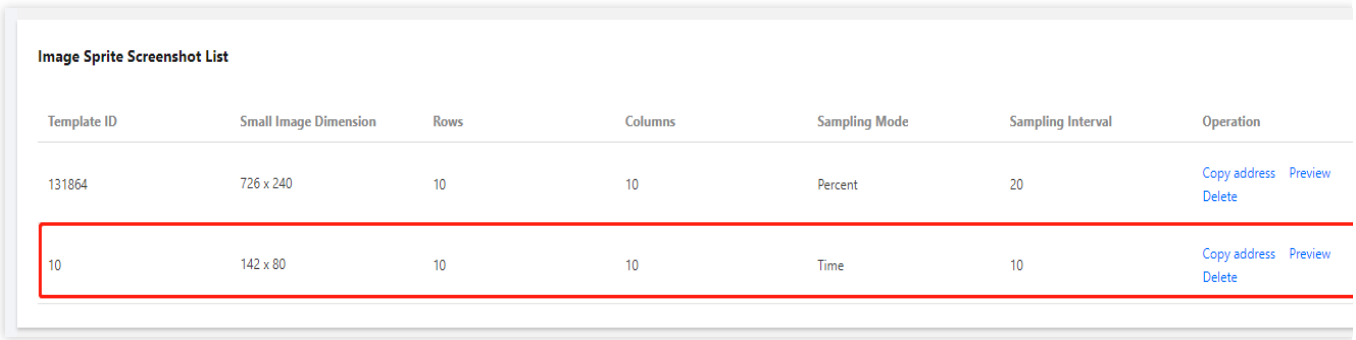
4.1 In the **Basic Info** area:

You can view the thumbnail generated and the outputs of adaptive bitrate streaming (template ID: 12).



4.2 Select the **Screenshots** tab:

You can view the image sprite generated (template ID: 10).



Step 2. Generate a Player Signature

In this step, you can use the signature tool to quickly generate a signature for the player to play back the video.

1. In the VOD console, select Application Management on the left sidebar and select the target application. Go to **Distribution and Playback > Player Signature Tools** and complete the following settings:

Video file ID: Enter the file ID generated in Step 1.

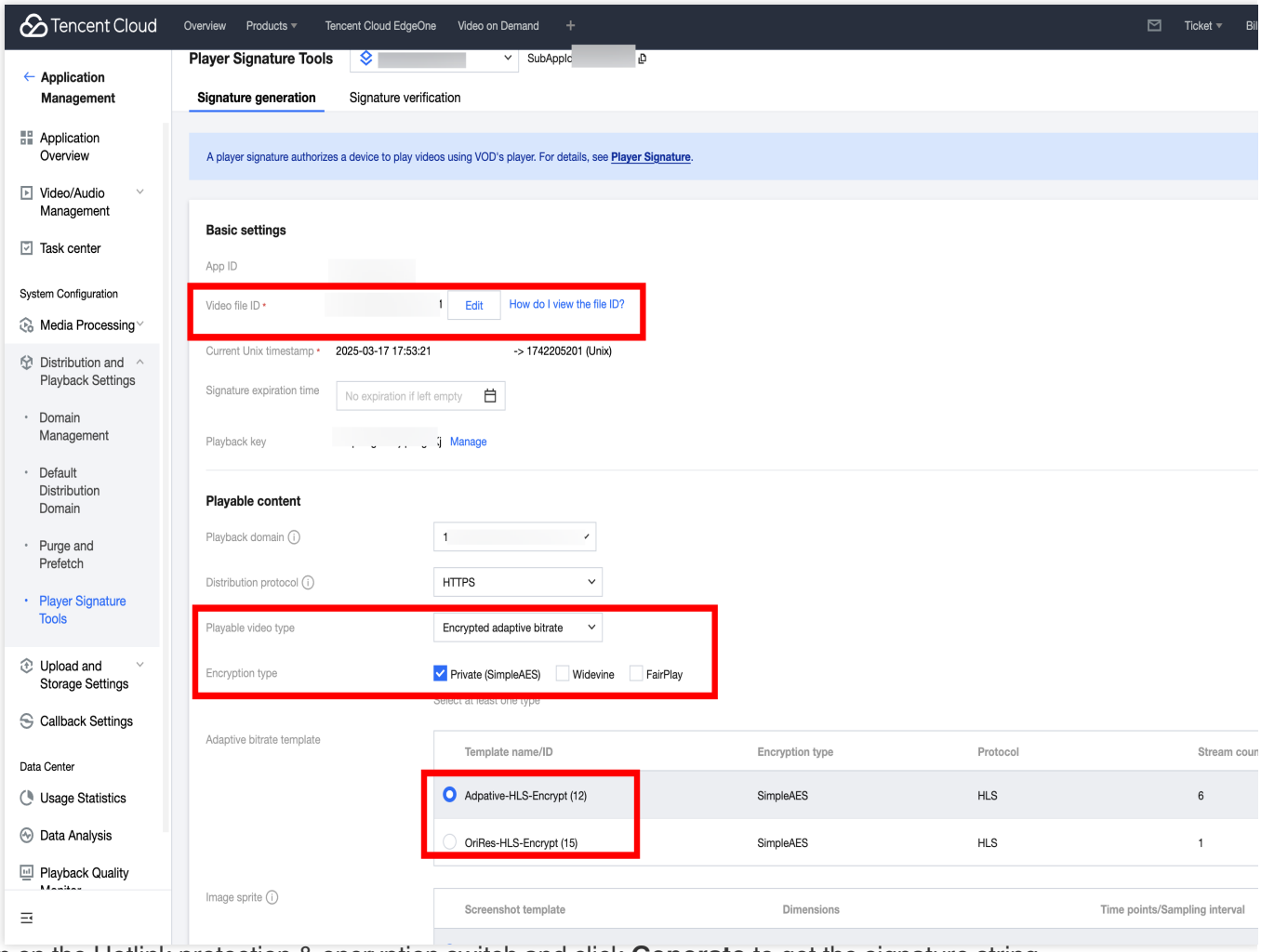
Signature expiration time: Enter the player signature expiration time. If you leave it empty, the signature will never expire.

Playable video type: Select **Encrypted adaptive bitrate**.

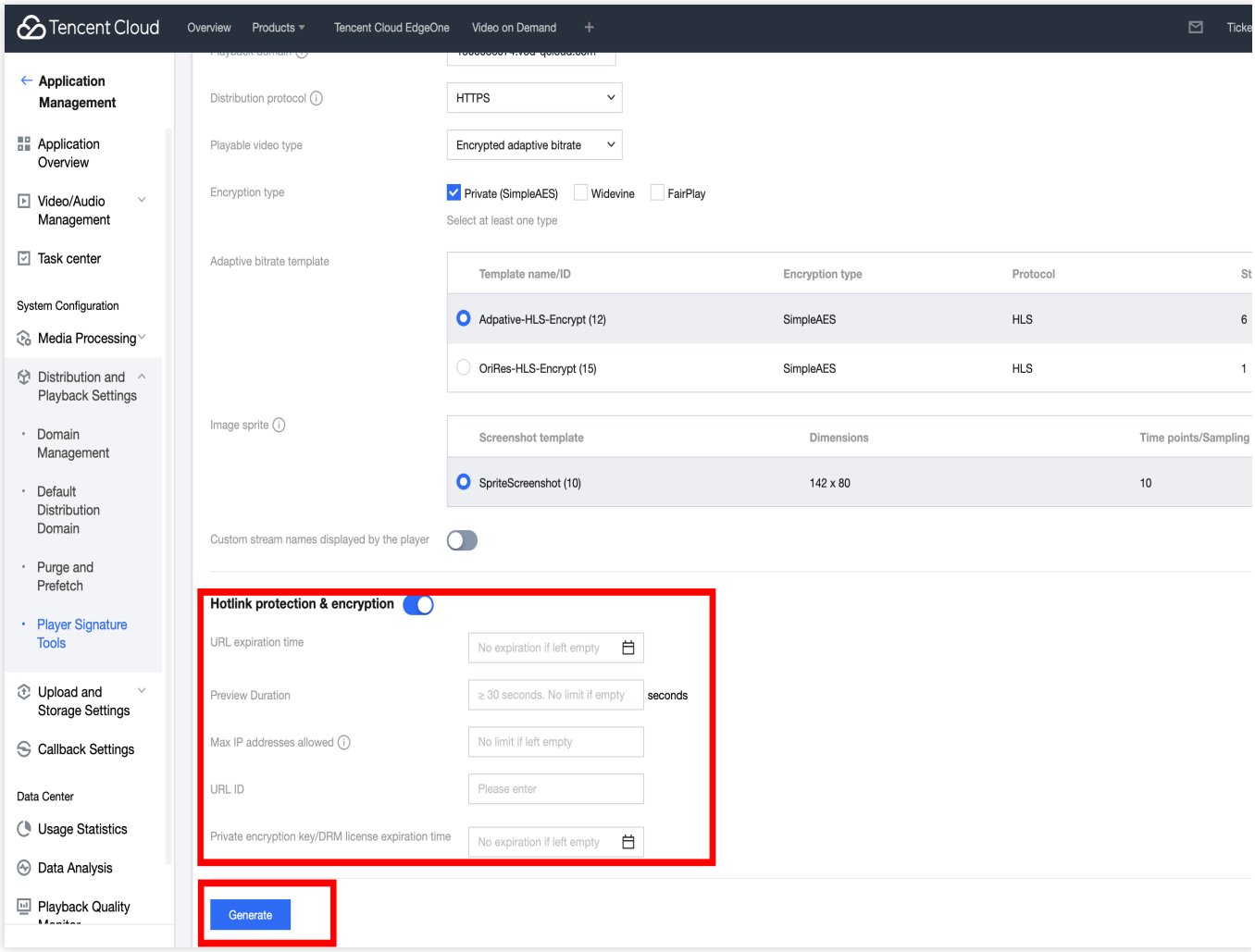
Encryption type: Select **Private (SimpleAES)**.

Adaptive bitrate template: Select `Adpative-HLS-Encrypt (12)` .

Image sprite: Select `SpriteScreenshot (10)` .



2. Turn on the Hotlink protection & encryption switch and click **Generate** to get the signature string.



Step 3. Play the Video

After step 2, you have obtained the three parameters needed for video playback: `appId`, `fileId` and `psign` (player signature). The following describes how to play back the video on the web.

Playback on the web

1. Open the web player demo.

Select **Video playback**.

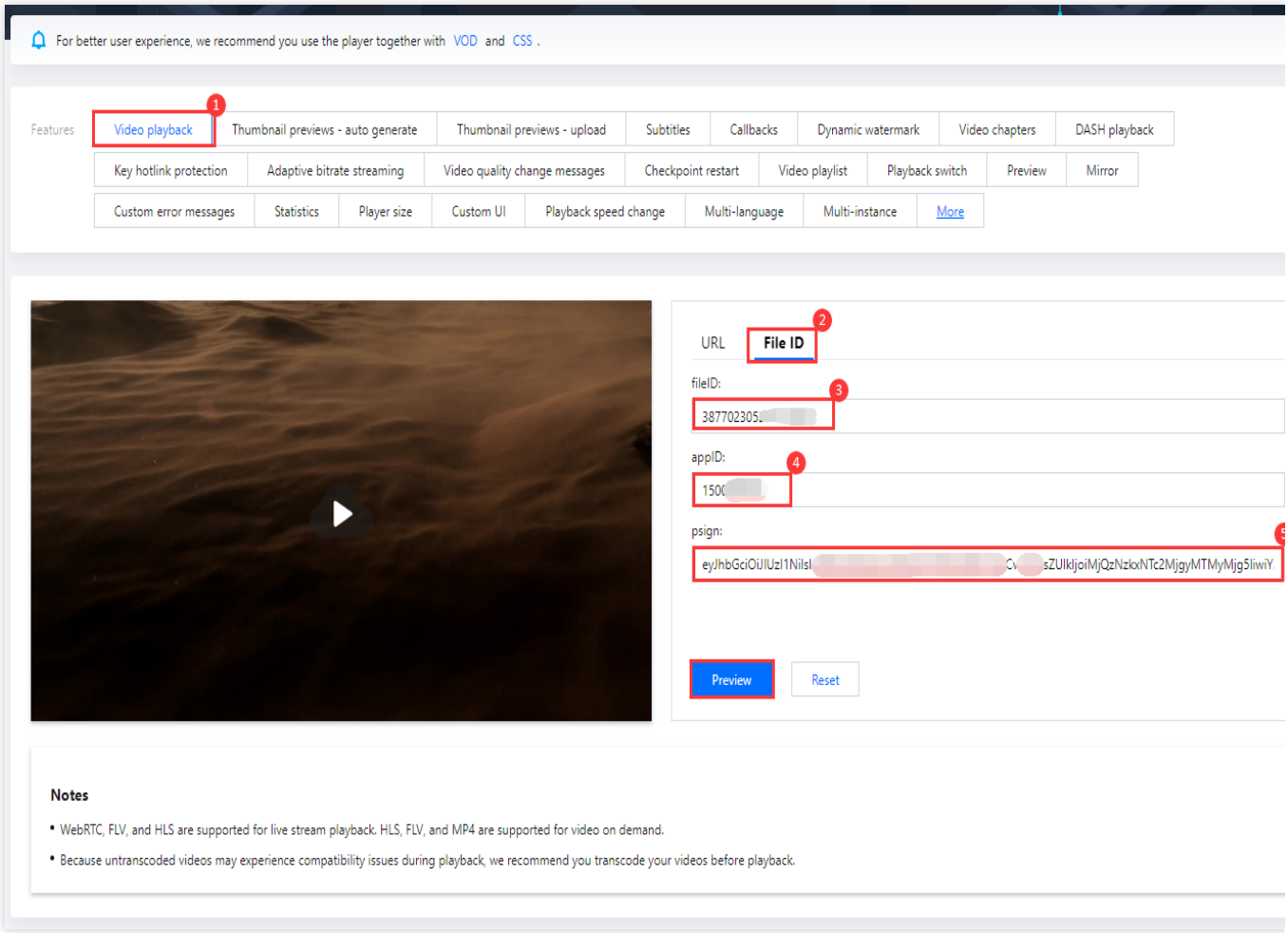
Select the **File ID** tab.

fileID: Enter the same file ID in the previous step.

appID: Enter the ID of the VOD application to which the file belongs (which is also the App ID displayed on the signature generation page in the previous step).

psign: Enter the signature string generated in the previous step.

2. Click **Preview** to play the video.



Multi-platform player demos

After generating the player signature, you can use our player demos for web, Android, and iOS to play the video. For details, see the source code for the demos.

Summary

At this point, you have understood how to encrypt a video and use the player to play it back.

Stage 5. Play back a long video

Last updated：2022-12-30 16:30:24

This document describes how to use the player to play back long videos common on audio/video platforms. It covers the web, iOS, and Android versions of the player and also details the features of key hotlink protection, adaptive bitrate streaming, video thumbnail preview, and video timestamping.

Overview

After reading this document, you will know:
How to configure key hotlink protection, which allows you to set the validity period, number of viewers, playback duration, etc.
How to output adaptive bitstreams in VOD (a player can dynamically select the most appropriate bitrate for playback based on the current bandwidth).
How to set video timestamps.
How to use an image sprite as a thumbnail in VOD.
How to use the player.
Before reading this document, make sure that you have read Stage 1. Play back a video with Player in the Player Guide and understand the concept of `fileid` in VOD.

Directions

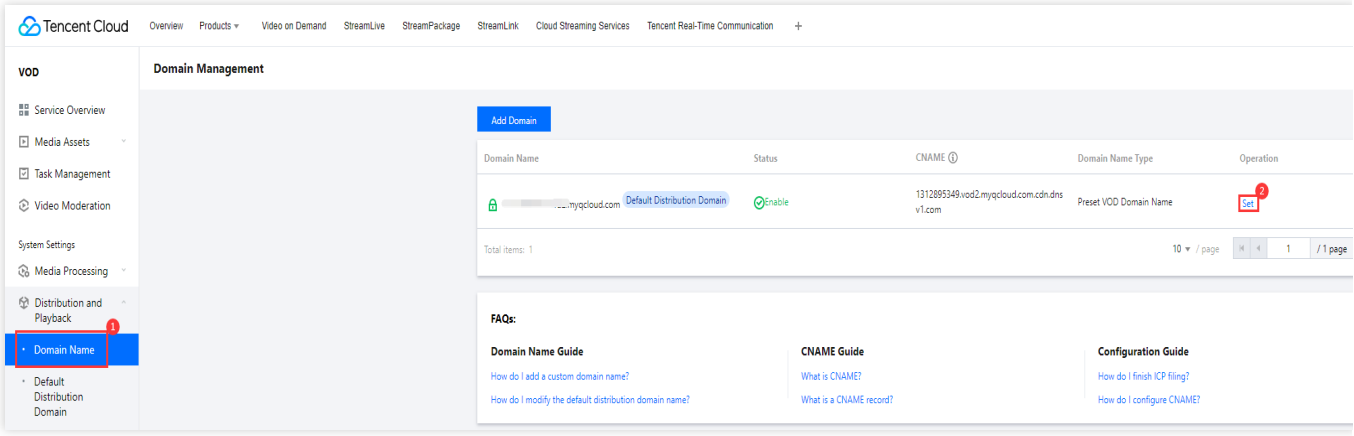
Step 1. Enable key hotlink protection

The following takes enabling key hotlink protection for the default distribution domain name under your account as an example:

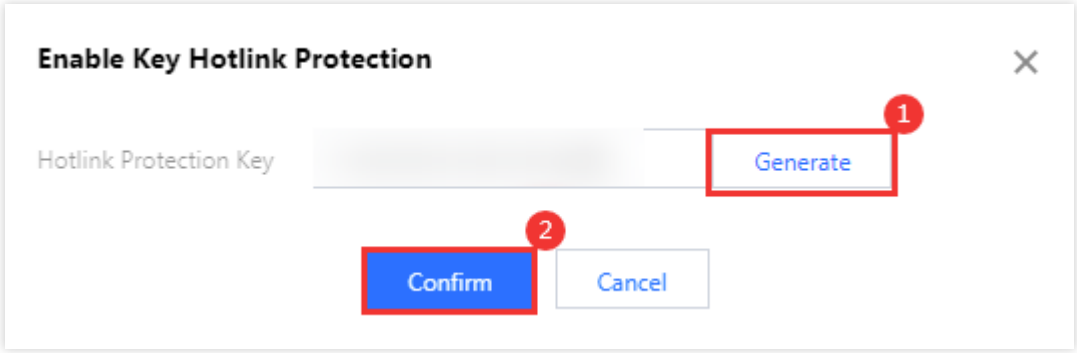
Note:

Do not directly enable hotlink protection for the domain name in your production environment; otherwise, playback of videos in the production environment may fail.

1. Log in to the VOD console, select **Distribution and Playback** > **Domain Name** to enter the settings page.



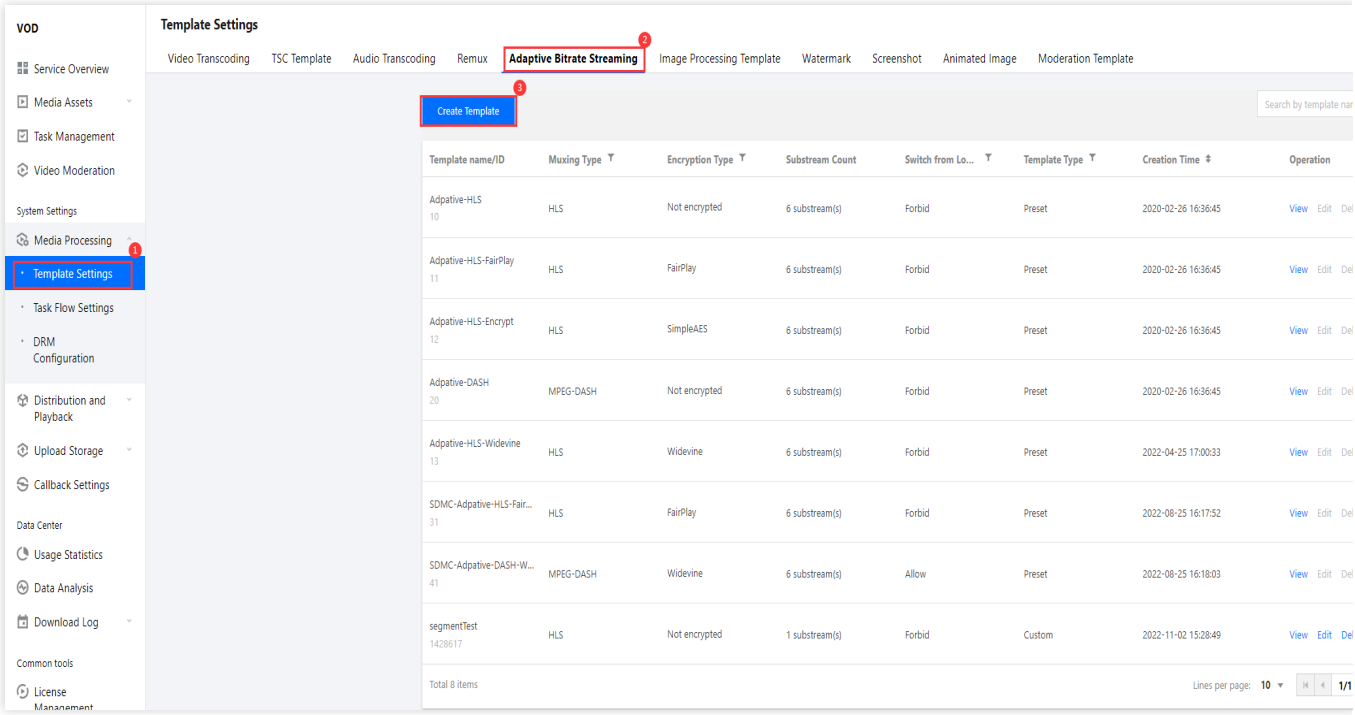
2. Click the **Access Control** tab, find **Key Hotlink Protection**, click the gray button to enable it, click **Generate** in the pop-up window to generate a random key, and click **OK** to save the configuration and make it take effect.



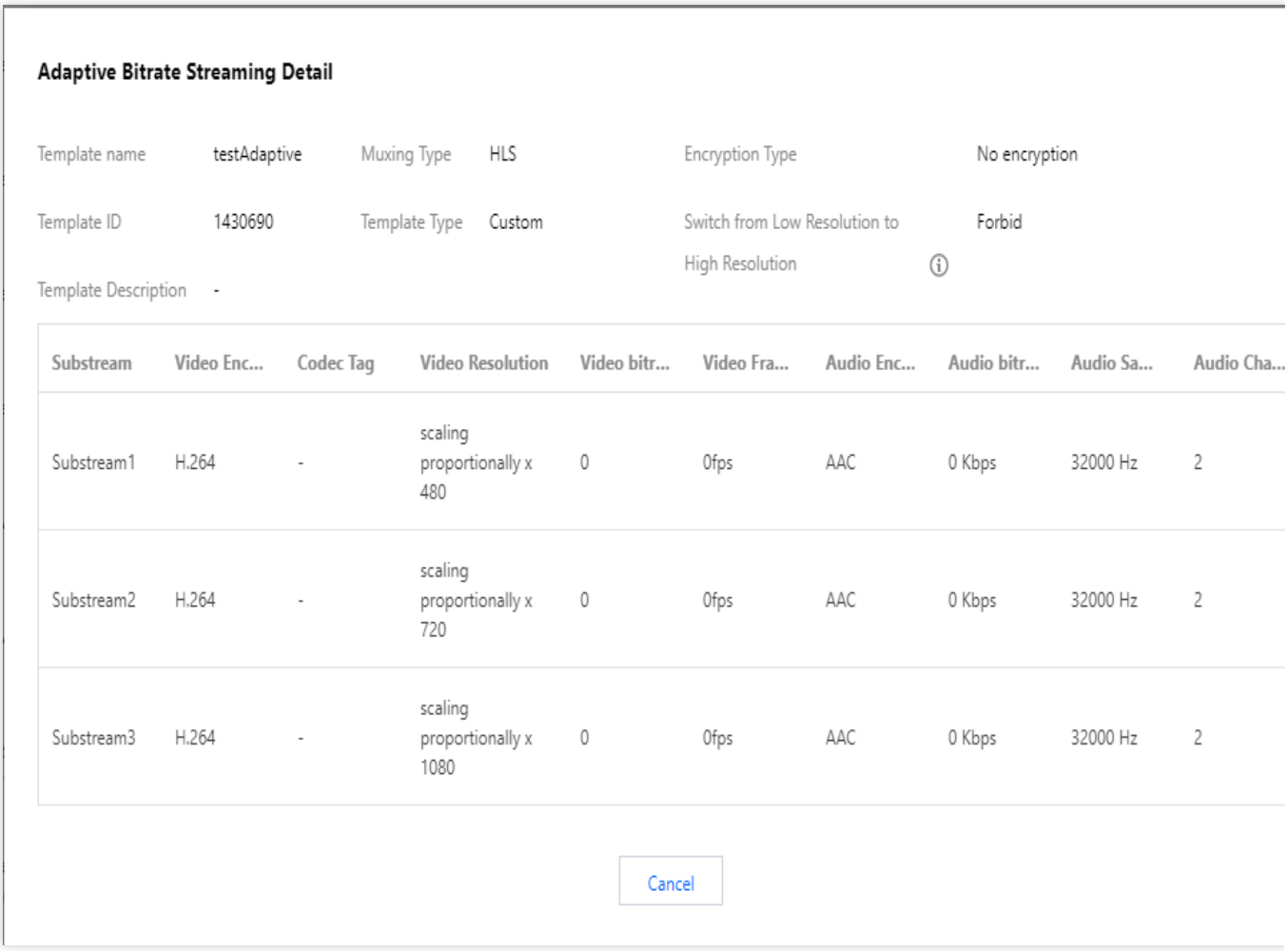
Step 2. Output adaptive bitstream and image sprite

This step describes how to transcode a video to adaptive bitstream and output image sprite.

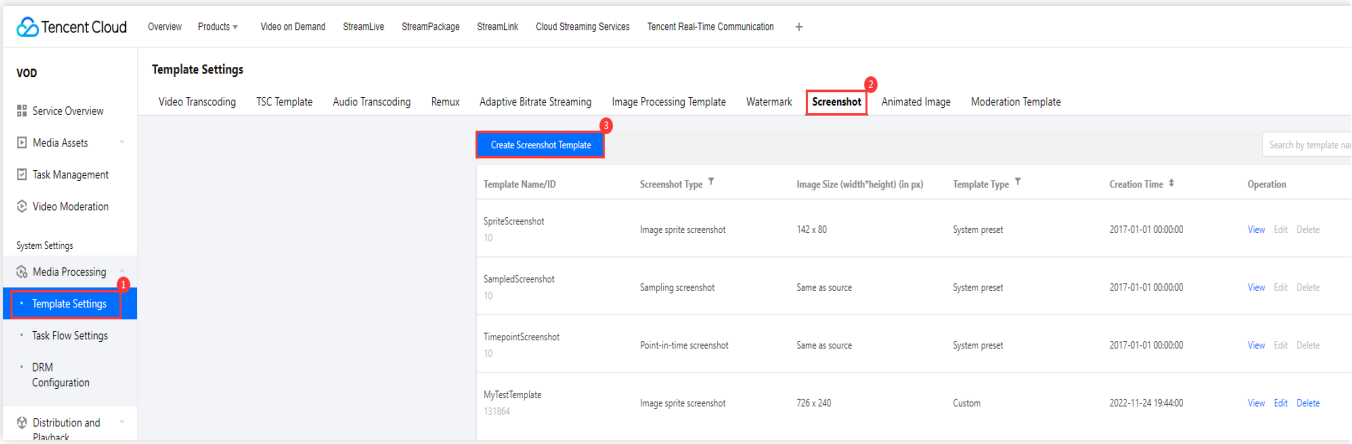
1. Log in to the VOD console, select **Video Processing Settings** > **Template Settings** > **Adaptive Bitrate Streaming Template**, and click **Create Template**.



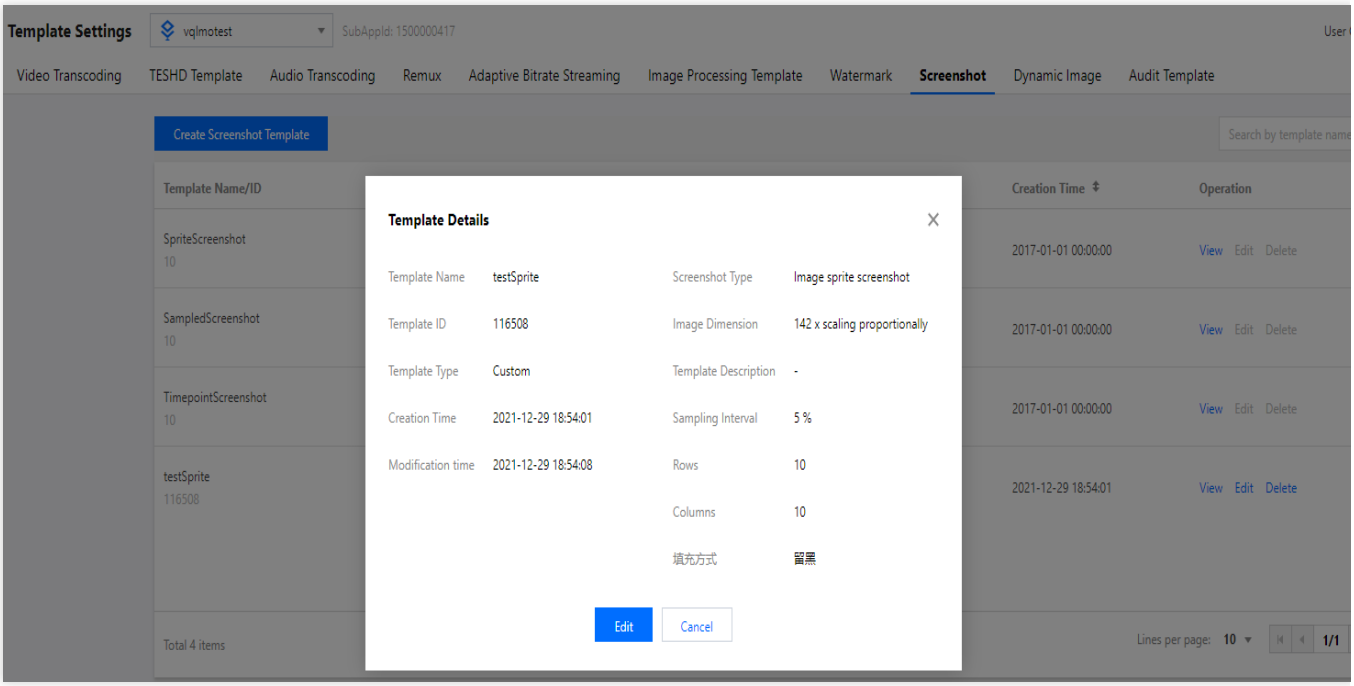
Create the adaptive bitstream through the template as needed. The following example shows an adaptive bitrate streaming template named `testAdaptive`, which contains three substreams with a resolution of 480p, 720p, and 1080p. The video bitrate, video frame rate, and audio bitrate are the same as the original video.



2. Select **Video Processing Settings** > **Template Settings** > **Screenshot Template** and click **Create Template**.

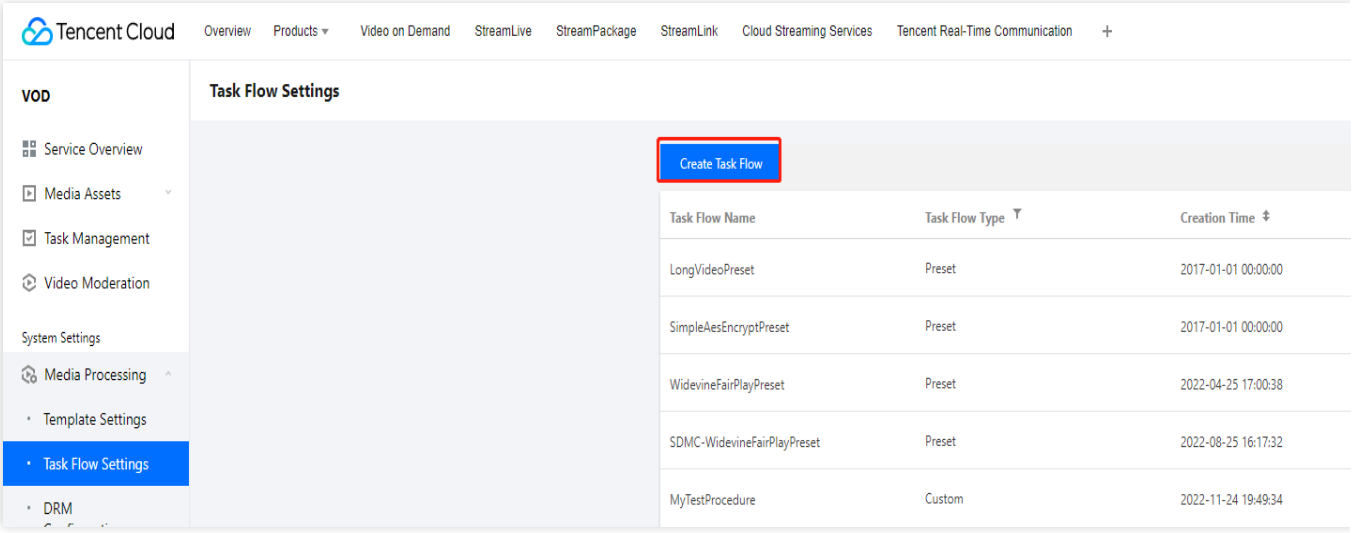


Create the image sprite through the template as needed. The following example shows an image sprite template named `testSprite`, with a sampling interval of 5%, 10 rows, and 10 columns.

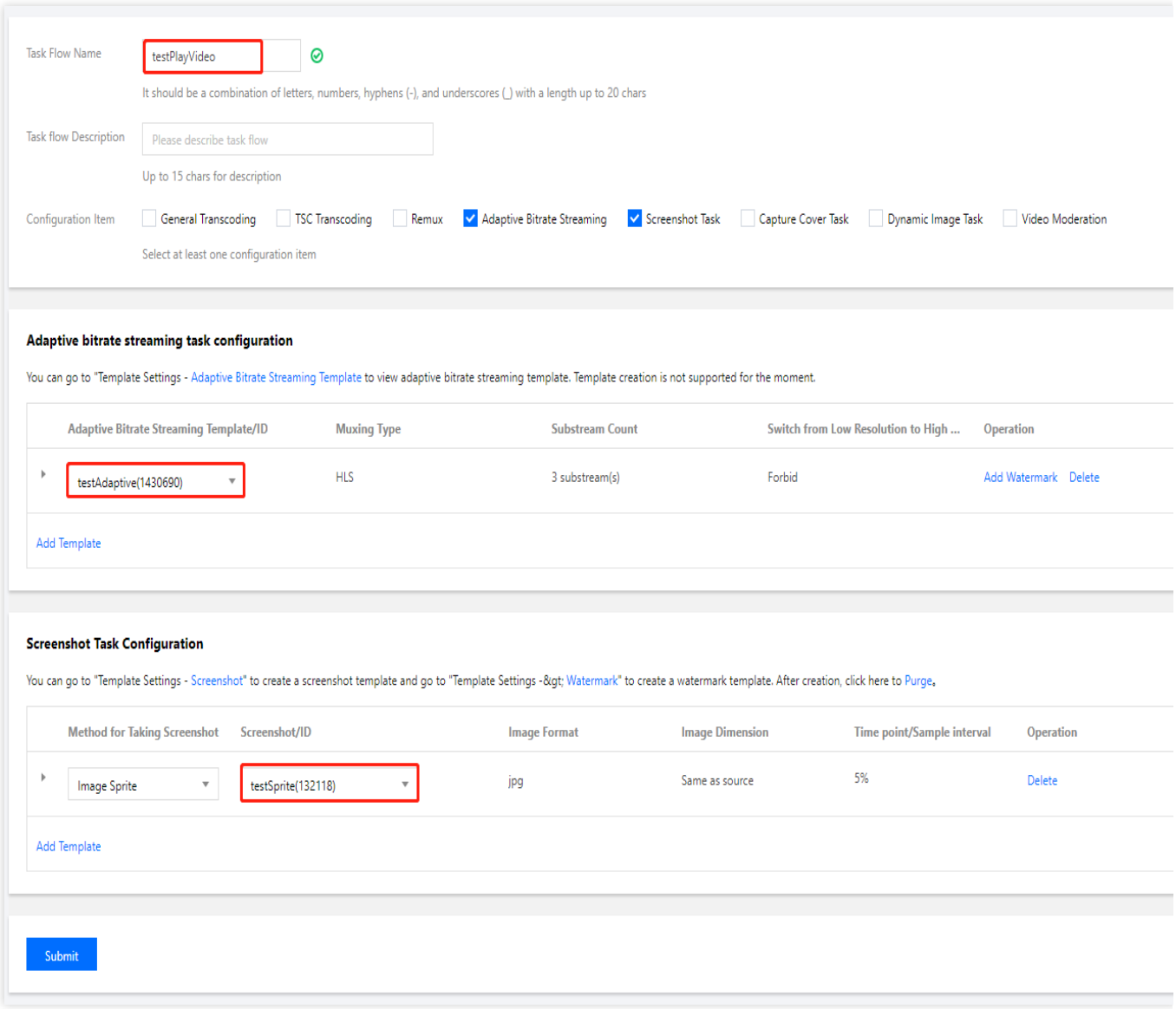


3. Add the adaptive bitrate streaming and image sprite templates through the task flow.

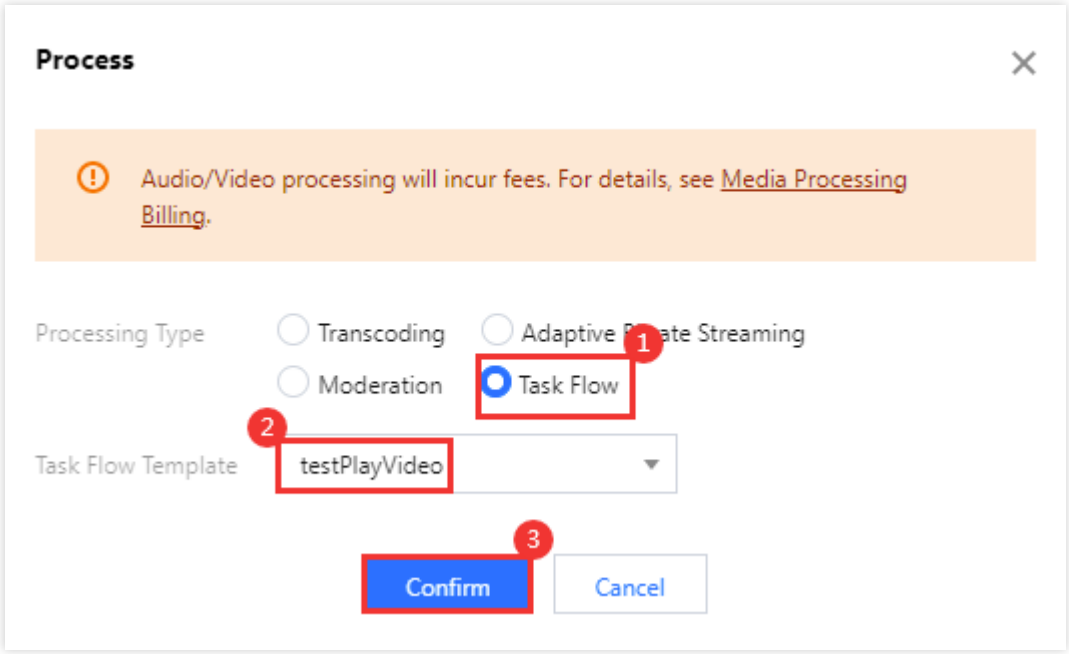
Select **Video Processing** > **Task Flow Settings** and click **Create Task Flow**.



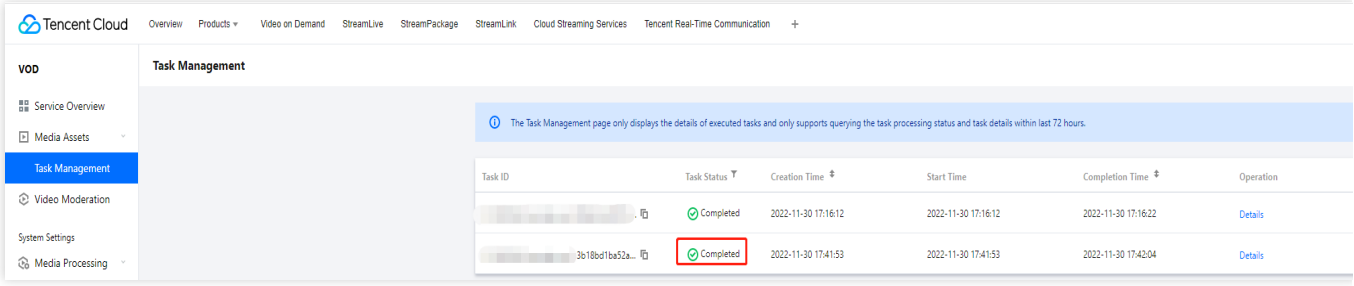
Add a task through the task flow as needed. The following example shows a task flow named `testPlayVideo`, which only adds the adaptive bitrate streaming and image sprite templates from the previous examples.



4. Select **Media Assets** > **Video/Audio Management**, select the target video (FileId: 387xxxxx8142975036), click **Task Flow**, and select a task flow template to start the task.



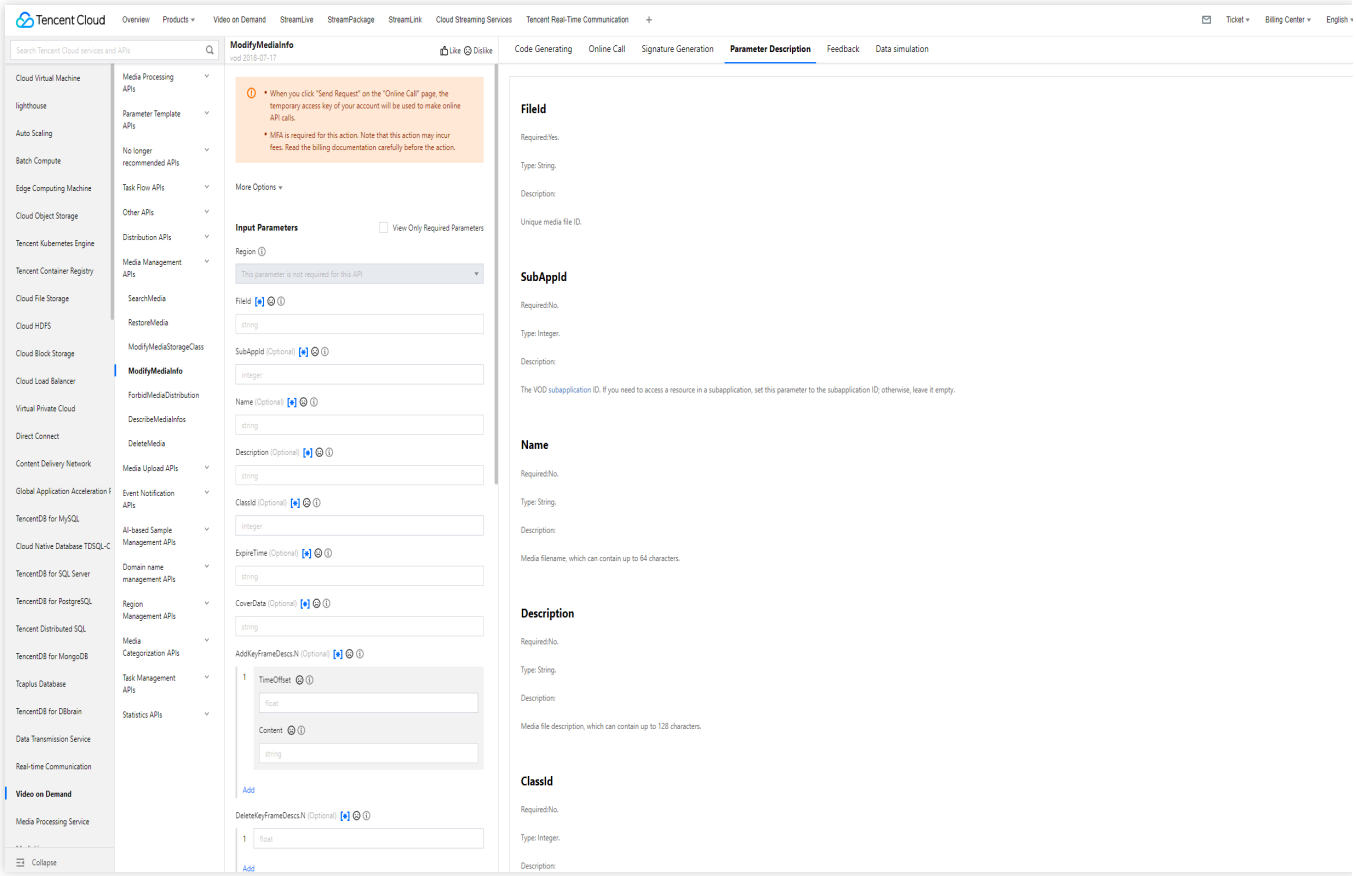
5. At this point, you can view the task execution status in the **Task Managment** page and get the task result after completion.



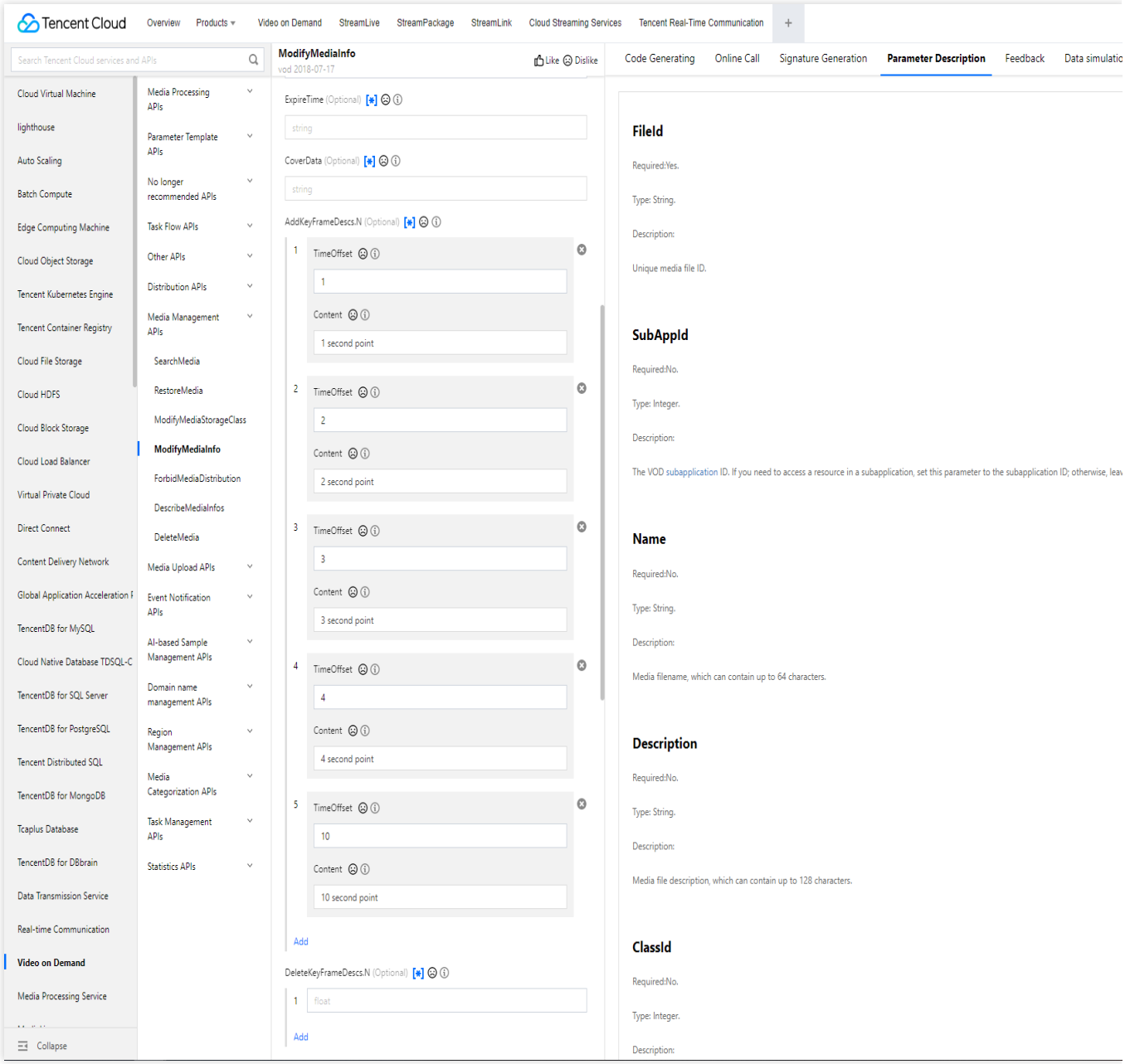
Step 3. Add video timestamps

This step describes how to add a set of video timestamps.

1. Go to VOD Server APIs > **Media Asset Management APIs** > **ModifyMediaInfo** and click **Debug** to enter the TencentCloud API console for debugging.



2. Add the specified video timestamps through the **AddKeyFrameDescs.N** parameter.



At this point, you have completed the operation in the cloud, output the adaptive bitstreams and image sprite, and added the video timestamps.

Step 4. Generate a player signature

In this step, you can use the signature tool to quickly generate a signature for the player to play back the video.

Select **Distribution and Playback** > **Player Signature Tools** and enter the following information:

Video file ID: Enter the `FileId` (387xxxxx8142975036) used in **step 2**.

Signature expiration time: Enter the player signature expiration time. If you leave it empty, the signature will never expire.

Playable video type: Select **Unencrypted adaptive bitrate**.

Playable adaptive bitrate streaming template: Select `testAdaptive (1429229)` .

Image sprite template for thumbnail preview: Select `testSprite (131353)` .
Hotlink protection and encryption: Toggle it on and configure as follows:
Link expiration time: Set it to the expiration time of the obtained hotlink protection signature.
Maximum playback IPs: Set the maximum number of IPs allowed for playback.
Click **Generate** to get the signature string.

Step 5. Integrate the player

After step 4, you have obtained the three parameters needed for video playback: `appId` , `fileId` and `psign` (player signature).
This step describes how to play back the adaptive bitstreams, thumbnails, and timestamps in the player for web, iOS, and Android.
Web
iOS
Android
You need to integrate the RT-Cube Player as instructed in Web integration. After importing the player's SDK file, you can play back the video by using the `appId` , `fileId` , and `psign` .
The construction method of the player is `TCPlayer` , which can be used to create a player instance for playback.

1. Place the player container in the HTML file

Place the player container in the desired place on the page. For example, add the following code to `index.html` (the container ID, width, and height can be customized).

```
<video id="player-container-id" width="414" height="270" preload="auto"
playsinline webkit-playsinline>
</video>
```

2. Play back with the fileID

Add the following initialization script to the `index.html` page initialization code to pass in the obtained `fileID` and `appID` for playback.

```
var player = TCPlayer('player-container-id', { // player-container-id is the
player container ID, which must be the same as that in HTML
    fileID: '387xxxxx8142975036', // `fileID` of the video to be played back
    appID: '1400329073', // `appID` of the VOD account to play back the video
    psign:'psignxxxx'   // `psign` is a player signature. For more information
on the signature and how to generate it, see [Player Signature]
(https://www.tencentcloud.com/document/product/266/38099).
});
```

You need to integrate the RT-Cube Player as instructed in iOS Integration. Then, you can play back the video by using the `appId`, `fileId`, and `psign`.
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 a delegate for events
_playerView.delegate = self;
// Set the parent view. _playerView will be automatically added under holderView.
_playerView.fatherView = self.holderView;
```

Playing back with the `fileId`

```
SuperPlayerModel *model = [[SuperPlayerModel alloc] init];
model.appId = 1400329073;// Configure AppId
model.videoId = [[SuperPlayerVideoId alloc] init];
model.videoId.fileId = @"387xxxxx8142975036"; // Configure `FileId`
// `pSign` is a player signature. For more information on the signature and how to
model.videoId.pSign = @"psignxxxx";
[_playerView playWithModelNeedLicence:model];
[_playerView playWithModel:model];
```

You need to integrate the RT-Cube Player as instructed in Android integration. Then, you can play back the video by using the `appId`, `fileId`, and `psign`.
The main class of the player is `SuperPlayerView`, and videos can be played back after it is created.

1. Create the SuperPlayerView in the layout file

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

2. Play back with the fileId

```
// Import `SuperPlayerView` into the layout file and create an instance
mSuperPlayerView = (SuperPlayerView) findViewById(R.id.superVodPlayerView);

SuperPlayerModel model = new SuperPlayerModel();
model.appId = 1400329073;// Configure AppId
```

```
model.videoId = new SuperPlayerVideoId();
model.videoId.fileId = "387xxxxx8142975036"; // Configure `FileId`
// `pSign` is a player signature. For more information on the signature and how to
model.videoId.pSign = "psignxxxx";

mSuperPlayerView.playWithModel(model);
```

Summary

At this point, you can play back media files with hotlink protection enabled, view the image sprite and video timestamps, and automatically switch dynamic adaptive bitstreams in the player.
For more features, see Feature Description.

Integration (UI Included)
Web Integration
TCPlayer Integration Guide

Last updated：2024-10-17 21:44:58

This document introduces the Web Player SDK (TCPlayer) tailored for both VOD and live streaming. It can be quickly integrated with your own web application to enable video playback features. TCPlayer comes with a default set of UI elements, which you can use as needed.

Overview

The web player utilizes the `<video>` tag of HTML5 and Flash for video playback. It offers a uniform video playback experience across different platforms when browsers do not natively support video playback. In conjunction with Tencent's Video on Demand service, it provides hotlink protection and features for playing standard encrypted HLS videos.

Supported protocols

Audio/Video Protocol	Use	URL Format	PC Browser	Mobile Browser
MP3	Audio	http://xxx.vod.myqcloud.com/xxx.mp3	Supported	Supported
MP4	VOD playback	http://xxx.vod.myqcloud.com/xxx.mp4	Supported	Supported
HLS(M3U8)	Live stream	http://xxx.liveplay.myqcloud.com/xxx.m3u8	Supported	Supported
	VOD playback	http://xxx.vod.myqcloud.com/xxx.m3u8	Supported	Supported
FLV	Live stream	http://xxx.liveplay.myqcloud.com/xxx.flv	Supported	Partially supported
	VOD playback	http://xxx.vod.myqcloud.com/xxx.flv	Supported	Partially supported
WebRTC	Live stream	webrtc://xxx.liveplay.myqcloud.com/live/xxx	Supported	Supported

Note:

Only H.264 encoding is supported.

The player is compatible with mainstream browsers and can automatically select the optimal playback scheme depending on the browser.

In some browser environments, HLS and FLV video playback depends on Media Source Extensions.

If a browser does not support WebRTC, a WebRTC URL passed in will be converted automatically to better support playback.

Supported Features

Feature\\Browser	Chrome	Firefox	Edge	QQ Browser	Mac Safari	iOS Safari	WeChat	Android Chrome
Player dimension configuration	✓	✓	✓	✓	✓	✓	✓	✓
Resuming playback	✓	✓	✓	✓	✓	✓	✓	✓
Playback speed change	✓	✓	✓	✓	✓	✓	✓	✓
Preview thumbnails	✓	✓	✓	✓	-	-	-	-
Changing `fileID` for playback	✓	✓	✓	✓	✓	✓	✓	✓
Flipping videos	✓	✓	✓	✓	✓	✓	✓	✓
Progress bar marking	✓	✓	✓	✓	✓	-	-	-
HLS adaptive bitrate	✓	✓	✓	✓	✓	✓	✓	✓
Referer hotlink protection	✓	✓	✓	✓	✓	✓	✓	-
Definition change notifications	✓	✓	✓	✓	-	-	-	✓
Preview	✓	✓	✓	✓	✓	✓	✓	✓

Playing HLS videos encrypted using standard schemes	✓	✓	✓	✓	✓	✓	✓	✓
Playing HLS videos encrypted using private protocols	✓	✓	✓	-	-	-	Android:✓ iOS: -	✓
Video statistics	✓	✓	✓	✓	-	-	-	-
Video data monitoring	✓	✓	✓	✓	-	-	-	-
Custom UI messages	✓	✓	✓	✓	✓	✓	✓	✓
Custom UI	✓	✓	✓	✓	✓	✓	✓	✓
On-screen comments	✓	✓	✓	✓	✓	✓	✓	✓
Watermark	✓	✓	✓	✓	✓	✓	✓	✓
Ghost watermark	✓	✓	✓	✓	✓	✓	✓	✓
Playlist	✓	✓	✓	✓	✓	✓	✓	✓
Frame sync under poor network conditions	✓	✓	✓	✓	✓	✓	✓	✓

Note:

Only H.264 encoding is supported.

Chrome and Firefox for Windows and macOS are supported.

Chrome, Firefox, Edge, and QQ Browser need to load `hls.js` to play HLS.

The Referer hotlink protection feature is based on the Referer field of HTTP request headers. Some HTTP requests initiated by Android browsers do not carry the Referer field.

The player is compatible with mainstream browsers and can automatically select the optimal playback scheme depending on the browser used. For example, for modern browsers such as Chrome, the player uses the HTML5 technology for playback, and for mobile browsers, it uses the HTML5 technology or the browser's built-in capabilities.

Preparations

From version 5.0.0, the TCPlayer SDK for Web (TCPlayer) requires access to a License authorization for use. If you don't need the new premium functions, you can apply for a basic License to **try TCPlayer for free;** if you want to use the newly added premium functions, you need to purchase a premium License. The detailed information is as follows:

TCPlayer feature	Feature Scope	Required License	Pricing	Authorization Unit
Basic Functions	Includes all features provided in versions prior to 5.0.0, see Product Features for details	Web Player Basic Version License	**0 CNY** Free Application	An exact domain (1 License can authorize up to 10 exact domains)
Premium Functions	Basic Version Features, VR Playback, Security Check	Web Player Premium Version License	399 CNY/month Buy Now	Wildcard Domain (1 License can authorize up to 1 wildcard domain)

Note:
1. Web Player Basic Version License can be applied for free, with a default validity of 1 year; if the remaining validity period is less than 30 days, it can be renewed for free.
2. To facilitate local development, the player won't authenticate localhost or 127.0.0.1; hence, these types of local service domain names need not be applied for when requesting a License.

Integration Guide

By following these steps, you can add a video player to your website.

Step 1. Import files into the page

The Player SDK supports two integration methods: CDN and NPM:

1. Integration through CDN

Create a new index.html file in your local project and import the player style and script files into the HTML page.
It is recommended to deploy resources on your own when using the Player SDK, click Download player resources.
Deploy the unzipped folder without altering its directory structure to prevent cross-referencing issues between resources.
If the deployment address is `aaa.xxx.ccc` , import the player style and script files at the appropriate places. When

deploying on your own, you need to manually reference the dependency files under the libs folder of the resource package, otherwise, the Tencent Cloud CDN files will be requested by default.

```
<link href="aaa.xxx.ccc/tcplayer.min.css" rel="stylesheet"/>
<!--To play HLS format videos in modern browsers like Chrome and Firefox through H
<script src="aaa.xxx.ccc/libs/hls.min.x.xx.m.js"></script>
<!--Player script file-->
<script src="aaa.xxx.ccc/tcplayer.vx.x.x.min.js"></script>
```

2. Integration through npm

First, install the tcplayer npm package:

```
npm install tcplayer.js
```

Import the SDK and style files:

```
import TCPlayer from 'tcplayer.js';
import 'tcplayer.js/dist/tcplayer.min.css';
```

Step 2. Place the player container

Add the player container to the location on the page where the player is to be displayed. For example, add the following code in index.html (the container ID and dimensions can be custom defined).

```
<video id="player-container-id" width="414" height="270" preload="auto" playsinline
</video>
```

Note:

The player container must be a `<video>` tag.

In the example, `player-container-id` is the ID of the player container, which you can set yourself.

We recommend you set the size of the player container zone through CSS, which is more flexible than the attribute and can achieve effects such as fit to full screen and container adaption.

The `preload` attribute in the example specifies whether to load the video after the page is loaded. It is usually set to `auto` for faster video playback. Other options are: `meta` (to only load metadata after the page loads) and `none` (to not load the video after the page loads). Videos will not automatically load on mobile devices due to system restrictions.

The attributes `playsinline` and `webkit-playsinline` are used to achieve inline playback in standard mobile browsers without hijacking video playback. This is just an example, please use as needed.

Setting the `x5-playsinline` attribute in the TBS kernel will utilize the X5 UI player.

Step 3. Initialize the player

After page initialization, you can play video resources. The player supports both video on demand (VOD) and live streaming playback scenarios as follows:
VOD playback: The player can play Tencent Video on Demand media resources through FileID. For the specific VOD process, please refer to the Using the Player for Playback document.
Live playback: The player can pull a live audio/video stream for playback by passing in a URL. For information on generating a Tencent Cloud Streaming Services URL, see Splicing Live Streaming URLs.
URL playback (VOD and live)
File ID playback (VOD)
After page initialization, call the method in the player instance to pass in the URL to the method.

```
// `player-container-id` is the player container ID, which must be the same as in H
var player = TCPlayer('player-container-id', {
    sources: [{
      src: 'Please replace with your playback URL',
    }],
    licenseUrl: 'Please replace with your licenseUrl', // License URL, see the prep
    language: 'Please replace with your Setting language', // Setting language en |
});

// player.src(url); // URL playback address
```

In the initialization code on the index.html page, add the following initialization script. Pass in the video ID and appID obtained during the preparation phase from the fileID (in Media Management) and (peek in **Account Information** > Basic Information).

```
var player = TCPlayer('player-container-id', { // player-container-id is the player
    fileID: 'Please enter your fileID', // Enter the fileID of the video to be play
    appID: 'Please enter your appID', // Enter the appID of your VOD account
    // Enter the player Signature psign, for information on the Signature and how t
    psign: 'Please enter your player Signature psign',
    licenseUrl: 'Please enter your licenseUrl', // Refer to the preparation section
    language: 'Please replace with your Setting language', // Setting language en |
});
```

Please Note:
Not all videos can be played successfully in a browser. We recommend you use Tencent Cloud's services to transcode a video before playing it.

Step 4. Implement more features

You can utilize the server-side capabilities of Video on Demand (VOD) for advanced features, such as automatic switching of adaptive streams, video thumbnail previews, and adding video marker information. These features are detailed in Play back a long video, which you can refer to for implementation.

Additionally, the player offers more features. For a list of features and instructions on how to use them, please see the Feature Demonstration page.

TCPlayer Resolution Configuration Guide

Last updated：2024-05-13 17:49:25

During playback, the video resolution can be switched automatically or manually to accommodate different sizes of playback devices and network conditions, thus enhancing viewing experience. This document will explain several scenarios.

Live Streaming

Live streaming videos are played in the form of URLs. When initializing the player, you can specify the URL to be played through the sources field. Alternatively, after initializing the player, you can play by calling the src method on the player instance.

1. Adaptive Bitrate (ABR)

ABR URLs can seamlessly transition during switches without causing interruptions or jumps, ensuring a smooth transition in both visual and auditory experiences. This technology is also relatively simple to use; it passes the playback URL to the player, which will automatically parse the sub-streams and render the resolution switching component on the control bar.

Example 1: Playing HLS ABR URLs

During the player's initialization, when receiving an ABR URL, the player will automatically generate a resolution switching component, so it can switch automatically based on network conditions.

```
const player = TCPlayer('player-container-id', { // player-container-id is the play
  sources: [{
    src: 'https://hls-abr-url', // hls ABR URL
  }],
});
```

Note:
Parsing the substreams of an HLS ABR requires the dependency on the MSE API of the playback environment. In browsers not supporting MSE (e.g., Safari on iOS), the browser internally handles this by automatically switching resolutions based on network conditions, but it won't be able to parse multiple resolutions for manual switching.

Example 2: Playing WebRTC ABR URLs

In the WebRTC ABR scenario, when receiving an URL, the player will automatically decompose the substream URLs based on the ABR template in the URL.

```
const player = TCPlayer('player-container-id',{
```

```
  sources: [{
    src: 'webrtc://global-lebtest-play.myqcloud.com/live/lebtest?txSecret=f22a813b2
  }],

  webrtcConfig: {
    // Whether to render multiple resolutions; enabled by default; optional
    enableAbr: true,
    // The label name corresponding to the template name; optional
    abrLabels: {
      d1080p: 'FHD',
      d540p: 'HD',
      d360p: 'SD',
      auto: 'AUTO',
    },
  },
});
```

The following explanations are provided for the parameters in the WebRTC URL:

1. tabr_bitrates specifies the ABR template, and the number of templates will determine the number of rendered resolutions. If no separate resolution label is set, the template name (e.g., d1,080p) will be used as the name of the resolution.

2. tabr_start_bitrate specifies the initial resolution setting.

3. tabr_control sets whether automatic resolution switching is enabled. When the automatic switching is enabled, the player will provide an option for automatic resolution adjustment.

2. Manually Setting Resolution

If the playback URL is not an ABR URL, you can also manually set the resolution. See the following code:

```
const player = TCPlayer('player-container-id', { // player-container-id is the play
  multiResolution:{
    //  Configure multiple resolution URLs
    sources:{
      'SD':[{
        src: 'http://video-sd-url',
      }],
      'HD':[{
        src: 'http://video-hd-url',
      }],
      'FHD':[{
        src: 'http://video-fhd-url',
      }]
    },
    //  Configure the tag for each resolution
    labels:{
      'SD':'Standard Definition','HD':'High Definition','FHD':'Full High Definition
```

```
    },
    //  Configure the order of resolutions in the player component
    showOrder:['SD','HD','FHD'],
    //  Configure the default resolution
    defaultRes: 'SD',
  },
});
```

VOD

For VOD playback using a fileID, the player signature specifies the type of file to be played (source, transcoded, or ABR) and the substream resolutions of ABR files. For a complete understanding of the VOD playback process, you can refer to the guide Play back an ABR streaming video.

When calculating the player signature, you can set the display names of substreams of different resolutions through the resolutionNames in the contentInfo field. If you leave it blank or fill in an empty array, the default configuration will be used.

```
resolutionNames: [{
  MinEdgeLength: 240,
  Name: '240P',
}, {
  MinEdgeLength: 480,
  Name: '480P',
}, {
  MinEdgeLength: 720,
  Name: '720P',
}, {
  MinEdgeLength: 1080,
  Name: '1080P',
}, {
  MinEdgeLength: 1440,
  Name: '2K',
}, {
  MinEdgeLength: 2160,
  Name: '4K',
}, {
  MinEdgeLength: 4320,
  Name: '8K',
}]
```

The number of substreams during playback depends on the number of substreams converted according to different ABR templates during transcoding. These substreams will fall within the MinEdgeLength range set by

resolutionNames based on short-side length and then be displayed with the corresponding Name as the resolution name.

To get a quick start on generating player signatures, you can use the Tencent Cloud VOD console's Player Signature Generation Tools.

TCPlayer Swift Live Streaming Downgrade Notice

Last updated：2024-05-13 17:49:25

Downgrade Scenarios

Live Event Broadcasting (LEB) relies on WebRTC technology, requiring both the operating system and browser to support WebRTC.
Currently, the SDK has been tested on the following operating systems and browsers, with the test results as follows.

Operating System	OS Version	Browser	Browser Version	Support for Stream Pull
Windows	Windows 10	Chrome	86+	✓
		Firefox	88+	✓
		Microsoft Edge	86+	✓
macOS	10.5+	Safari	13.1+	✓
		Chrome	86+	✓
		Firefox	88+	✓
		Microsoft Edge	86+	✓
iOS	13.1.1+	Safari	13.7+	✓
		Chrome	86+	✓
		Firefox	33+	✓
		Microsoft Edge	89	✓
		WeChat embedded	-	✓
Android	-	Chrome	86+	✓
		Firefox	88+	✓
		WeChat embedded	X5 kernel	✓

		WeChat embedded	XWeb kernel	✓

Additionally, in some browsers that support WebRTC, there may be decoding failures or server-side issues. In these cases, the player will convert the WebRTC URL to a more compatible HLS URL for playback. This behavior is known as downgrade processing.

To summarize, there are several scenarios that trigger downgrading:

The browser environment does not support WebRTC.

Failed to connect to the server, and the number of retries has exceeded the preset value (internal status code -2004).

Failed to decode during playback (internal status code -2005).

Other WebRTC-related errors (internal status code -2001).

Downgrade Methods

1. Automatic Downgrade

During player initialization, the LEB URL is received through the sources field. In environments requiring downgrade processing, the player automatically converts the LEB URL to the HLS URL.

For example, the LEB URL

```
webrtc://global-lebtest-play.myqcloud.com/live/lebtest?txSecret=f22a813b284137ed10d
```

will automatically convert to:

```
https://global-lebtest-play.myqcloud.com/live/lebtest.m3u8?txSecret=f22a813b284137e
```

2. Specified Downgrade

In ABR playback scenarios, downgrade processing requires manually specifying an HLS URL. Direct format conversion is not supported. This method can also be used for scenarios where users require manual configuration of downgrade URLs, and it is compatible with protocols beyond HLS.

```
var player = TCPlayer('player-container-id',{
    sources: 'webrtc://global-lebtest-play.myqcloud.com/live/lebtest?txSecret=f22a8
    webrtcConfig: {
      fallbackUrl: 'https://global-lebtest-play.myqcloud.com/live/lebtest_HLSABR.m3
    },
});
```

Downgrade Callback

When a downgrade is triggered, the player will initiate a callback.

```
player.on('webrtcfallback', function(event) {
    console.log(event);
});
```

iOS Integration Guide

Last updated：2024-04-18 17:06:53

Overview

The iOS Player is an open-source component that allows you to integrate powerful playback capabilities similar to those in Tencent Video into your project with just a few lines of code changes. In addition to basic features such as landscape/portrait orientation, resolution selection, gestures, and small-window playback, it also supports buffering, software/hardware decoding, and changing playback speed. Compared with built-in players, the RT-Cube Player supports more formats, has better compatibility, and offers more capabilities. It also features instant streaming and low latency, and comes with advanced features such as thumbnail generation.
If the Player component cannot meet your requirements, and you have some knowledge of engineering, you can integrate the Player SDK to customize the UI and playback features.

Limits

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

This Document Describes

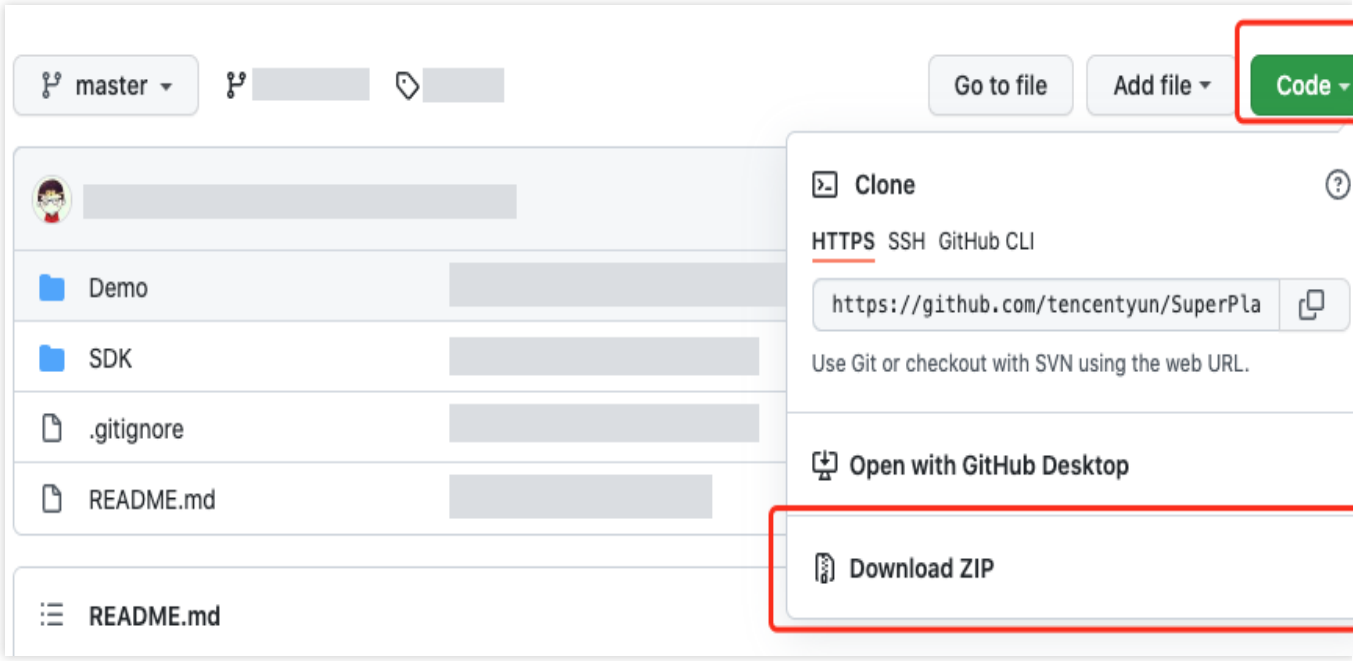
How to integrate the Player component for iOS
How to create and use a player

Prerequisites

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 Git command

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



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

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

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

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

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

Filename	Description
SDK	The folder of the Player component's frameworks and static libraries.
Demo	The folder of the Player demo.
App	The entry point UI.
SuperPlayerDemo	The Player demo.

SuperPlayerKit	The Player component.

Step 2. Integrate the component

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

Integrate via CocoaPods

Manually download the SDK

1. To install the component using CocoaPods, add the code below to 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 Player Premium edition, add the following dependency to `podfile` :

```
pod 'SuperPlayer/Player_Premium'
```

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

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

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

1. Download the SDK and demo at GitHub.

2. Import `TXLiteAVSDK_Player_Premium.framework` into your project and select **Do Not Embed**.

3. Copy `Demo/TXLiteAVDemo/SuperPlayerKit/SuperPlayer` to your project directory.

4. The third-party libraries the player depends on are `AFNetworking` , `SDWebImage` , `Masonry` , and `TXLiteAVSDK_Player` .

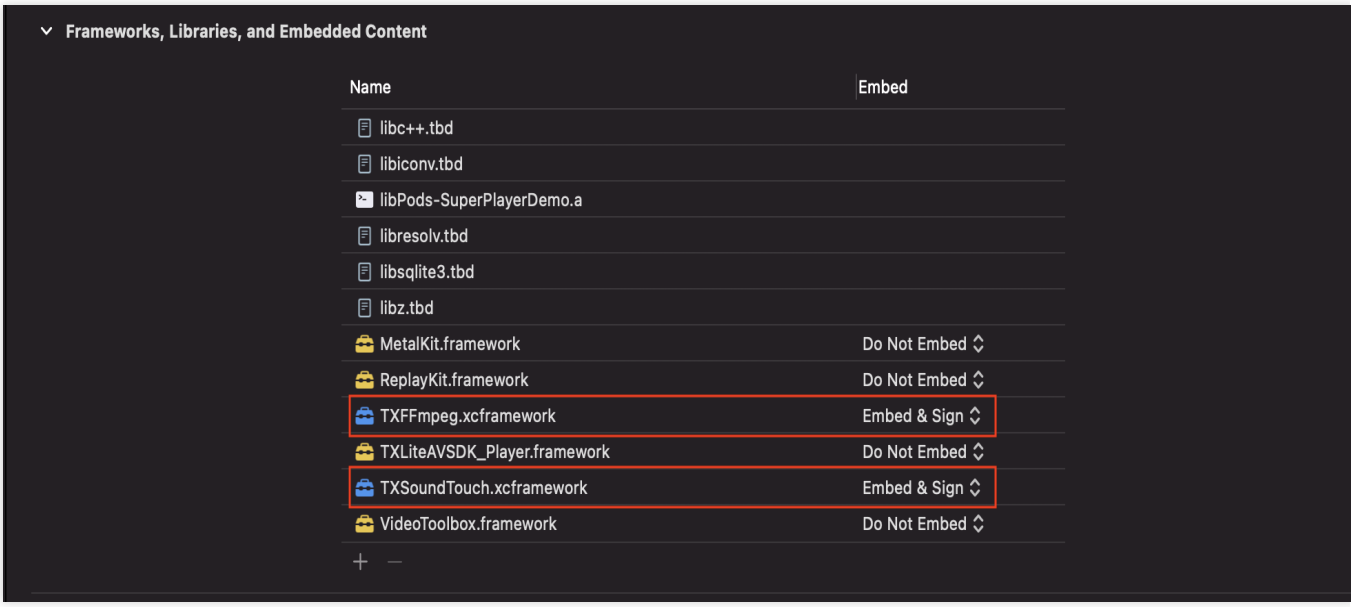
To integrate `TXLiteAVSDK_Player` manually, you need to add the required system frameworks and libraries:

System frameworks: MetalKit, ReplayKit, SystemConfiguration, CoreTelephony, VideoToolbox, CoreGraphics, AVFoundation, Accelerate, and MobileCoreServices.

System libraries: libz, libresolv, libiconv, libc++, and libsqlite3.

For detailed directions, see Manually integrate the SDK.

In addition, you need to add `TXFFmpeg.xcframework` and `TXSoundTouch.scframework` under the `TXLiteAVSDK_Player` file as dynamic libraries.



5. If you integrate `TXLiteAVSDK_Player` as a pod, no libraries need to be added.

Step 3. Use the player features

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

1. **Create a player**

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

```
// 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 obtained a license, you can view the license URL and key in the VOD console.

If you don't have the required license yet, you can get it as instructed in Video Playback License.

After obtaining the License information, you need to initialize and configure the License before calling the relevant interfaces of the SDK. For detailed tutorials, please see Configuring View License.

3. **Video playback:**

This step describes how to play back a video. The 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

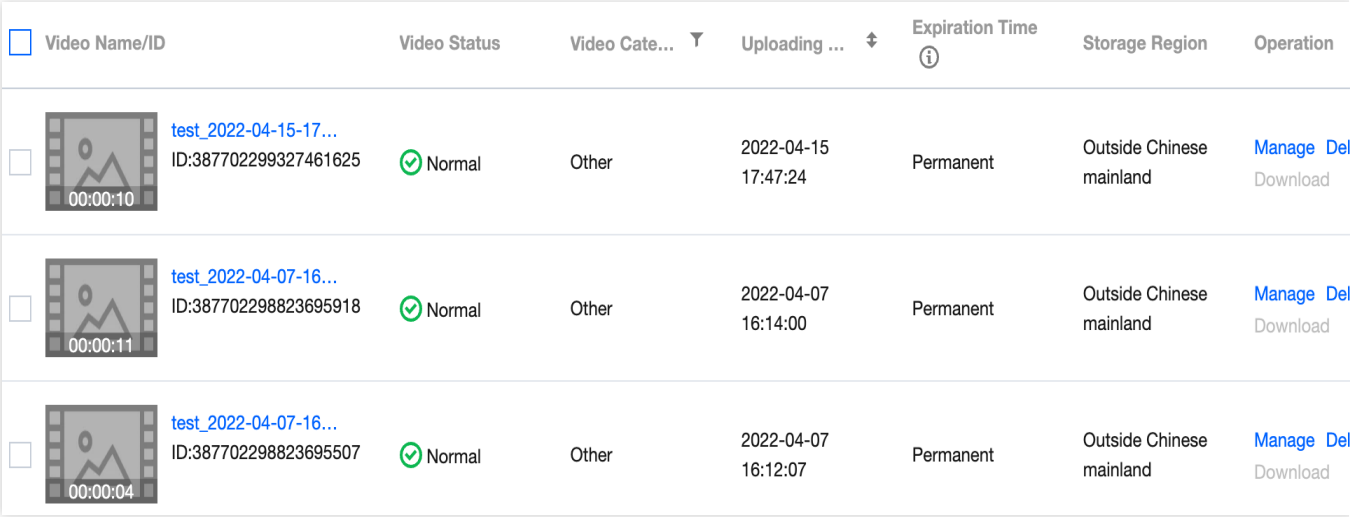
Play Local video

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

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

2. After a video is uploaded to the server, the notification for successful upload will contain a file ID for the video.

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



Note:

1. 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.

2. If a "no v4 play info" error occurs, it indicates that you haven't transcoded the video or used the player signature correctly. Troubleshoot the issue according to the above documents or get the playback URL of the video and play it by URL.

3. **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, transcode your video using the Adaptive-HLS template (ID: 10) or get
the playback URL of the video and play it by URL.

SuperPlayerModel *model = [[SuperPlayerModel alloc] init];
model.appId = 1400329071;// Configure AppId
model.videoId = [[SuperPlayerVideoId alloc] init];
model.videoId.fileId = @"5285890799710173650"; // Configure `FileId`
// `psign` is a player signature. For more information on the signature and how
to generate it, see [Player Signature]
(https://www.tencentcloud.com/document/product/266/38099).
model.videoId.pSign =
@"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTQwMDMyOTA3MSwiZmlsZUlkIjoi
```

```
NTI4NTg5MDc5OTcxMDE3MzY1MCIsImN1cnJlbnRUaW1lU3RhbXAiOjEsImV4cGlyZVRpbWVTdGFtcCI
6MjE0NzQ4MzY0NywidXJsQWNjZXNzSW5mbyI6eyJ0IjoiN2ZmZmZmZmYifSwiZHJtTGljZW5zZUluZm
8iOnsiZXhwaXJlVGltZVN0YW1wIjoyMTQ3NDgzNjQ3fX0.yJxpnQ2Evp5KZQFfuBBK05BoPpQAzYAWo
6liXws-LzU";
[_playerView playWithModelNeedLicence:model];
```

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

```
SuperPlayerModel *model = [[SuperPlayerModel alloc]init];
//Add your video file to the project, and then get the file path of the video throu
NSString *filePath = [[NSBundle mainBundle] pathForResource:@"your_video_name" ofTy
model.videoURL = [filePath stringByReplacingOccurrencesOfString:@"file://" withStri
[_playerView playWithModelNeedLicence:model];
```

4.Stop playback

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

```
[_playerView resetPlayer];
```

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

More Features

1. Full screen playback

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

You can call the API below to enter full screen from the windowed playback mode:

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

Features of full screen playback mode

Back to windowed mode

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;
```

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

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

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

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

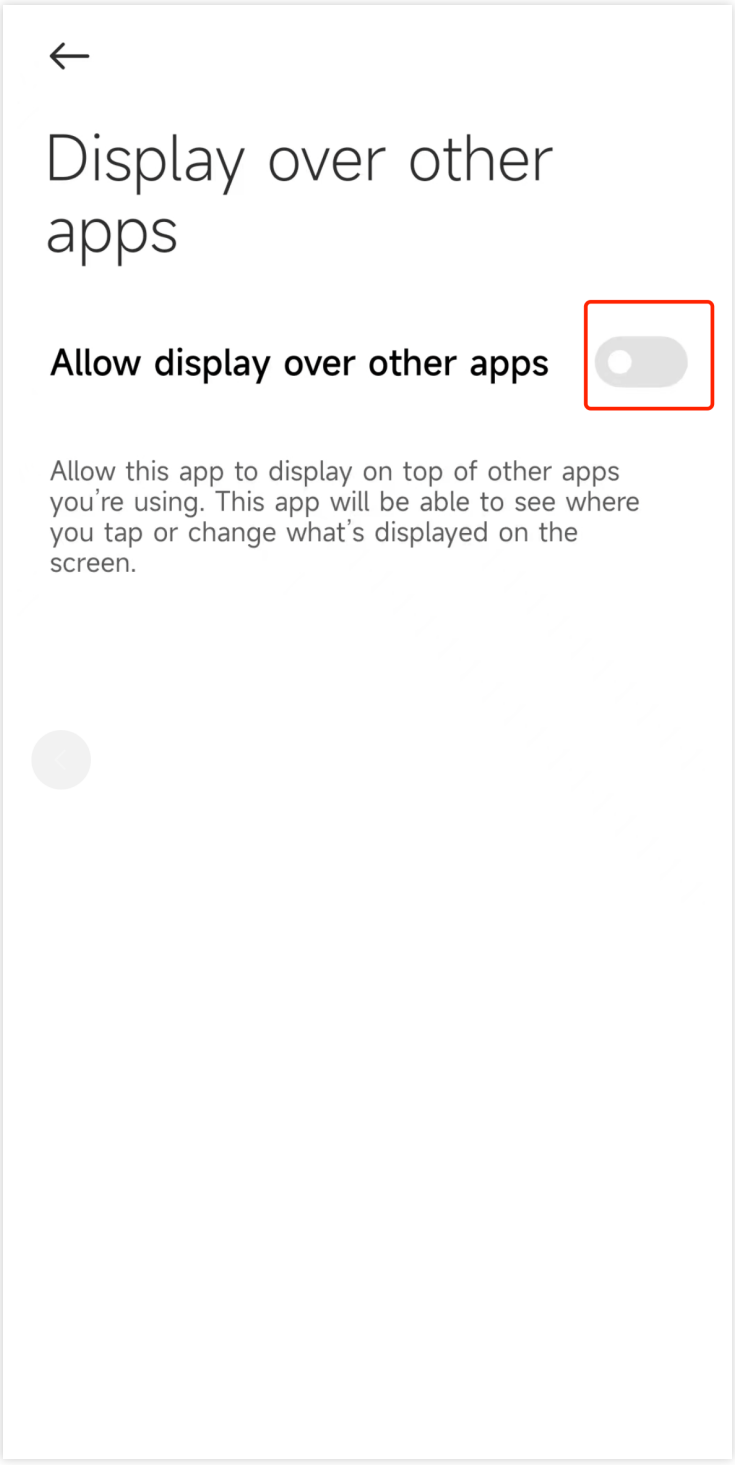
CFDanmakuView: Configure the attributes of on-screen commenting during initialization.

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

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

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

The Player component allows users to take and save a screenshot of a video during playback. The SDK will handle the screenshot button tapping event and no callbacks will be sent for successful or failed screenshots. Screenshots are saved to the phone album.
Users can change the video definition (such as SD, HD, and FHD) during playback. After the definition selection button is tapped, the SDK will implement the logic for displaying the definition selection view and handle the selection event. No callbacks will be sent.

2. Floating window playback

The Player component supports playback in a small floating window, which allows users to switch to another page of the application without interrupting the video playback. You can try out this feature in TCToolkit App > **Player** > **Player Component** by tapping **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 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`, videos are played only after users tap the play button, and 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
model.action  = PLAY_ACTION_MANUAL_PLAY;
// Specify the URL of an online file to use as the thumbnail. If `coverPictureUrl`
model.customCoverImageUrl = @"http://1500005830.vod2.myqcloud.com/6c9a5118vodcq1500
[self.playerView playWithModelNeedLicence:model];
```

4. Video playlist loop

The Player component supports looping video playlists.

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 startIn

(void)playWithModelListNeedLicence:(NSArray *)playModelList isLoopPlayList:(BOOL)is
```

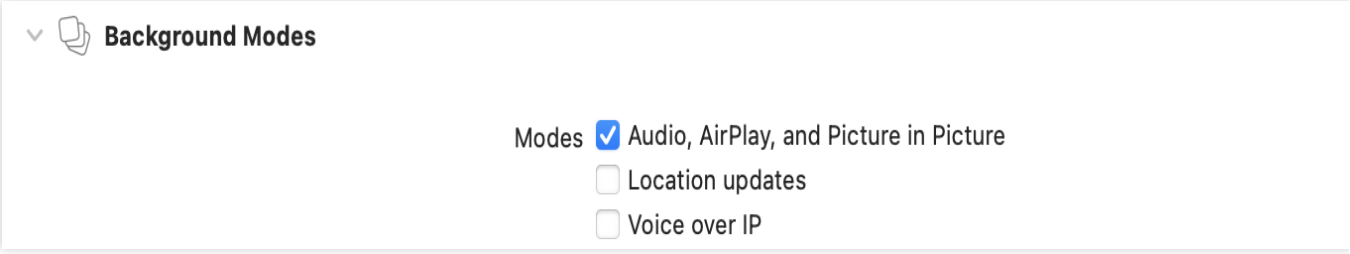
API parameters:

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

5. Picture-in-Picture (PiP) feature

The Picture-in-Picture (PiP) feature has been launched on iOS 9 but can currently be used only on iPads. To use PiP on an iPhone, you need to update the iOS version to iOS 14.

The Player component supports both in-app PiP and system-wide PiP. To use the feature, you need to enable background modes: In Xcode, choose your target, click **Signing & Capabilities** > **+Capability** > **Background Modes**, and select **Audio, AirPlay, and Picture in Picture**.



Code sample for using PiP capabilities:

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

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

6. Video preview

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

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

`TXVipWatchModel` class parameter description:

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

7. Dynamic watermark

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

```
// Step 1. Create a video source information model
SuperPlayerModel *  playermodel   = [SuperPlayerModel new];
// Add other information of the video source
// Step 2. Create a dynamic watermark model
DynamicWaterModel *model = [[DynamicWaterModel alloc] init];
// Step 3. Set the data of the dynamic watermark
model.dynamicWatermarkTip = @"shipinyun";
model.textFont = 30;
model.textColor = [UIColor colorWithRed:255.0/255.0 green:255.0/255.0 blue:255.0/25
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

8. Video download

Video download allows users to cache online videos and watch them offline. The cached video can be played back only in the client but cannot be actually downloaded to the device. This feature can effectively prevent downloaded videos from being distributed without authorization and protect the video security.
You can try out this feature in full screen mode in TCToolkit App > Player > Player Components > Offline Cache.

`VideoCacheView` (cache selection list view) is used to select and download videos at different definitions. After selecting the definition in the top-left corner, click the option of the video to be downloaded. When a check mark appears, the download has started. After clicking the **video download list** button below, you will be redirected to the Activity where `VideoDownloadListView` is located.

```
// Step 1. Initialize the cache selection list view
//@property (nonatomic, strong) VideoCacheView *cacheView;
_cacheView = [[VideoCacheView alloc] initWithFrame:CGRectZero];
_cacheView.hidden = YES;
[self.playerView addSubview:_cacheView];
```

```
// Step 2. Set the options of the video being played back
[_cacheView setVideoModels:_currentPlayVideoArray currentPlayingModel:player.player

// Click event of the **video download list** button
- (UIButton *)viewCacheListBtn;
```

```
- (void)setVideoModels:(NSArray *)models currentPlayingModel:(SuperPlayerModel *)cu
```

API parameters:

Parameter	Type	Description
models	NSArray	The video data model of the download list
SuperPlayerModel	currentModel	The video data model of the video being played back

`VideoCacheListView` (video download list)

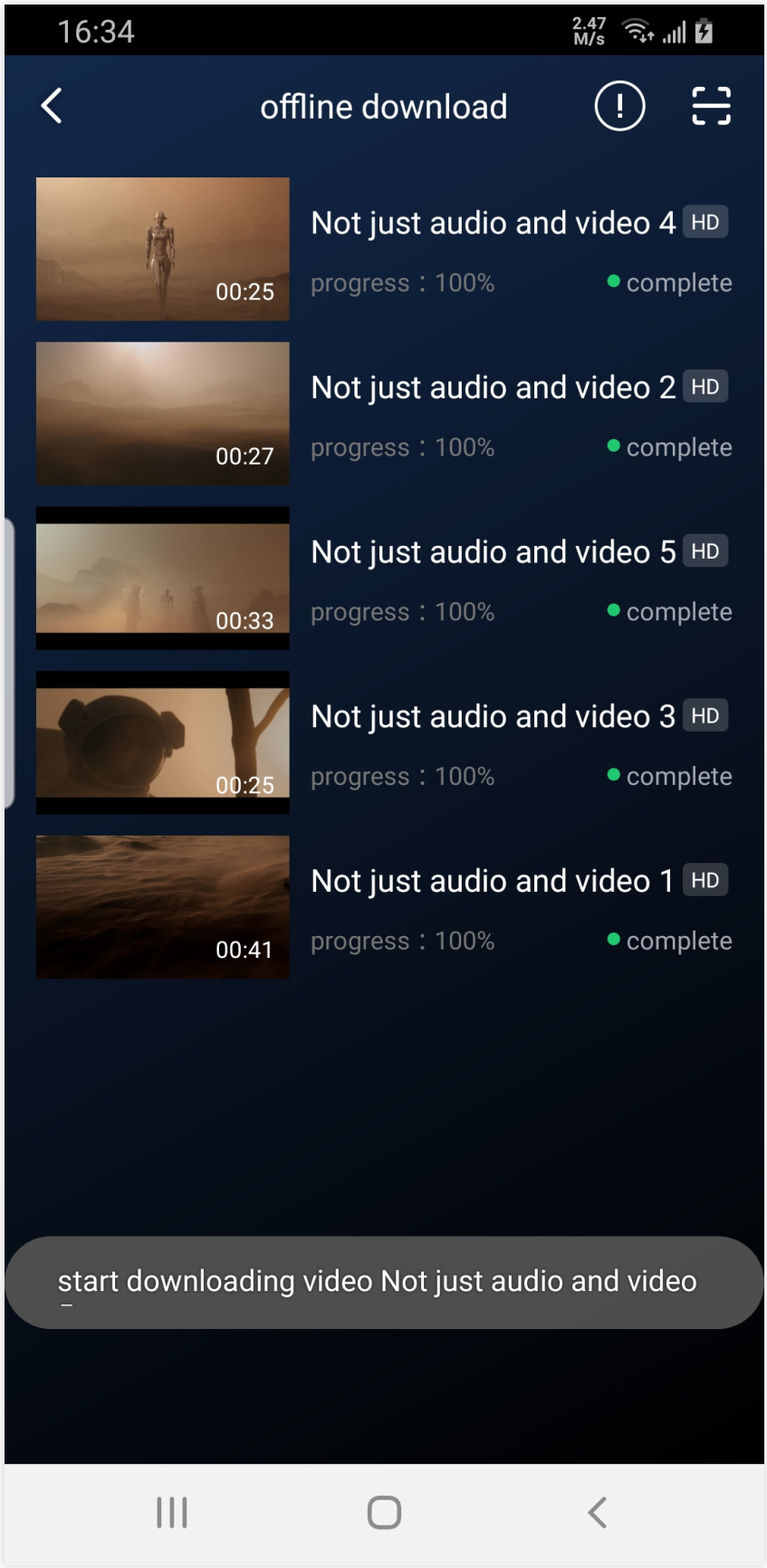
displays the list of views of all the videos that are being downloaded and have been downloaded.

When this button is clicked：

if the download is in progress, it will be paused.

if it is paused,it will be resumed.

if it has completed, the video will be played back.




```
// Add data, which is obtained from the `TXVodDownloadManager#getDownloadMediaInfoL
NSArray<TXVodDownloadMediaInfo *> *array = [[[TXVodDownloadManager shareInstance] g
```

```
for (TXVodDownloadMediaInfo *info in array) {
    VideoCacheListModel *model = [[VideoCacheListModel alloc] init];
    model.mediaInfo = info;
    [self.videoCacheArray addObject:model];
}

// List items support operations such as click to play and hold and press to delete
- (void)longPress:(UILongPressGestureRecognizer *)longPress;  // Hold and press
```

The downloaded video supports playing without network connection, please refer to the following code when playing:

```
NSArray<TXVodDownloadMediaInfo *> *mediaInfoList = [[TXVodDownloadManager shareInst
TXVodDownloadMediaInfo *mediaInfo = [mediaInfoList firstObject];
SuperPlayerUrl *superPlayerUrl = [[SuperPlayerUrl alloc] init];
superPlayerUrl.title = @"*********";
superPlayerUrl.url = mediaInfo.playpath;
NSArray<SuperPlayerUrl *> *multiVideoURLs = @[superPlayerUrl];
SuperPlayerModel *playerModel = [[SuperPlayerModel alloc] init];
playerModel.multiVideoURLs = multiVideoURLs;
[self.playerView playWithModelNeedLicence:playerModel];
```

Note：

When video files are downloaded without network playback, be sure to obtain the download list and play through the PlayPath of the video object TXVodDownloadMediaInfo in the download list, and do not directly save the PlayPath object.

9. Image sprite and timestamp information

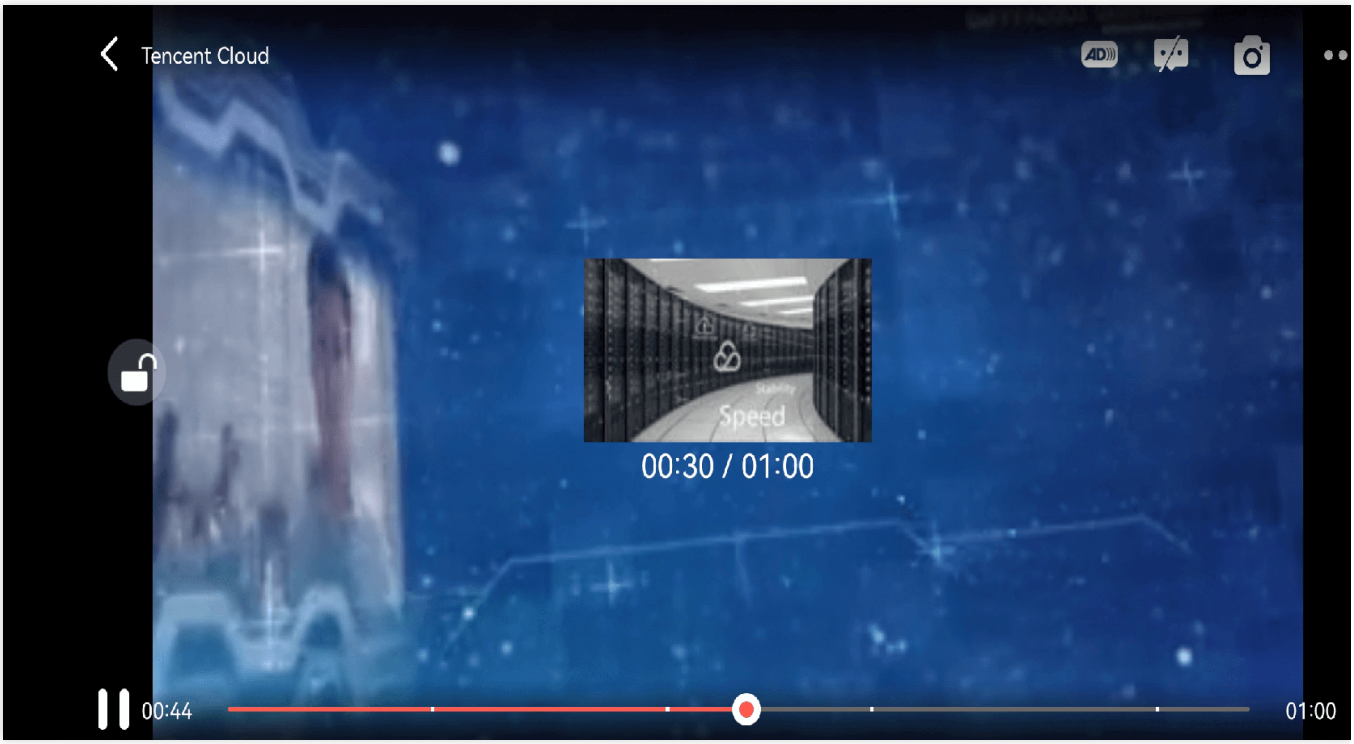
Timestamp information

You can add text descriptions at key positions on the progress bar, which the user can click to view and quickly understand the video information at the current position. After clicking the video information, the user can seek to the desired position.

You can try out this feature in full screen mode in TCToolkit App > Player > Player Components > Tencent Cloud Video.

Image sprite

Users can view video thumbnails when dragging or seeking on the progress bar so as to quickly understand the video content at the specified position. The thumbnail preview is implemented based on the video's image sprite. You can generate the image sprite of a video file in the VOD console or directly generate an image sprite file.
You can try out this feature in full screen mode in TCToolkit App > Player > Player Components > Tencent Cloud Video.




```
// Step 1. Get the image sprite and timestamp information in the `onPlayEvent` call
[self.playerView playWithModelNeedLicence:playerModel];

// Step 2. Get keyframes and image sprite information in the `VOD_PLAY_EVT_GET_PLAY
NSString *imageSpriteVtt = [param objectForKey:VOD_PLAY_EVENT_IMAGESPRIT_WEBVTTURL]
NSArray<NSString *> *imageSpriteList = [param objectForKey:VOD_PLAY_EVENT_IMAGESPRI
NSArray<NSURL *> *imageURLs = [self convertImageSpriteList:imageSpriteList];
[self.imageSprite setVTTUrl:[NSURL URLWithString:imageSpriteVtt] imageUrls:imageURL

// Step 3. Display the obtained timestamp information and image sprite on the UI
if (self.isFullScreen) {
   thumbnail = [self.imageSprite getThumbnail:draggedTime];
}
if (thumbnail) {
   [self.fastView showThumbnail:thumbnail withText:timeStr];
}
```

10. External subtitles

Note:

External subtitles depend on the premium version SDK of the media player, and the SDK needs to be version 11.3 or above to support it.

Currently, only SRT and VTT subtitle formats are supported. The usage is as follows:

Step 1: Add external subtitles.

Pass the external subtitle category field to `SuperPlayerModel#subtitlesArray` .

```
// // Pass in subtitle url, subtitle name, and subtitle type
SuperPlayerSubtitles *subtitleModel = [[SuperPlayerSubtitles alloc] init];
subtitleModel.subtitlesUrl = @"https://mediacloud-76607.gzc.vod.tencent-cloud.com/D
subtitleModel.subtitlesName = @"ex-cn-srt";
subtitleModel.subtitlesType = 0;
[subtitlesArray addObject:subtitleModel];

// Play
[self.playerView playWithModelNeedLicence:model];
```

Step 2: Switch subtitles after playback.

```
// After the video starts playing, select the added external subtitles.
- (void)controlViewSwitch:(UIView *)controlView withSubtitlesInfo:(TXTrackInfo *)in
    if (info.trackIndex == -1) {
        [self.vodPlayer deselectTrack:preInfo.trackIndex];
        self->_lastSubtitleIndex = -1;
    } else {
        if (preInfo.trackIndex != -1) {
            // Deselect other subtitles if they are not needed
            [self.vodPlayer deselectTrack:preInfo.trackIndex];
        }
        // Select subtitles. [self.vodPlayer
        [self.vodPlayer selectTrack:info.trackIndex];
        self->_lastSubtitleIndex = info.trackIndex;
    }
}
```

Step 3: Configure subtitle styles.

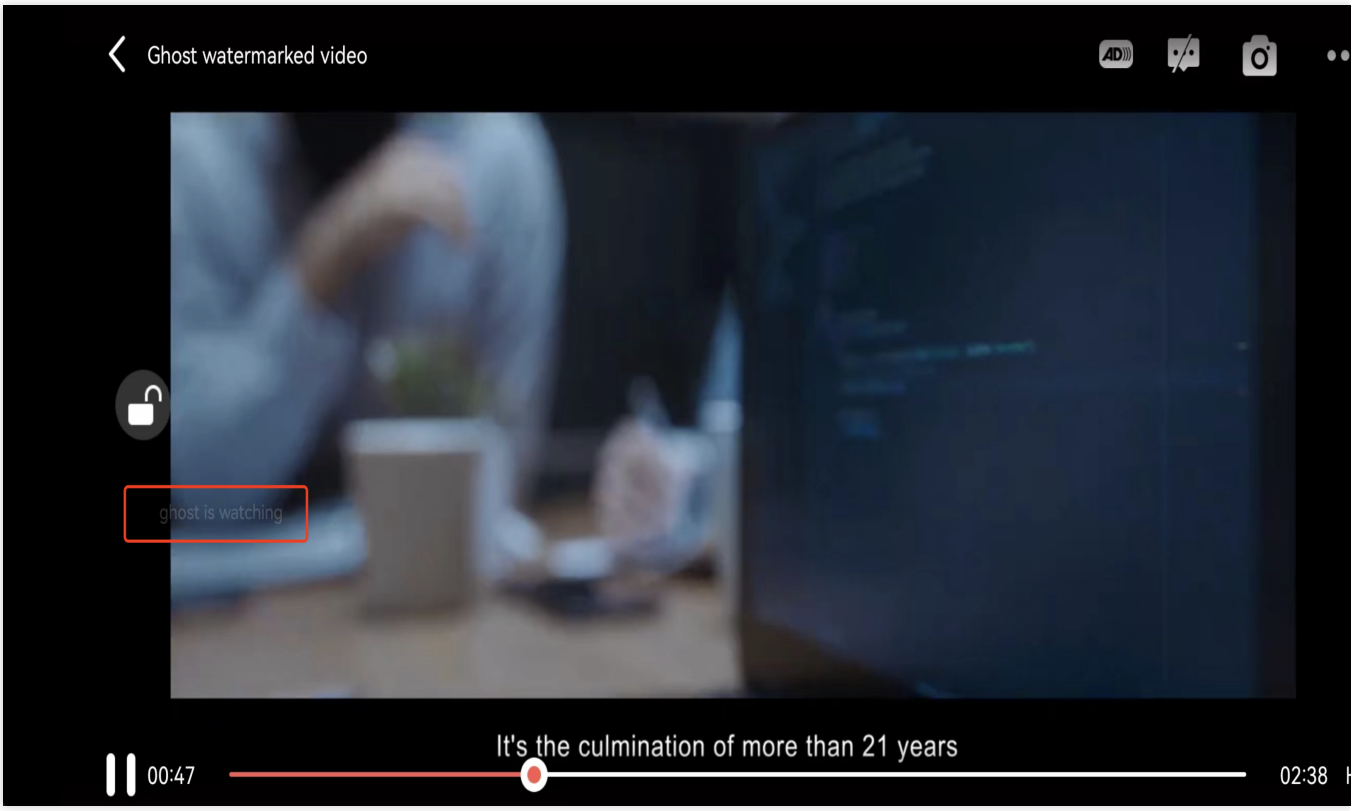
Subtitle styles can be configured before or during playback.

```
TXPlayerSubtitleRenderModel *model = [[TXPlayerSubtitleRenderModel alloc] init];
model.canvasWidth = 1920;   // Subtitle render canvas width
model.canvasHeight = 1080; // Subtitle render canvas height
model.isBondFontStyle = NO;  // Set whether the subtitle font is bold
model.fontColor = 0xFF000000; // Set the subtitle font color, default white and opa
[_txVodPlayer setSubtitleStyle:model];
```

11. Ghost watermark

The content of the ghost watermark is filled in the player signature and is ultimately displayed on the playback end through collaboration between the cloud and the player, ensuring the security of the watermark throughout the transmission process. Follow the tutorial to configure the ghost watermark in the player signature. The ghost watermark only appears on the video for a very short time, and this flashing has a minimal impact on viewing the video. The position of each watermark appearance is not fixed, which eliminates attempts by others to cover up the watermark. The effect is shown in the figure below. A watermark appears once when the video starts playing, and then disappears. The content of the ghost watermark can be obtained through [param objectForKey:@"EVT_KEY_WATER_MARK_TEXT"] after receiving the VOD_PLAY_EVT_GET_PLAYINFO_SUCC event from the player.

Note: Supported from player version 11.6.




```
// Step 1: Configure the FileId that supports ghost watermark to play the video.
SuperPlayerModel *model = [[SuperPlayerModel alloc] init];
model.appId = 1500006438;
model.videoId = [[SuperPlayerVideoId alloc] init];
model.videoId.fileId = @"3877023078471291 27";
model.videoId.pSign =
@"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTUwMDAwNjQzOCwiZmlsZUlkIjoiMzg3
[_playerView playWithModelNeedLicence:model];

// Step 2: After SuperPlayerView receives the PLAY_EVT_GET_PLAYINFO_SUCC event, obt
- (void)onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary
    dispatch_async(dispatch_get_main_queue(), ^{
        if (EvtID == PLAY_EVT_PLAY_EVT_GET_PLAYINFO_SUCCPLAY_PROGRESS) {
            NSString *ghostWaterText = [param objectForKey:@"EVT_KEY_WATER_MARK_TEX
            if (ghostWaterText && ghostWaterText.length > 0) {
                DynamicWaterModel *model = [[DynamicWaterModel alloc] init];
                model.showType = ghost;
                model.duration = self.playerModel.duration;
                model.dynamicWatermarkTip = ghostWaterText;
                model.textFont = 30;
                model.textColor = [UIColor redColor];
                if (![self.subviews containsObject:self.watermarkView]) {
                    [self addSubview:self.watermarkView];
                    [self.watermarkView mas_makeConstraints:^(MASConstraintMaker *m
```

```
                        make.edges.equalTo(self);
                    }];
                }
                [self.watermarkView setDynamicWaterModel:model];
            }
        }
    };
}
```

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**.
During the application upgrade and maintenance, the demo source code can still be used normally.

Android Integration Guide

Last updated：2024-10-08 11:34:06

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 requirements, and you have some knowledge of engineering, you can integrate the Player SDK to customize the UI and playback features.

Limits

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.

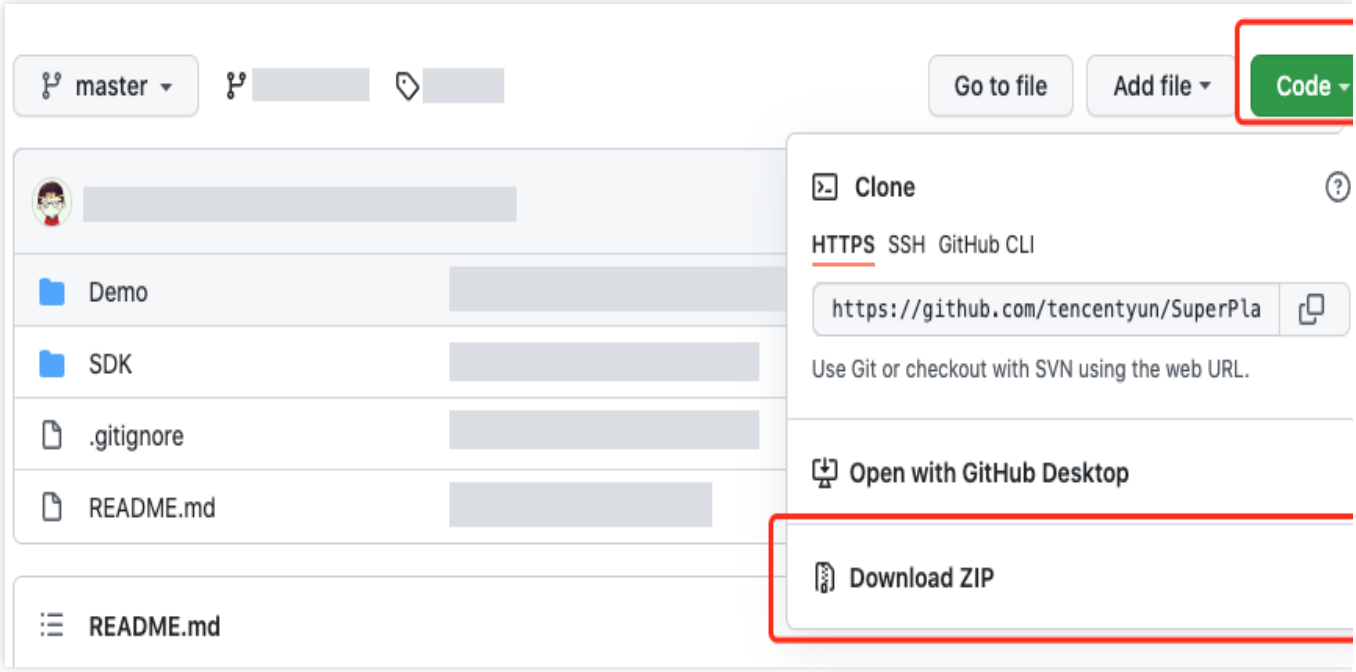
This Document Describes

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

Prerequisites

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



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

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

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

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

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

Filename	Description
LiteAVDemo(Player)	The Player demo project, which can be run directly after being imported into Android Studio.
app	Homepage entry
superplayerkit	The Player component (SuperPlayerView), which provides common features such as playback, pause, and gesture control.
superplayerdemo	The Player component demo code

common	Tool module
SDK	Player SDK, including `LiteAVSDK_Player_x.x.x.aar` (SDK provided in AAR format) and `LiteAVSDK_Player_x.x.x.zip` (SDKs provided in lib and JAR formats)
Player Documentation (Android).pdf	The Player component user guide

Step 2. Integrate the component

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

Automatic loading in Gradle (AAR)

Manual download in Gradle (AAR)

SDK integration (jar + so)

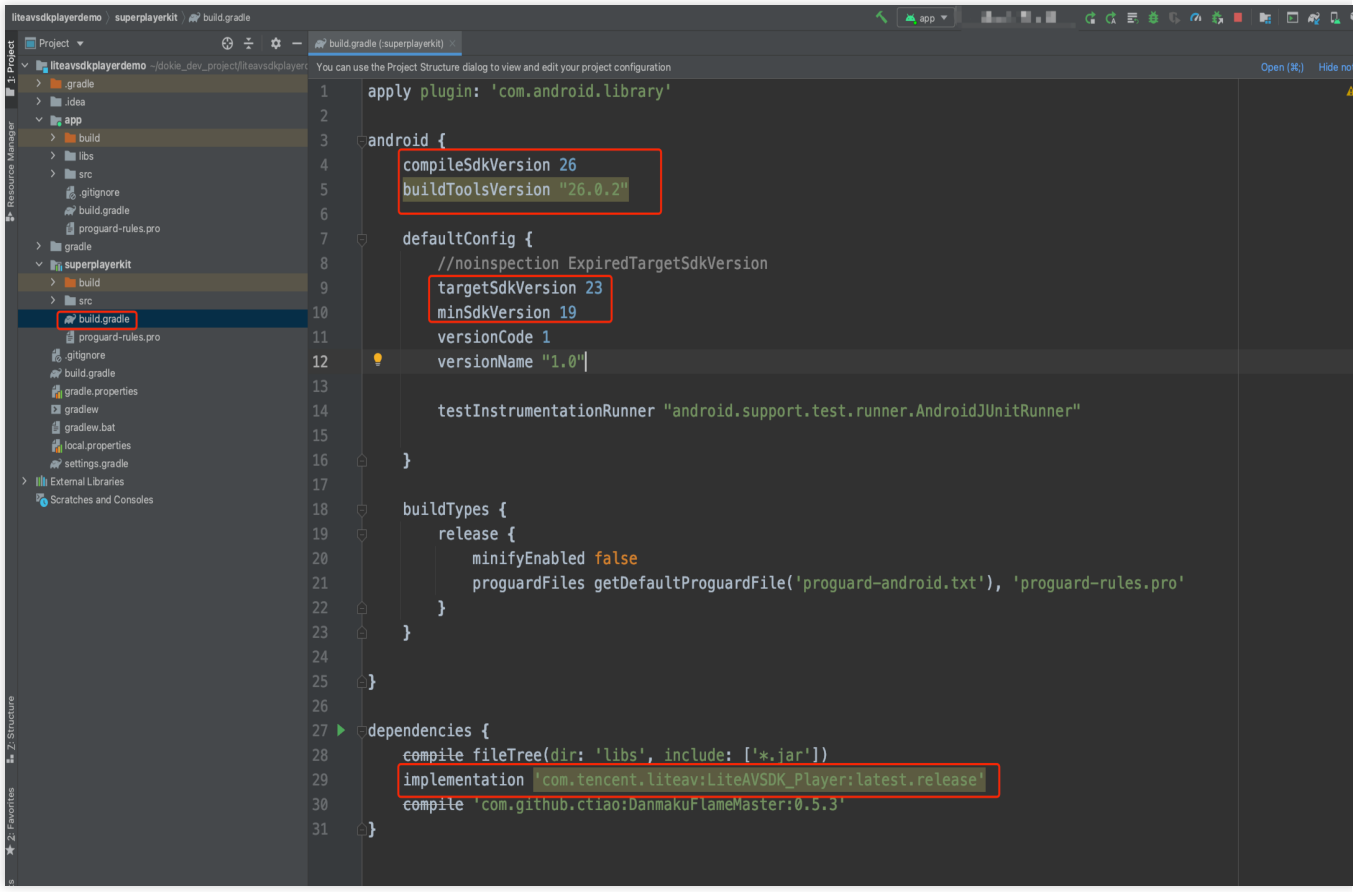
1. Download the SDK + demo package for Android here.

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`.

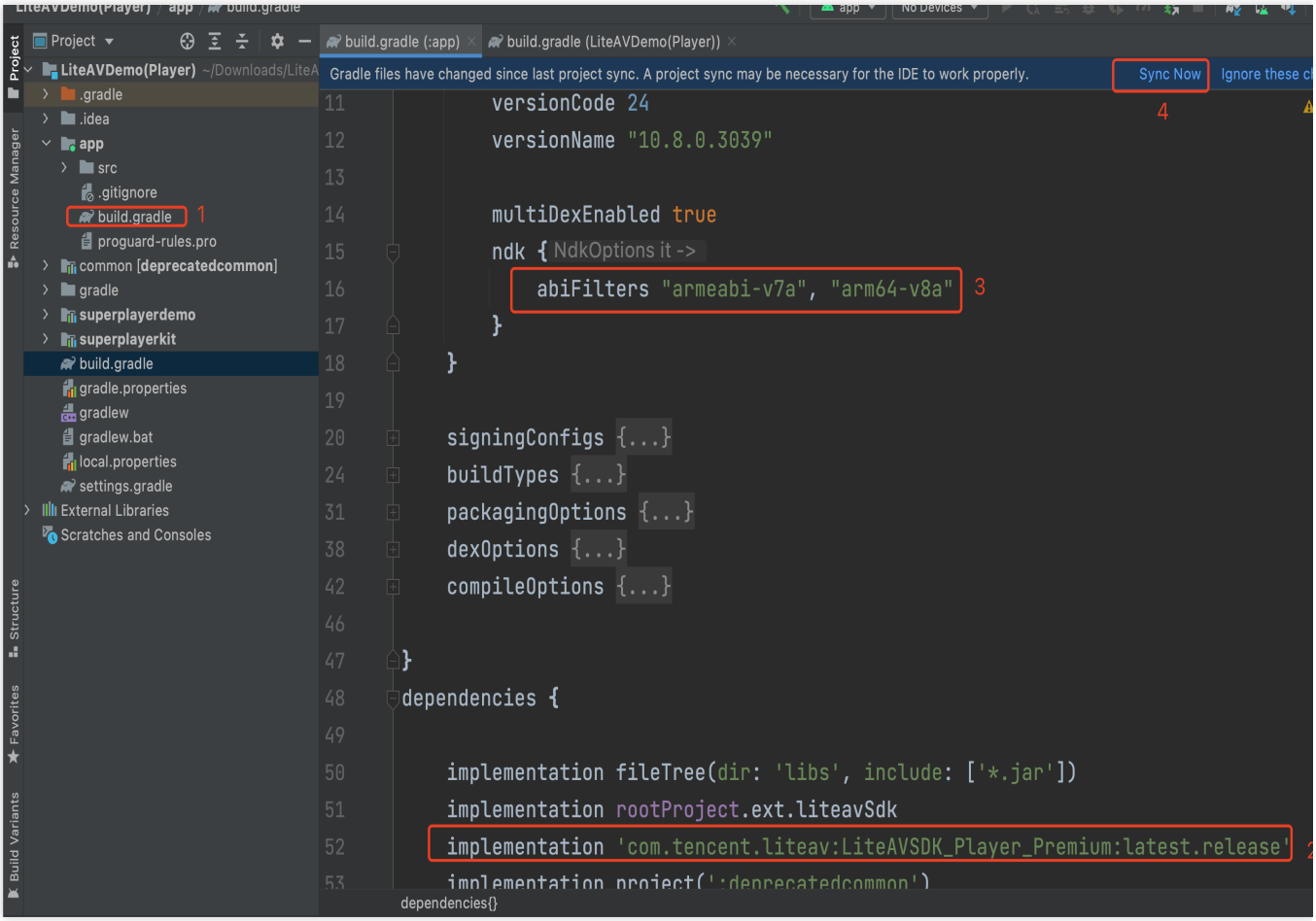



```
compileSdkVersion 26
buildToolsVersion "26.0.2"

defaultConfig {
  targetSdkVersion 23
  minSdkVersion 19
}

dependencies {
  // To integrate an older version, change `latest.release` to the corresponding ve
  implementation 'com.tencent.liteav:LiteAVSDK_Player_Premium:latest.release'

  // If you want to integrate the basic version of the player
  // 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` library in Gradle, and LiteAVSDK will be automatically downloaded and updated.

Open `app/build.gradle` and configure as follows:



3.1 Add the `LiteAVSDK_Player_Premium` dependencies to `dependencies`.

```
dependencies {
    implementation 'com.tencent.liteav:LiteAVSDK_Player_Premium:latest.release'
    // If you want to integrate the basic version of the player
    // implementation 'com.tencent.liteav:LiteAVSDK_Player:latest.release'
    implementation project(':superplayerkit')
}
```

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

```
dependencies {
    // Integrate the LiteAVSDK_Player_Premium SDK v10.8.0.29000
    implementation 'com.tencent.liteav:LiteAVSDK_Player_Premium:10.8.0.29000'

    // If you want to integrate the basic version of the player
    // implementation 'com.tencent.liteav:LiteAVSDK_Player:latest.release'
}
```

4. 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"
}
```

5. Add the `mavenCentral` library to the `build.gradle` in your project directory.

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

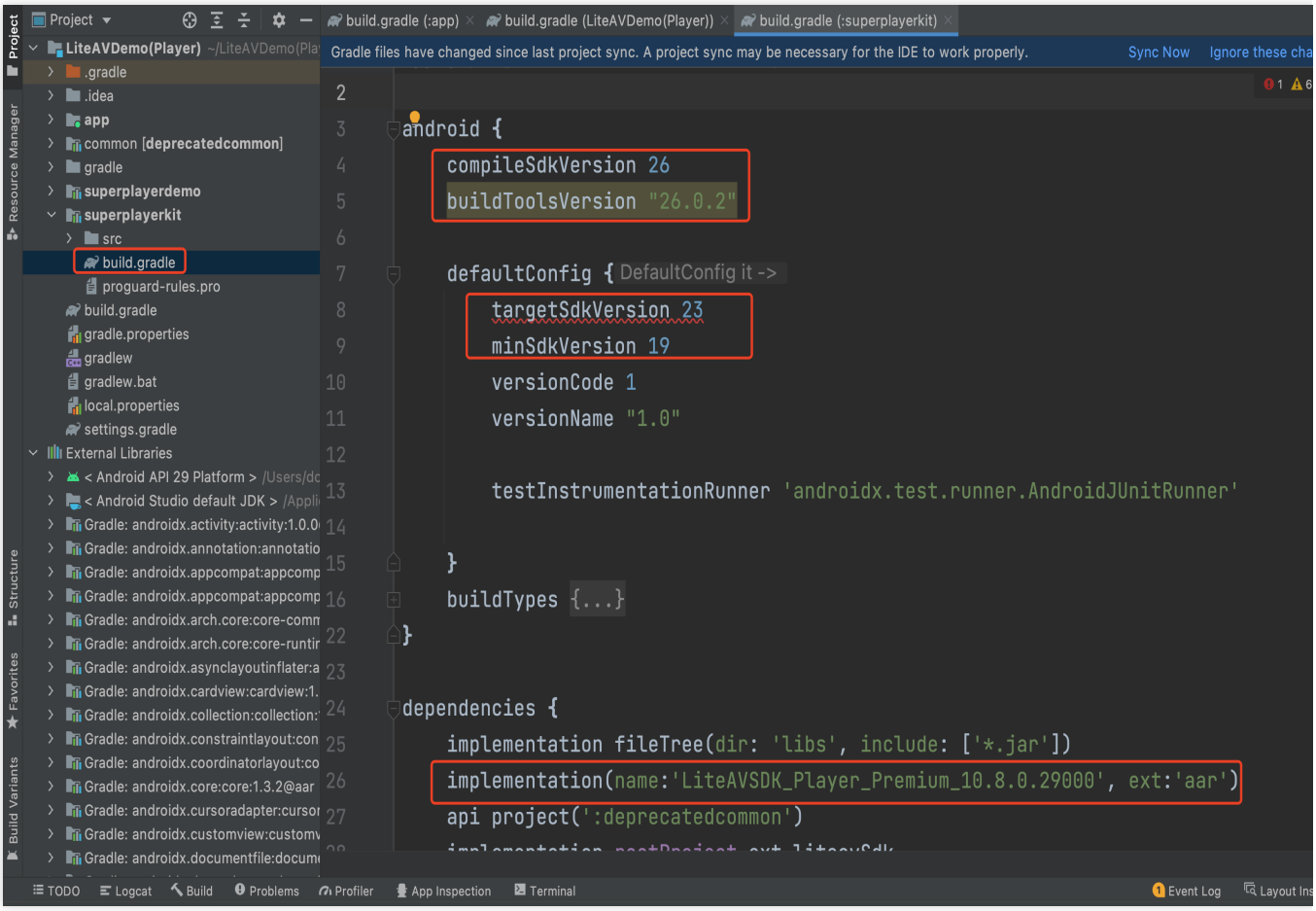
6. Click 

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.

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

```
include ':superplayerkit'
```

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




```
compileSdkVersion 26
buildToolsVersion "26.0.2"

defaultConfig {
  targetSdkVersion 23
  minSdkVersion 19
}

dependencies {
  implementation(name:'LiteAVSDK_Player_Premium_10.8.0.29000', ext:'aar')
}
```

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

Configure `repositories`

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

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

```
compile(name:'LiteAVSDK_Player_Premium_10.8.0.29000', ext:'aar')
implementation project(':superplayerkit')
// Third-party library for integration of the on-screen commenting feature of the P
implementation 'com.github.ctiao:DanmakuFlameMaster:0.5.3'
```

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

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

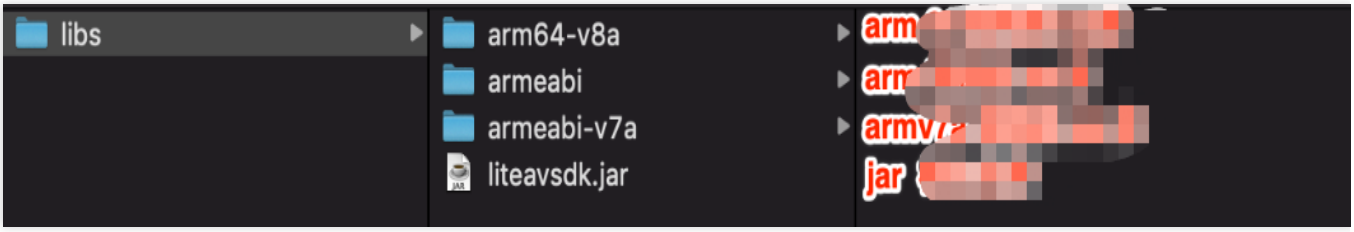
7. In `defaultConfig` of `app/build.gradle` , specify the CPU architecture to be used by the application (currently, LiteAVSDK supports armeabi, armeabi-v7a, and arm64-v8a).

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

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

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

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

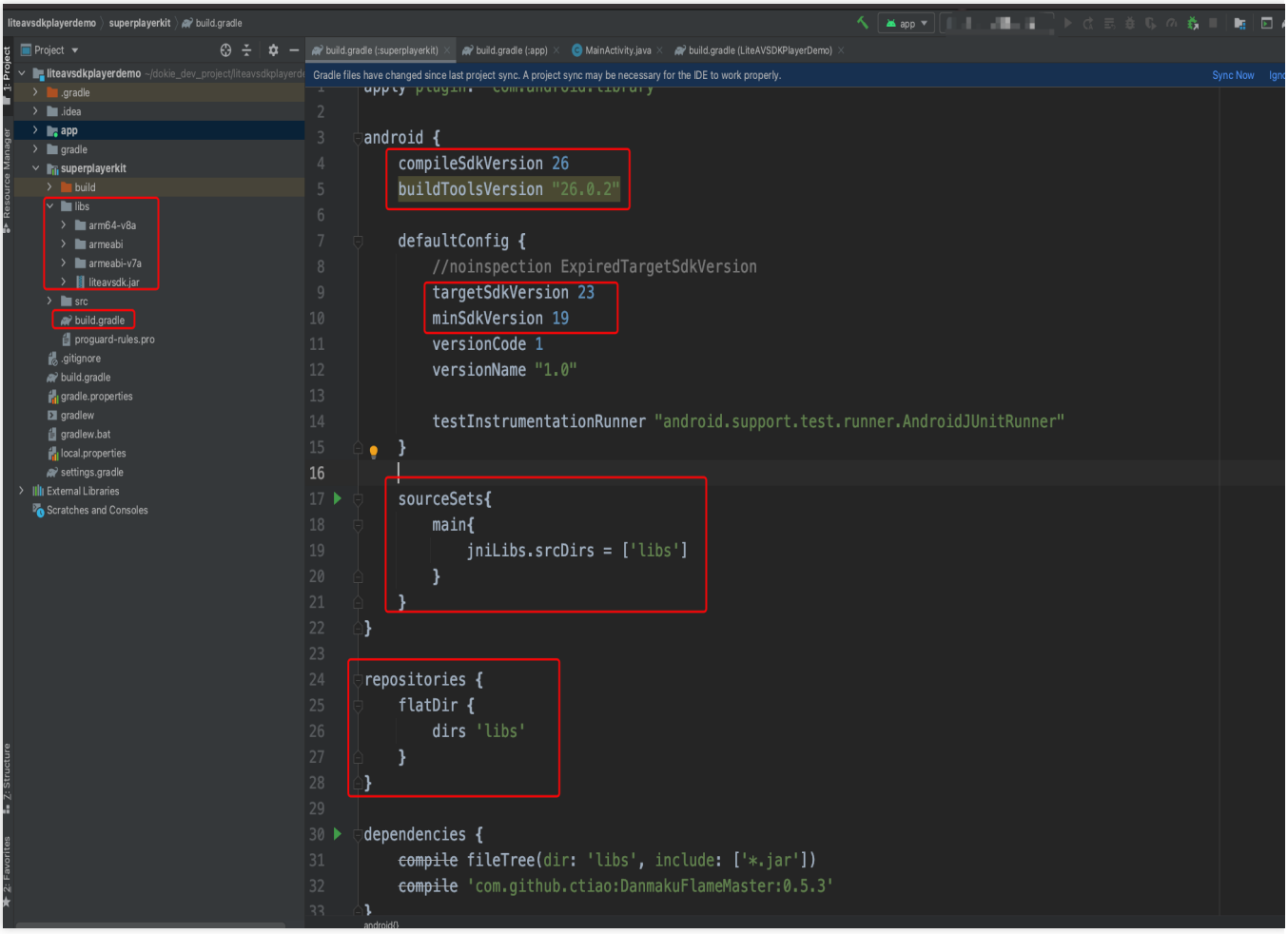


2. Copy the `Demo/superplayerkit` module to your project and import `superplayerkit` into `setting.gradle` in your project directory.

```
include ':superplayerkit'
```

3. Copy the `libs` folder obtained by decompression in step 1 to the `superplayerkit` project root directory.

4. Modify the `superplayerkit/build.gradle` file:




```
compileSdkVersion 26
buildToolsVersion "26.0.2"

defaultConfig {
  targetSdkVersion 23
  minSdkVersion 19
}
```

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

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

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

Configure `repositories`, add `flatDir`, and specify the paths of the local repositories.

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

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

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

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

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

Step 3. Configure application permissions

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

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

Network security configuration allows the app to send HTTP requests

For security reasons, starting from Android P, Google requires that requests from apps use encrypted connections. The player SDK will start a localsever proxy to make HTTP requests. If your app's targetSdkVersion is greater than or equal to 28, you can enable the permission to send HTTP requests to 127.0.0.1 through network security configuration. Otherwise, an "java.io.IOException: Cleartext HTTP traffic to 127.0.0.1 not permitted" error will occur during playback, causing the video to fail to play. The configuration steps are as follows:

1. Create a new res/xml/network_security_config.xml file in your project and set the network security configuration.

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">127.0.0.1</domain>
    </domain-config>
```

```
</network-security-config>
```

2. Add the following attributes to the application tag in the AndroidManifest.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:networkSecurityConfig="@xml/network_security_config"
                    ... >
        ...
    </application>
</manifest>
```

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 obtained a license, you can view the license URL and key in the VOD console.

If you don't have the required license yet, you can get it as instructed in Video Playback License.

After obtaining the License information, you need to initialize and configure the License before calling the relevant interfaces of the SDK. For

detailed tutorials, please see Configuring View License.

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.

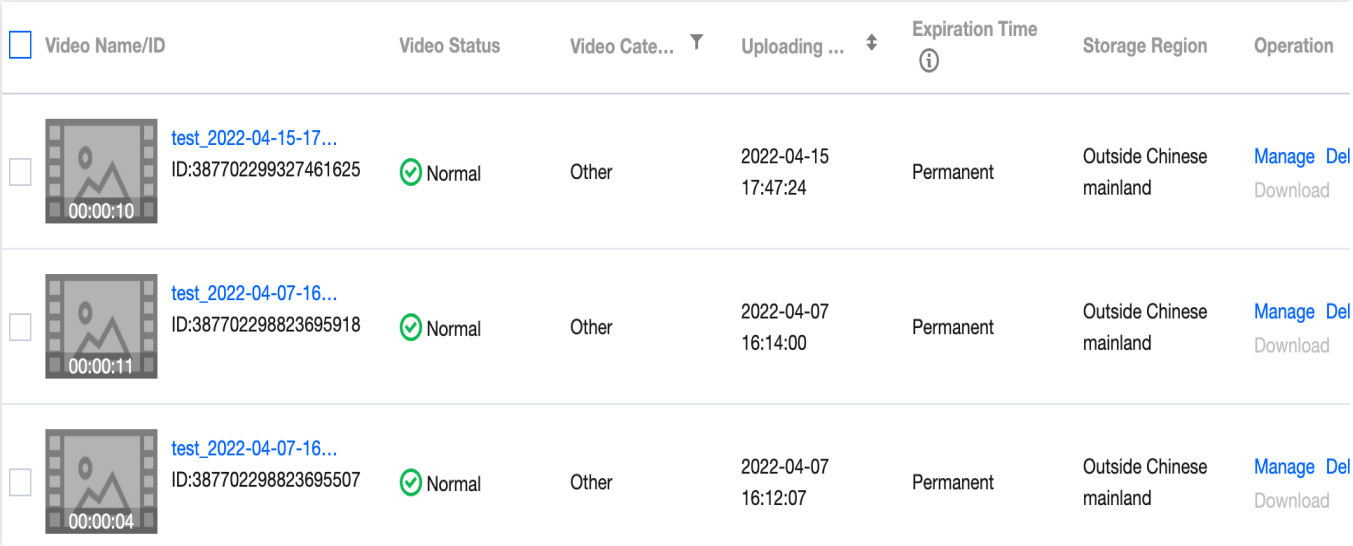
```
SuperPlayerModel model = new SuperPlayerModel();
model.appId = 1400329073; // Configure `AppId`
model.url = "http://your_video_url.mp4";   // Configure a URL for your video for pl
mSuperPlayerView.playWithModelNeedLicence(model);
```

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

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

2. After a video is uploaded to the server, the notification for successful upload will contain a file ID for the video.

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



Note

1. 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.

2. 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.

3.

We recommend you transcode videos for playback because untranscoded videos may experience compatibility issue s during playback.

```
// If you haven't enabled hotlink protection and a "no v4 play info" error occurs,

SuperPlayerModel model = new SuperPlayerModel();
model.appId = 1400329071;// Configure AppId
model.videoId = new SuperPlayerVideoId();
model.videoId.fileId = "5285890799710173650"; // Configure `FileId`
// `psign` is a player signature. For more information on the signature and how to
model.videoId.pSign = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTQwMDMyOTA
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 Features.

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

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

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

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

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

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

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

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

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

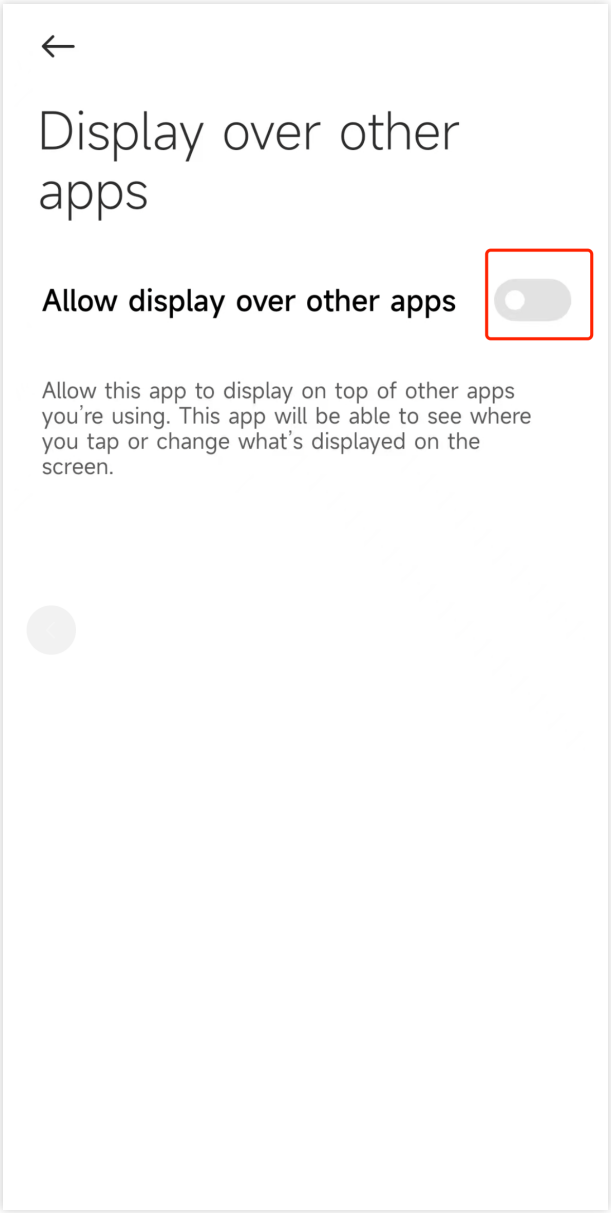
Users can change the video definition (such as SD, HD, and FHD) during playback.

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

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

2. Floating window playback

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



Floating window playback relies on the following permission in `AndroidManifest` :

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

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

3. Thumbnail

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

When the Player component is set to the automatic playback mode `PLAY_ACTION_AUTO_PLAY`, the 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`, videos are played only after users tap the play button, and 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
model.playAction = PLAY_ACTION_MANUAL_PLAY;
// Specify the URL of an online file to use as the thumbnail. If `coverPictureUrl`
model.coverPictureUrl = "http://1500005830.vod2.myqcloud.com/6c9a5118vodcq150000583
mSuperPlayerView.playWithModelNeedLicence(model);
```

4. Video playlist loop

The Player component supports looping video playlists.

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);
```

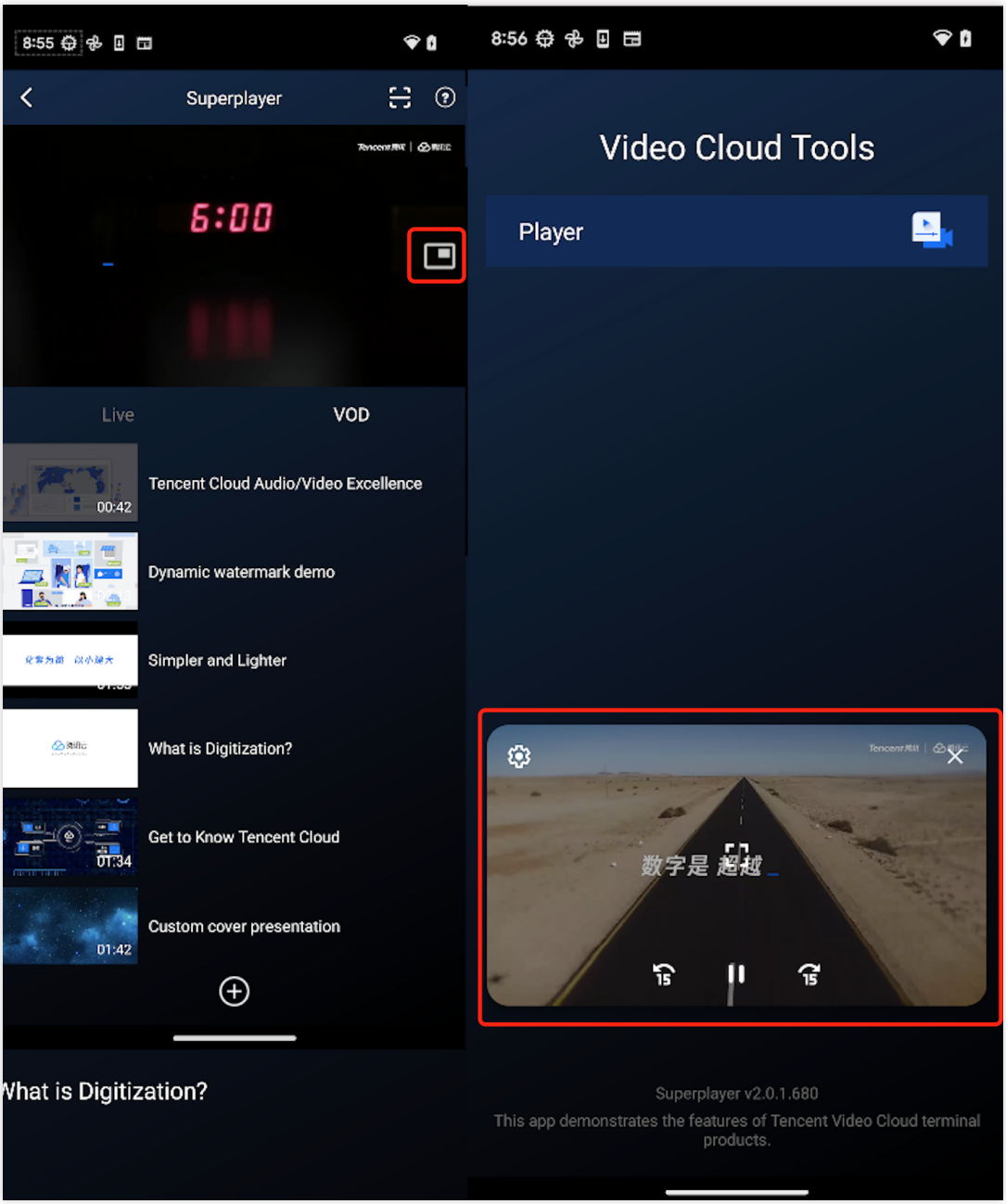
```
public void playWithModelListNeedLicence(List<SuperPlayerModel> models, boolean isL
```

API parameters:

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

5. Picture-in-picture

Starting from Android 8.0 (API level 26), Android allows launching activities in picture-in-picture (PiP) mode.



If you need to enable or disable picture-in-picture, simply change the value of enablePIP in SuperPlayerGlobalConfig. To add picture-in-picture to your app, you need to add the following attribute to the activity that supports picture-in-picture in AndroidManifest.

```
<activity>
    android:name=".demo.SuperPlayerActivity"
    android:resizeableActivity="true"
    android:supportsPictureInPicture="true"
    android:documentLaunchMode="intoExisting"
    android:excludeFromRecents="true"
    android:configChanges="orientation|keyboardHidden|screenSize|smallestScreenSize

</activity>
```

At the same time, the lifecycle of the activity that supports picture-in-picture needs to be handled specially according to SuperPlayerActivity. To enable picture-in-picture, use PictureInPictureHelper in SuperPlayerView.

```
PictureInPictureHelper mPictureInPictureHelper = new PictureInPictureHelper(mContex
mPictureInPictureHelper.setListener(this);
mPictureInPictureHelper.enterPictureInPictureMode(getPlayerState(),mTXCloudVideoVie
```

You need to release it in SuperPlayerView when exiting.

```
mPictureInPictureHelper.release();
```

If you need to modify the time interval for moving the custom button forward or backward in picture-in-picture, simply modify the value of PIP_TIME_SHIFT_INTERVAL in PictureInPictureHelper.

6. 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
 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
  // 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

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**.

```
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, Co
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, Co
 // Step 2. Call the method for setting the dynamic watermark feature
mSuperPlayerView.setDynamicWatermarkConfig(dynamicWaterConfig);
```

```
public DynamicWaterConfig(String dynamicWatermarkTip, int tipTextSize, int tipTextC
```

API parameters:

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

8. Video download

Video download allows users to cache online videos and watch them offline. The cached video can be played back only in the client but cannot be actually downloaded to the device. This feature can effectively prevent downloaded videos from being distributed without authorization and protect the video security.
You can try out this feature in full screen mode in TCToolkit App > Player > Player Components > Offline Cache.

`DownloadMenuListView` (cache selection list view) is used to select and download videos at different definitions. After selecting the definition in the top-left corner, click the option of the video to be downloaded. When a check mark appears, the download has started. After clicking the **video download list** button below, you will be redirected to the Activity where `VideoDownloadListView` is located.

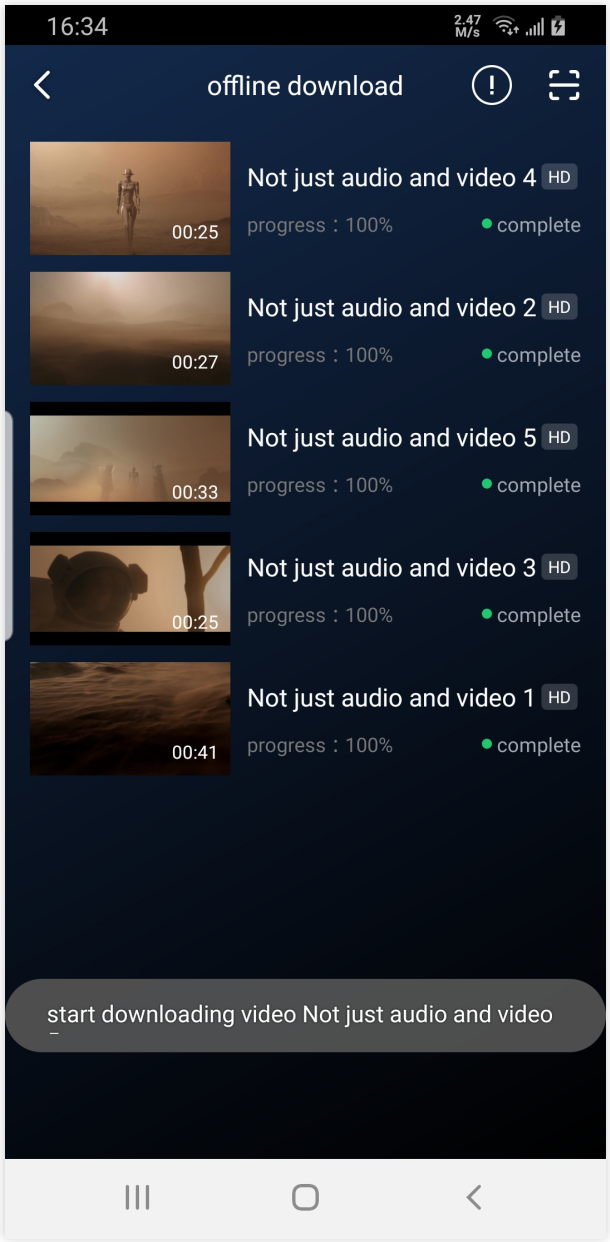
```
// Step 1. Initialize the download data with the following parameters
DownloadMenuListView mDownloadMenuView = findViewById(R.id.superplayer_cml_cache_me
mDownloadMenuView.initDownloadData(superPlayerModelList, mVideoQualityList, mDefaul

// Step 2. Set the options of the video being played back
mDownloadMenuView.setCurrentPlayVideo(mSuperplayerModel);

// Step 3. Set the click event of the **video download list** button
mDownloadMenuView.setOnCacheListClick(new OnClickListener() {
     @Override
     public void onClick(View v) {
       // Redirect to the Activity where `VideoDownloadListView` is located
       startActivity(DownloadMeduListActivity.this,VideoDownloadListActivity.class)
     }
});

// Step 4. Display the view with animation
mDownloadMenuView.show();
```

```
public void initDownloadData(List<SuperPlayerModel> superPlayerModelList,
                             List<VideoQuality> qualityList,
                             VideoQuality currentQuality,
                             String userName)
```

API parameters:

Parameter	Type	Description
superPlayerModelList	List<SuperPlayerModel>	The downloaded video data
qualityList	List<VideoQuality>	The video definition data
currentQuality	VideoQuality	The current video definition
userName	String	The username

`VideoDownloadListView` (video download list) displays the list of views of all the videos that are being downloaded and have been downloaded. When this button is clicked, if the download is in progress, it will be paused; if it is paused, it will be resumed; if it has completed, the video will be played back.




```
// Step 1. Bind the control
VideoDownloadListView mVideoDownloadListView = findViewById(R.id.video_download_lis

//Step 2. Add data
mVideoDownloadListView.addCacheVideo(mDataList, true);
```

API parameters:

```
public void addCacheVideo(List<TXVodDownloadMediaInfo> mediaInfoList, boolean
isNeedClean);
```

Parameter	Type	Description

mediaInfoList	List<TXVodDownloadMediaInfo>	The type of the added video data
isNeedClean	boolean	Whether to clear the previous data

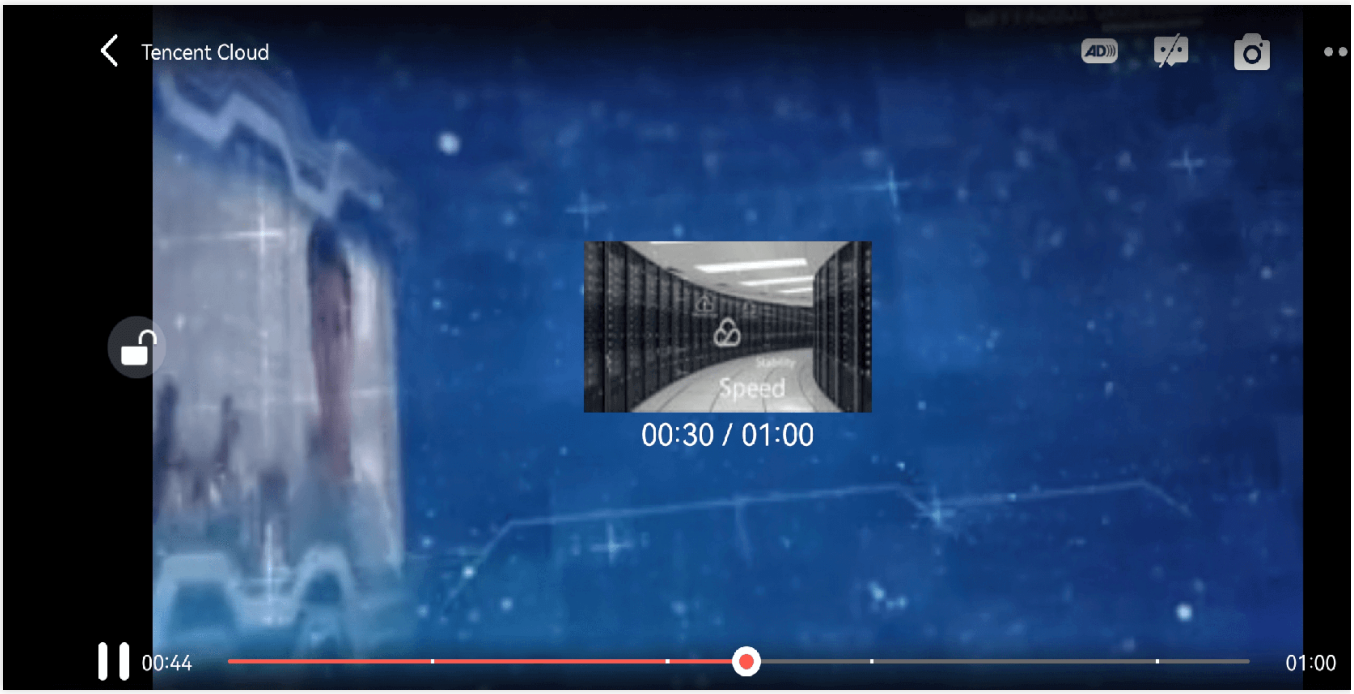
9. Image sprite and timestamp information

Timestamp information

You can add text descriptions at key positions on the progress bar, which the user can click to view and quickly understand the video information at the current position. After clicking the video information, the user can seek to the desired position.
You can try out this feature in full screen mode in TCToolkit App > Player > Player Components > Tencent Cloud Video.

Image sprite

Users can view video thumbnails when dragging or seeking on the progress bar so as to quickly understand the video content at the specified position. The thumbnail preview is implemented based on the video's image sprite. You can generate the image sprite of a video file in the VOD console or directly generate an image sprite file.
You can try out this feature in full screen mode in TCToolkit App > Player > Player Components > Tencent Cloud Video.




```
// Step 1. Get the image sprite and timestamp information in the `onPlayEvent` call
mSuperplayerView.play(superplayerModel);

// Step 2. Get timestamp and image sprite information in the `VOD_PLAY_EVT_GET_PLAY
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
    switch (event) {
        case TXVodConstants.VOD_PLAY_EVT_GET_PLAYINFO_SUCC:

            // Get the list of image URLs of the image sprite
            playImageSpriteInfo.imageUrls = param.getStringArrayList(TXVodConstants
            // Get the download URL of the image sprite WebVTT file
            playImageSpriteInfo.webVttUrl = param.getString(TXVodConstants.EVT_IMAG
            // Get the timestamp information
           ArrayList<String> keyFrameContentList =
                    param.getStringArrayList(TXVodConstants.EVT_KEY_FRAME_CONTENT_L
            // Get the time information of the timestamp information
            float[] keyFrameTimeArray = param.getFloatArray(TXVodConstants.EVT_KEY_

            // Construct the list of timestamp information
            if (keyFrameContentList != null && keyFrameTimeArray != null
                    && keyFrameContentList.size() == keyFrameTimeArray.length) {
                for (int i = 0; i < keyFrameContentList.size(); i++) {
                    PlayKeyFrameDescInfo frameDescInfo = new PlayKeyFrameDescInfo()
                    frameDescInfo.content = keyFrameContentList.get(i);
                    frameDescInfo.time = keyFrameTimeArray[i];
                    mKeyFrameDescInfoList.add(frameDescInfo);
                }
```

```
            }
            break;
        default:
            break;
    }
}

// Step 3. Assign the obtained timestamp information and image sprite to the corres
// The view of the image sprite corresponds to `mIvThumbnail` in the `VideoProgress
// The view of the timestamp information corresponds to `TCPointView` in the `Point
updateVideoImageSpriteAndKeyFrame(playImageSpriteInfo,keyFrameDescInfoList);
```

10. External subtitles

Note:

External subtitles depend on the premium version SDK of the media player, and the SDK needs to be version 11.3 or above to support it.

Currently, only SRT and VTT subtitle formats are supported. The usage is as follows:

Step 1: Add external subtitles.

Pass the external subtitle category field to `SuperPlayerModel#subtitleSourceModelList` .

```
// Pass in subtitle url, subtitle name, and subtitle type
SubtitleSourceModel subtitleSourceModel = new SubtitleSourceModel();
subtitleSourceModel.name = "ex-cn-srt";
subtitleSourceModel.url = "https://mediacloud-76607.gzc.vod.tencent-cloud.com/DemoR
subtitleSourceModel.mimeType = TXVodConstants.VOD_PLAY_MIMETYPE_TEXT_SRT;
model.subtitleSourceModelList.add(subtitleSourceModel);
```

```
// Play
mSuperPlayerView.playWithModelNeedLicence(model);
```

Step 2: Switch subtitles after playback.

```
// After the video starts playing, select the added external subtitles.
public void onClickSubTitleItem(TXTrackInfo clickInfo) {
    List<TXTrackInfo> subtitleTrackInfoList = mVodPlayer.getSubtitleTrackInfo();
    for (TXTrackInfo trackInfo : subtitleTrackInfoList) {
        if (trackInfo.trackIndex == clickInfo.trackIndex) {
            // Select the subtitle
            mVodPlayer.selectTrack(trackInfo.trackIndex);
            mSelectedSubtitleTrackInfo = trackInfo;
        } else {
            // Deselect other subtitles if they are not needed.
            mVodPlayer.deselectTrack(trackInfo.trackIndex);
        }
    }
}
```

Step 3: Configure subtitle styles.

Subtitle styles can be configured before or during playback.

```
TXSubtitleRenderModel model = new TXSubtitleRenderModel();
model.canvasWidth = 1920;  // Subtitle render canvas width
model.canvasHeight = 1080;  // Subtitle render canvas height
model.fontColor = 0xFFFFFFFF; // Set the subtitle font color, default white and opa
model.isBondFontStyle = false;  // Set whether the subtitle font is bold
mVodPlayer.setSubtitleStyle(model);
```

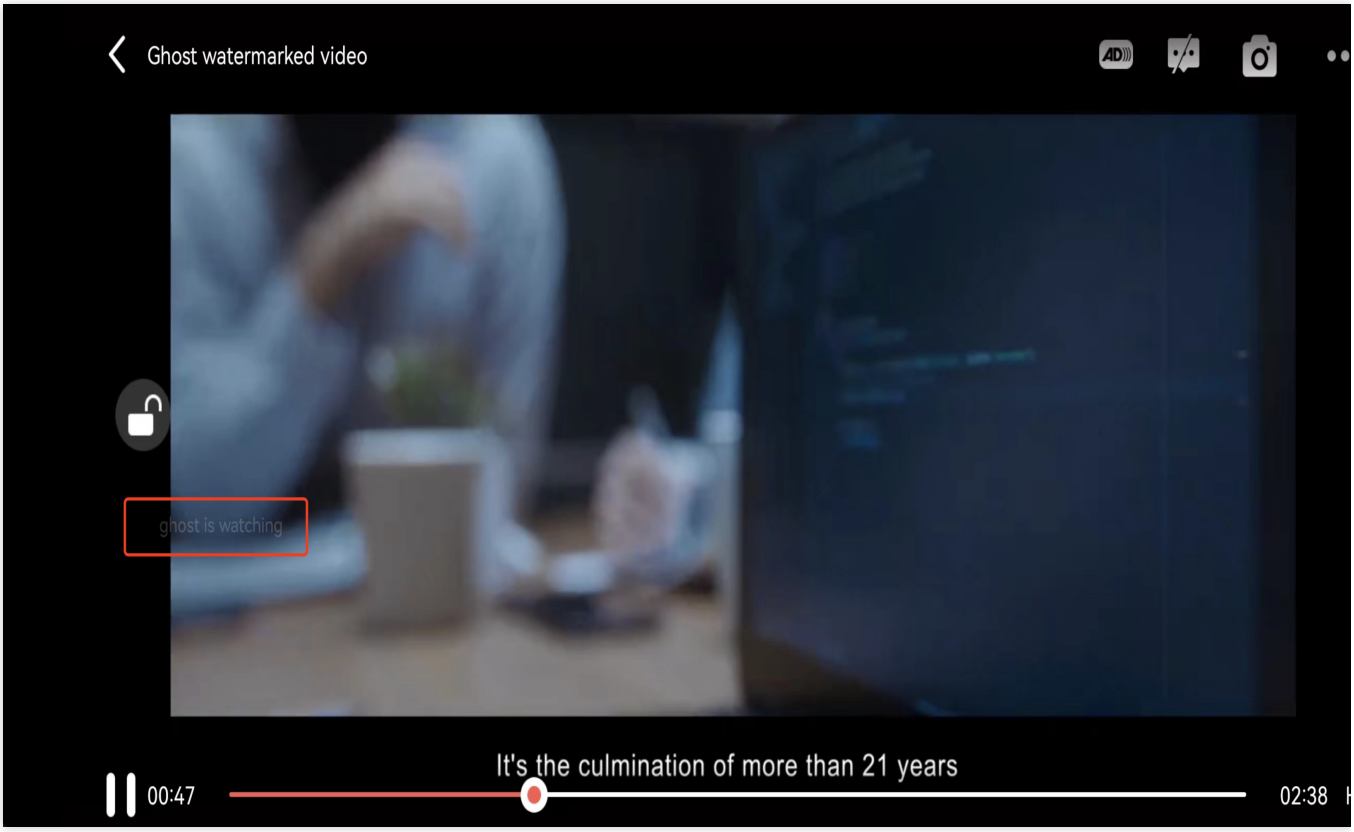
111. Ghost watermark

The content of the ghost watermark is filled in the player signature, and is finally displayed on the player through the cloud on-demand backend. The entire transmission link process is coordinated by the cloud and the player to ensure the security of the watermark. Tutorial on configuring ghost watermark in player signature. The ghost watermark only appears on the video for a short period of time, and this flash has little impact on the viewing of the video. The position of the watermark on the screen is not fixed each time, preventing others from trying to block the watermark. The effect is as shown in the picture below. When the video starts playing, the watermark will appear once and then disappear. Wait until the next time it appears before disappearing.

The content of the ghost watermark is obtained through param.getString(TXVodConstants.EVT_KEY_WATER_MARK_TEXT) after receiving the

TXVodConstants#VOD_PLAY_EVT_GET_PLAYINFO_SUCC event from the player.

Note: Supported from player version 11.6.




```
// Step 1: Configure FileId that supports ghost watermark to play video
SuperPlayerModel model = new SuperPlayerModel();
model.appId = 1500006438;
model.videoId.fileId = "387702307847129127";
model.videoId.pSign = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTUwMDA"
                    + "wNjQzOCwiZmlsZUlkIjoiMzg3NzAyMzA3ODQ3MTI5MTI3IiwiY29udG"
                    + "VudEluZm8iOnsiYXVkaW9WaWRlb1R5cGUiOiJSYXdBZGFwdGl2ZSIsIn"
                    + "Jhd0FkYXB0aXZlRGVmaW5pdGlvbiI6MTB9LCJjdXJyZW50VGltZVN0YW1w"
                    + "IjoxNjg2ODgzMzYwLCJnaG9zdFdhdGVybWFya0luZm8iOnsidGV4dCI6I"
                    + "mdob3N0IGlzIHdhdGNoaW5nIn19.0G2o4P5xVZ7zF"
                    + "lFUgBLntfX03iGxK9ntD_AONClUUno";
mSuperPlayerView.playWithModelNeedLicence(model);

// Step 2: After receiving the ghost watermark content callback in SuperPlayerView#
public void onRcvWaterMark(String text, long duration) {
    if (!TextUtils.isEmpty(text)) {
        DynamicWaterConfig dynamicWaterConfig = new DynamicWaterConfig(text, 30, Co
        dynamicWaterConfig.durationInSecond = duration;
        dynamicWaterConfig.setShowType(DynamicWaterConfig.GHOST_RUNNING);
        setDynamicWatermarkConfig(dynamicWaterConfig);
```

```
        }
    }
```

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. 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.

Flutter Integration Guide

Last updated：2025-04-01 15:11:25

SDK Download

The Tencent Cloud Player SDK for Flutter can be downloaded here.

Intended Audience

This document describes some of the proprietary capabilities of Tencent Cloud. Make sure that you have activated the relevant Tencent Cloud services before reading this document. If you don't have an account yet, sign up for free trial first.

This Document Describes

How to integrate the Tencent Cloud Player SDK for Flutter.
How to use the Player component for VOD playback.

Player Component Overview

The Player component for Flutter is an extension of the Player SDK for Flutter. Compared with the VOD player, the Player component is easier to use and integrates more features, including full screen playback, definition selection, progress bar, playback control, and thumbnails. To integrate Flutter video playback capabilities more easily, you can use the Player component for Flutter.
To integrate Flutter video playback capabilities more easily, you can use the Superplayer SDK for Flutter.
Supported features:
Full screen playback
Adaptive screen rotation during playback
Custom video thumbnail
Definition selection
Audio and brightness adjustment
Playback speed change
Hardware acceleration
Picture-in-picture (PiP) on Android and iOS

Image sprite and keyframe timestamp information

More features to come soon.

Integration Guide

1. Copy the `superplayer_widget` directory from the project to your own Flutter project.

2. Add the dependency to your project's configuration file `pubspec.yaml` .

Branch integration

Pub integration

```
superplayer_widget:
    # The path should be changed according to the location where superplayer_widget
    path: ../superplayer_widget
super_player:
  git:
    url: https://github.com/LiteAVSDK/Player_Flutter
    path: Flutter
    ref: main
# You can replace ref with the corresponding version or branch according to your ow
```

```
superplayer_widget:
    # The path should be changed according to the location where superplayer_widget
    path: ../superplayer_widget
# The pub integration defaults to the professional version.
# If there are requirements for other versions, please use the branch integration m
super_player: ^12.3.0
```

3. Modify the `superPlayer` dependency of superplayer_widget.

Enter the `pubspec.yaml` file of superplayer_widget and make the necessary modifications.

Replace the configuration with the following:

```
super_player:
  path: ../
```

Replace with:

```
super_player:
  git:
    url: https://github.com/LiteAVSDK/Player_Flutter
    path: Flutter
    ref: main
```

You can replace ref with the corresponding version or branch according to your own project needs.

4. As the player component is now integrated with internationalization, it is necessary to add the internationalization component in the entry function, as shown in the following example:

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    localizationsDelegates: [
      SuperPlayerWidgetLocals.delegate,
      // ...... your app other delegate
    ],
    supportedLocales: [
      Locale.fromSubtags(languageCode: 'en'),
      Locale.fromSubtags(languageCode: 'zh'),
      // ....... other language
    ],
    // ...... your app other code
  );
}
```

5. Import the dependency package of superplayer_widget on the pages where it is needed, as shown below:

```
import 'package:superplayer_widget/demo_superplayer_lib.dart';
```

6. For other native-related configurations, please refer to the Integration Guide.

SDK Integration

Step 1. Apply for and integrate a video playback license

Before you integrate the player, you need to sign up for a Tencent Cloud account, apply for the video playback license, and configure the license as follows (we recommend you do this when the application is launched):
If you don't configure a license, errors may occur during playback.

```
String licenceURL = ""; // The license URL obtained
String licenceKey = ""; // The license key obtained
SuperPlayerPlugin.setGlobalLicense(licenceURL, licenceKey);
```

Step 2. Set the SDK connection environment

In order to help you conduct business with higher quality and security in compliance with applicable laws and regulations in different countries and regions, Tencent Cloud provides two SDK connection environments. If you serve global users, we recommend you use the following API to configure the global connection environment.

```
SuperPlayerPlugin.setGlobalEnv("GDPR");
```

Step 3. Create a controller

```
SuperPlayerController _controller = SuperPlayerController(context);
```

Step 4. Configure the player

```
FTXVodPlayConfig config = FTXVodPlayConfig();
// If `preferredResolution` is not configured, the 720x1280 resolution stream will
config.preferredResolution = 720 * 1280;
_controller.setPlayConfig(config);
```

For detailed configuration in `FTXVodPlayConfig`, see the player configuration API of the VOD player SDK for Flutter.

Step 5. Configure event listening

```
_controller.onSimplePlayerEventBroadcast.listen((event) {
  String evtName = event["event"];
  if (evtName == SuperPlayerViewEvent.onStartFullScreenPlay) {
    setState(() {
      _isFullScreen = true;
    });
  } else if (evtName == SuperPlayerViewEvent.onStopFullScreenPlay) {
    setState(() {
      _isFullScreen = false;
    });
  } else {
    print(evtName);
  }
});
```

Step 6. Add a layout

```
Widget _getPlayArea() {
    return Container(
    height: 220,
    child: SuperPlayerView(_controller),
  );
}
```

Step 7. Listen on the Back button clicking event

Add listening for the return event to ensure that the full screen mode is exited first if the player is in full screen mode when the return event is triggered, and the page will be exited only when the return event is triggered again.
If you want to directly exit the page in full screen playback mode, you don't need to implement the listening.

```
  @override
Widget build(BuildContext context) {
  return WillPopScope(
      child: Container(
        decoration: BoxDecoration(
            image: DecorationImage(
              image: AssetImage("images/ic_new_vod_bg.png"),
              fit: BoxFit.cover,
            )),
        child: Scaffold(
          backgroundColor: Colors.transparent,
          appBar: _isFullScreen
              ? null
              : AppBar(
            backgroundColor: Colors.transparent,
            title: const Text('SuperPlayer'),
          ),
          body: SafeArea(
            child: Builder(
              builder: (context) => getBody(),
            ),
          ),
        ),
      ),
      onWillPop: onWillPop);
}

Future<bool> onWillPop() async {
  return !_controller.onBackPress();
}
```

Step 8. Start the playback

Through the URL
Through `fileId`

```
SuperPlayerModel model = SuperPlayerModel();
model.videoURL = "http://1400329073.vod2.myqcloud.com/d62d88a7vodtranscq1400329073/
_controller.playWithModelNeedLicence(model);
```

```
SuperPlayerModel model = SuperPlayerModel();
model.appId = 1500005830;
model.videoId = new SuperPlayerVideoId();
model.videoId.fileId = "8602268011437356984";
// `psign` is a player signature. For more information on the signature and how to
model.videoId.pSign = "psignXXX"
_controller.playWithModelNeedLicence(model);
```

Find the target video file in Media Assets, and you can view the `FileId` below the filename.

Play back the video through the `FileId`, and the player will request the backend for the real playback URL. If the network is abnormal or the `FileId` doesn't exist, the `SuperPlayerViewEvent.onSuperPlayerError` event will be received.

Step 9. Stop the playback

Remember to call the controller termination method when stopping the playback, especially before the next call of `startVodPlay`. This can prevent memory leak and screen flashing issues, as well as ensure that playback is stopped when the page is exited.

```
  @override
void dispose() {
  // must invoke when page exit.
  _controller.releasePlayer();
  super.dispose();
}
```

Player Component APIs

1. Playing back a video

Note

Starting from v10.7, `startPlay` is replaced by `startVodPlay`, and playback will succeed only after you use `{@link SuperPlayerPlugin#setGlobalLicense}` to set the license; otherwise, playback will fail (black screen occurs). The license needs to be set only once globally. You can use the license for CSS, UGSV, or video playback. If you have no such licenses, you can quickly apply for a trial license.

Description

This API is used to start video playback.

API

```
_controller.playWithModelNeedLicence(model);
```

Parameter description

1. SuperPlayerModel

Parameter	Type	Description
appId	int	The application's `appId`, which is required for playback via `fileId`.
videoURL	String	The video URL, which is required for playback via URL.
multiVideoURLs	List<String>	Multi-bitrate playback URLs, which are required for playback via multi-bitrate URLs.
defaultPlayIndex	int	The default playback bitrate number, which is used together with `multiVideoURLs`.
videoId	SuperPlayerVideoId	The `fileId` storage object, which is further described below.
title	String	The video title. You can use this to customize the title and overwrite the title internally requested by the player from the server.
coverUrl	String	The thumbnail image pulled from the Tencent server, whose value will be assigned automatically in `SuperVodDataLoader`.
customeCoverUrl	String	A custom video thumbnail. This parameter is used preferentially and is used to customize the video thumbnail.
duration	int	The video duration in seconds.
videoDescription	String	The video description.
videoMoreDescription	String	The detailed video description.
playAction	int	Valid values: PLAY_ACTION_AUTO_PLAY, PLAY_ACTION_MANUAL_PLAY, PLAY_ACTION_PRELOAD, as described below.

2. SuperPlayerVideoId

Parameter	Type	Description
fileId	String	The file ID, which is required.
psign	String	The player signature. For more information on the signature and how to generate it, see Player Signature.

3. playAction

PLAY_ACTION_AUTO_PLAY: The video will be automatically played back after `playWithModel` is called.

PLAY_ACTION_MANUAL_PLAY: The video needs to be played back manually after `playWithModel` is called. The player doesn't load the video and only displays the thumbnail image, which consumes no video playback resources compared with `PLAY_ACTION_PRELOAD` .

PLAY_ACTION_PRELOAD: The player will display the thumbnail image and won't start the video playback after `playWithModel` is called, but the video will be loaded. This can start the playback faster than `PLAY_ACTION_MANUAL_PLAY` .

2. Playback pause

Description

This API is used to pause video playback.

API

```
_controller.pause();
```

3. Resuming playback

Description

This API is used to resume the playback.

API

```
_controller.resume();
```

4. Restarting playback

Description

This API is used to restart the video playback.

API

```
_controller.reStart();
```

5. Resetting the player

Description

This API is used to reset the player status and stop the video playback.

API

```
_controller.resetPlayer();
```

6. Releasing the player

Description

This API is used to release the player resources and stop the video playback. After it is called, the controller can no longer be reused.

API

```
_controller.releasePlayer();
```

7. Processing the player Back button event

Description

This API is used to determine the action to perform when the Back button is clicked in full screen playback mode. If `true` is returned, the full screen mode is exited, and the Back button clicking event is consumed; if `false` is returned, the event is unconsumed.

API

```
_controller.onBackPress();
```

8. Switching the definition

Description

This API is used to switch the definition of the video being played back in real time.

API

```
_controller.switchStream(videoQuality);
```

Parameter description

`videoQuality` can generally be obtained through `_controller.currentQualiyList` (definition list) and `_controller.currentQuality` (default definition) after the playback starts. **The definition selection capabilities have been integrated into the player. You can click the definition in the bottom-right corner to switch the definition in full screen mode.**

Parameter	Type	Description
index	int	The definition number.
bitrate	int	Bitrate for the definition.
width	int	The video width for the definition.
height	int	The video height for the definition.
name	String	The definition abbreviation.

title	String	Displayed definition name.
url	String	The multi-bitrate URL, which is optional.

9. Adjusting the playback progress (seek)

Description

This API is used to adjust the current video playback progress.

API

```
_controller.seek(progress);
```

Parameter description

Parameter	Type	Description
progress	double	Target time in seconds.

10. Configuring the Player component

Description

This API is used to configure Superplayer.

API

```
_controller.setPlayConfig(config);
```

Parameter description

Parameter	Type	Description
connectRetryCount	int	The number of player reconnections. If the SDK is disconnected from the server due to an exception, the SDK will attempt to reconnect to the server.
connectRetryInterval	int	The interval between two player reconnections. If the SDK is disconnected from the server due to an exception, the SDK will attempt to reconnect to the server.
timeout	int	Player connection timeout period
playerType	int	Player type. Valid values: 0: VOD; 1: live streaming; 2: live stream replay.
headers	Map	Custom HTTP headers
enableAccurateSeek	bool	Whether to enable accurate seek. Default value: true.
autoRotate	bool	If it is set to `true`, the MP4 file will be automatically rotated according

		to the rotation angle set in the file, which can be obtained from the `PLAY_EVT_CHANGE_ROTATION` event. Default value: `true`.
smoothSwitchBitrate	bool	Whether to enable smooth multi-bitrate HLS stream switching. If it is set to `false` (default), multi-bitrate URLs are opened faster. If it is set to `true`, the bitrate can be switched smoothly when IDR frames are aligned.
cacheMp4ExtName	String	The cached MP4 filename extension. Default value: `mp4`.
progressInterval	int	Progress callback interval in ms. If it is not set, the SDK will call back the progress once every 0.5 seconds.
maxBufferSize	int	The maximum size of playback buffer in MB. The setting will affect `playableDuration`. The greater the value, the more the data that is buffered in advance.
maxPreloadSize	int	Maximum preload buffer size in MB
firstStartPlayBufferTime	int	Duration of the video data that needs to be loaded during the first buffering in ms. Default value: 100 ms
nextStartPlayBufferTime	int	The minimum buffered data size to stop buffering (secondary buffering for insufficient buffered data or progress bar drag buffering caused by `seek`) in milliseconds. Default value: 250 ms
overlayKey	String	The HLS security enhancement encryption and decryption key.
overlayIv	String	The HLS security enhancement encryption and decryption IV.
extInfoMap	Map	Some special configuration items.
enableRenderProcess	bool	Whether to allow the postrendering and postproduction feature, which is enabled by default. If the super-resolution plugin exists after it is enabled, the plugin will be loaded by default.
preferredResolution	int	Resolution of the video used for playback preferably. `preferredResolution` = `width` * `height`

11. Enabling/Disabling hardware decoding

Description

This API is used to enable/disable playback based on hardware decoding.

API

```
_controller.enableHardwareDecode(enable);
```

12. Getting the playback status

Description

This API is used to get the playback status.

API

```
SuperPlayerState superPlayerState = _controller.getPlayerState();
```

Parameter description

Parameter	Type	Description
INIT	SuperPlayerState	Initial status
PLAYING	SuperPlayerState	Playing back
PAUSE	SuperPlayerState	Paused
LOADING	SuperPlayerState	Loading
END	SuperPlayerState	Ended

13. Entering the PiP mode

Description

After the method is invoked, the video will enter Picture-in-Picture mode, which only supports Android 7.0 and above, as well as models that support Picture-in-Picture mode. Among them, iOS live streaming Picture-in-Picture requires premium permission and uses SDK version 12.1 or above.

API

```
_controller.enterPictureInPictureMode(
backIcon: "images/ic_pip_play_replay.png",
playIcon: "images/ic_pip_play_normal.png",
pauseIcon: "images/ic_pip_play_pause.png",
forwardIcon: "images/ic_pip_play_forward.png");
```

Parameter description

This parameter is only applicable to the Android platform. The default image is used for the iOS platform.

Parameter	Type	Description
backIcon	String	The seek backward icon, which can be up to 1 MB in size as limited by Android. And if it is not set, the system icon will be used. Passing an empty string will hide the icon.
playIcon	String	The playback icon, which can be up to 1 MB in size as limited by Android. And if it is not set, the system icon will be used. Passing an empty string will hide the icon.

pauseIcon	String	The pause icon, which can be up to 1 MB in size as limited by Android. And if it is not set, the system icon will be used. Passing an empty string will hide the icon.
forwardIcon	String	The fast forward icon, which can be up to 1 MB in size as limited by Android. And if it is not set, the system icon will be used. Passing an empty string will hide the icon.

Event Notifications

1. Listening on playback events

Description

This callback is used to listen for player operation events.

Code

```
_controller.onSimplePlayerEventBroadcast.listen((event) {
    String evtName = event["event"];
    if (evtName == SuperPlayerViewEvent.onStartFullScreenPlay) {
        setState(() {
        _ isFullScreen = true;
        });
    } else if (evtName == SuperPlayerViewEvent.onStopFullScreenPlay) {
        setState(() {
          _isFullScreen = false;
        });
    } else {
        print(evtName);
    }
});
```

Event description

Status	Description
onStartFullScreenPlay	Entered the full screen playback mode
onStopFullScreenPlay	Exited the full screen playback mode
onSuperPlayerDidStart	Playback started
onSuperPlayerDidEnd	Playback ended
onSuperPlayerError	Playback error
onSuperPlayerBackAction	Return event

Advanced Features

1. Requesting video data in advance through `fileId`

The `SuperVodDataLoader` API can be used to request the video data in advance to accelerate the playback start process.

Sample code

```
SuperPlayerModel model = SuperPlayerModel();
model.appId = 1500005830;
model.videoId = new SuperPlayerVideoId();
model.videoId.fileId = "8602268011437356984";
model.title = "VOD";
SuperVodDataLoader loader = SuperVodDataLoader();
// Values of the required parameters in `model` are directly assigned in `SuperVodD
loader.getVideoData(model, (resultModel) {
  _controller.playWithModelNeedLicence(resultModel);
})
```

2. Using the PiP mode

1. Platform configuration.

Android

IOS

In your project's Android package, find the build.gradle file and make sure that the compileSdkVersion and targetSdkVersion are version 31 or higher.

In your project's target, select **Signing & Capabilities** and add **Background Modes**, then check "**Audio, AirPlay, and Picture in Picture**".

2. Copy the sample code of `SuperPlayer` .

In `example/lib` in the GitHub project, copy the `SuperPlayer` package to the `lib` directory in your project and integrate the Player component as instructed in `demo_superplayer.dart` in the sample code.

Then, you can see the PiP mode button at the center on the right of the playback UI of the Player component and click the button to enter the PiP mode.

3. Listening on the lifecycle of the PiP mode.

You can use `onExtraEventBroadcast` in `SuperPlayerPlugin` to listen on the lifecycle of the PiP mode. As follows:

```
SuperPlayerPlugin.instance.onExtraEventBroadcast.listen((event) {
  int eventCode = event["event"];
  if (eventCode == TXVodPlayEvent.EVENT_PIP_MODE_ALREADY_EXIT) {
    // exit pip mode
  } else if (eventCode == TXVodPlayEvent.EVENT_PIP_MODE_REQUEST_START) {
```

```
      // enter pip mode
    } else if (eventCode == TXVodPlayEvent.EVENT_PIP_MODE_ALREADY_ENTER) {
      // already enter pip mode
    } else if (eventCode == TXVodPlayEvent.EVENT_IOS_PIP_MODE_WILL_EXIT) {
      // will exit pip mode
    } else if (eventCode == TXVodPlayEvent.EVENT_IOS_PIP_MODE_RESTORE_UI) {
      // restore UI only support iOS
    }
  });
```

4. Error codes for entering the PiP mode

If the user fails to enter the PiP mode, the failure will not only be logged but also be prompted through a toast. You can modify the error handling operations in the `_onEnterPipMode` method in `superplayer_widget.dart`. The error codes are as detailed below:

Parameter	Code	Description
NO_ERROR	0	Started successfully with no errors.
ERROR_PIP_LOWER_VERSION	-101	The Android version is too early and doesn't support the PiP mode.
ERROR_PIP_DENIED_PERMISSION	-102	The PiP mode permission wasn't enabled, or the current device doesn't support PiP.
ERROR_PIP_ACTIVITY_DESTROYED	-103	The current UI was terminated.
ERROR_IOS_PIP_DEVICE_NOT_SUPPORT	-104	The device model or system version doesn't support PiP (only supported on iPadOS 9+ and iOS 14+).
ERROR_IOS_PIP_PLAYER_NOT_SUPPORT	-105	The player doesn't support PiP.
ERROR_IOS_PIP_VIDEO_NOT_SUPPORT	-106	The video doesn't support PiP.
ERROR_IOS_PIP_IS_NOT_POSSIBLE	-107	The PiP controller was unavailable.
ERROR_IOS_PIP_FROM_SYSTEM	-108	The PiP controller reported an error.
ERROR_IOS_PIP_PLAYER_NOT_EXIST	-109	The player object didn't exist.
ERROR_IOS_PIP_IS_RUNNING	-110	The PiP feature was running.
ERROR_IOS_PIP_NOT_RUNNING	-111	The PiP feature didn't start.

5. Checking whether the current device supports PiP

You can use `isDeviceSupportPip` in `SuperPlayerPlugin` to check whether the PiP mode can be enabled as follows:

```
int result = await SuperPlayerPlugin.isDeviceSupportPip();
if(result == TXVodPlayEvent.NO_ERROR) {
  // pip support
}
```

The returned result in `result` means the same as the error code of the PiP mode.

6. Use picture-in-picture controllers to manage picture-in-picture

The picture-in-picture controller TXPipController is a tool for picture-in-picture encapsulated in the superplayer_widget, **and must be used in conjunction with SuperPlayerView**.

When entering picture-in-picture, the current interface will be automatically closed and the listener method set in advance will be called. In the callback method, you can save the necessary parameters of the current interface. After restoring from picture-in-picture, the previous interface will be pushed back and the previously saved parameters will be passed.

When using this controller, only one instance of picture-in-picture and the player can exist. When re-entering the player interface, picture-in-picture will be automatically closed.

6.1 At the entry point of your project, such as main.dart, call TXPipController to set up picture-in-picture control and jump to the player page for entering picture-in-picture.

You can set different interfaces according to your project. The code example is as follows:

```
TXPipController.instance.setNavigatorHandle((params) {
  navigatorKey.currentState?.push(MaterialPageRoute(builder: (_) => DemoSuperPlayer
});
```

6.2 To set up the listener for the picture-in-picture playback page, you need to implement the `TXPipPlayerRestorePage` method. After setting it up, when you are about to enter picture-in-picture, the controller will call the `void onNeedSavePipPageState(Map<String, dynamic> params)` method. At this time, you can store the necessary parameters for the current page in the params.

```
TXPipController.instance.setPipPlayerPage(this);
```

Later, when the user clicks on the enter picture-in-picture button on the SuperPlayerView, the `_onEnterPipMode` internal method of `SuperPlayerView` will be called to enter picture-in-picture, or you can call the `enterPictureInPictureMode` method of `SuperPlayerController` to enter it manually.

3、Video download

donwload Video

1. To use the video download feature of the player component, you first need to enable `isEnableDownload` in SuperPlayerModel, which is disabled by default.

```
SuperPlayerModel model = SuperPlayerModel();
// Enable video download capability
model.isEnableDownload = true;
```

The player component currently only enables downloading in VOD playback mode.

2. You can use the `startDownload` method of `SuperPlayerController` to directly download the video that the player is currently playing, corresponding to the clarity of the currently playing video. You can also use `DownloadHelper` to download a specified video, as follows:

```
DownloadHelper.instance.startDownloadBySize(videoModel, videoWidth, videoHeight);
```

You can use `startDownloadBySize` of `DownloadHelper` to download videos of a specific resolution. If the resolution is not available, a video with a similar resolution will be downloaded.

In addition to the above interfaces, you can also choose to pass in the quality ID or mediaInfo to download directly.

```
// QUALITY_240P  240p
// QUALITY_360P  360P
// QUALITY_480P  480p
// QUALITY_540P  540p
// QUALITY_720P  720p
// QUALITY_1080P 1080p
// QUALITY_2K    2k
// QUALITY_4K    4k
// The quality parameter can be customized to take the minimum value of the resolut
// (for example, for a resolution of 1280*720, if you want to download a stream of
// you can pass in QUALITY_720P for the quality parameter). The player SDK will sel
// a resolution less than or equal to the passed-in resolution for downloading.
// Using quality ID to download
DownloadHelper.instance.startDownload(videoModel, qualityId);
// Using mediaInfo to download
DownloadHelper.instance.startDownloadOrg(mediaInfo);
```

3. Quality ID conversion

For VOD, `CommonUtils` provides the `getDownloadQualityBySize` method to convert the resolution to the corresponding quality ID.

```
CommonUtils.getDownloadQualityBySize(width, height);
```

Stop downloading video

You can use `stopDownload` of `DownloadHelper` method to stop downloading the corresponding video.

Example code:

```
DownloadHelper.instance.stopDownload(mediaInfo);
```

The mediaInfo can be obtained through `getMediaInfoByCurrent` of `DownloadHelper` method or by using `getDownloadList` of `TXVodDownloadController` to obtain download information.

Delete downloaded video

You can use `deleteDownload` of `DownloadHelper` method to delete the corresponding video

```
bool deleteResult = await DownloadHelper.instance.deleteDownload(downloadModel.medi
```

The deleteDownload method will return the result of the deletion, so you can determine whether the deletion was successful.

Download status

DownloadHelper provides the basic isDownloaded method to determine whether a video has been downloaded. You can also register a listener to determine the download status in real time.
DownloadHelper distributes download events, and you can register for events using the following code.

```
// Registering download event listeners
DownloadHelper.instance.addDownloadListener(FTXDownloadListener((event, info) {
      // Download status change
    }, (errorCode, errorMsg, info) {
      // Download error callback
    }));
// Removing download event listeners
DownloadHelper.instance.removeDownloadListener(listener);
```

In addition, you can also use the `TXVodDownloadController.instance.getDownloadInfo(mediaInfo)` method or the `TXVodDownloadController.instance.getDownloadList()` method to directly query the downloadState in the mediaInfo to determine the download status.

Play downloaded videos

You can use the playPath field in the video information obtained from `TXVodDownloadController.instance.getDownloadInfo(mediaInfo)` and `TXVodDownloadController.instance.getDownloadList()` to play the downloaded videos directly with TXVodPlayerController.

```
controller.startVodPlay(mediaInfo.playPath);
```

4. The usage of screen orientation

Screen orientation switching configuration

To enable screen orientation switching for the player component, you need to open the project configuration in Xcode on iOS. Under the "Deployment" tab in the "General" section, select " `Landscape left` " and " `Landscape right` ". This will ensure that the iOS device can support landscape orientation.
If you want other pages of your app to remain in portrait mode and not be affected by automatic screen rotation, you need to configure the portrait mode at the entry point of your project. The code is as follows:

```
SystemChrome.setPreferredOrientations([DeviceOrientation.portraitUp]);
```

Automatically switch to full-screen mode according to the sensor configuration

On the Android side, you need to call the following method to start listening to the sensor:

```
SuperPlayerPlugin.startVideoOrientationService();
```

After calling this method, the Android device will start listening to the sensor, and rotate events will be sent to the Flutter side via `SuperPlayerPlugin.instance.onEventBroadcast` . The player component will automatically rotate according to these events. Here is an example of how to use this listener:

```
SuperPlayerPlugin.instance.onExtraEventBroadcast.listen((event) {
    int eventCode = event["event"];
    if (eventCode == TXVodPlayEvent.EVENT_ORIENTATION_CHANGED ) {
      int orientation = event[TXVodPlayEvent.EXTRA_NAME_ORIENTATION];
      // do orientation
    }
  });
```

Frequently Asked Questions

1. After integration, playback often shows sound but no picture.
Since the UI components and the Flutter player plugin will update their calling methods with version iterations, it is necessary to keep their versions consistent. The version of the player component can be confirmed using PlayerConstants.PLAYER_WIDGET_VERSION. For the version of the Flutter player plugin, in addition to the version during integration, you can also confirm it using FPlayerPckInfo.PLAYER_VERSION to ensure that the versions of both are consistent.
2. The ability of automatic screen rotation needs to be removed.
The player component adopts an open-source approach. To meet the business customization needs of different customers, it is recommended that customers customize the business by directly modifying the component code. If you need to keep up with the component version, you can create a private repository, fork the component code, and pick the changes after updates.

Demo experience

For more features and a demo experience of debugging, please click here.When running this demo, you need to set your own player license in the demo_config, and modify the package name and bundleId to your signed package name and bundleId in the Android and iOS configurations.

Integration (No UI)
Web Integration
TCPlayer Integration Guide

Last updated：2024-04-11 16:48:34

This document introduces the Web Player SDK (TCPlayer) tailored for both VOD and live streaming. It can be quickly integrated with your own web application to enable video playback features. TCPlayer comes with a default set of UI elements, which you can use as needed.

Overview

The web player utilizes the `<video>` tag of HTML5 and Flash for video playback. It offers a uniform video playback experience across different platforms when browsers do not natively support video playback. In conjunction with Tencent's Video on Demand service, it provides hotlink protection and features for playing standard encrypted HLS videos.

Supported protocols

Audio/Video Protocol	Use	URL Format	PC Browser	Mobile Browser
MP3	Audio	http://xxx.vod.myqcloud.com/xxx.mp3	Supported	Supported
MP4	VOD playback	http://xxx.vod.myqcloud.com/xxx.mp4	Supported	Supported
HLS(M3U8)	Live stream	http://xxx.liveplay.myqcloud.com/xxx.m3u8	Supported	Supported
	VOD playback	http://xxx.vod.myqcloud.com/xxx.m3u8	Supported	Supported
FLV	Live stream	http://xxx.liveplay.myqcloud.com/xxx.flv	Supported	Partially supported
	VOD playback	http://xxx.vod.myqcloud.com/xxx.flv	Supported	Partially supported
WebRTC	Live stream	webrtc://xxx.liveplay.myqcloud.com/live/xxx	Supported	Supported

Note:

Only H.264 encoding is supported.

The player is compatible with mainstream browsers and can automatically select the optimal playback scheme depending on the browser.

In some browser environments, HLS and FLV video playback depends on Media Source Extensions.

If a browser does not support WebRTC, a WebRTC URL passed in will be converted automatically to better support playback.

Supported Features

Feature\\Browser	Chrome	Firefox	Edge	QQ Browser	Mac Safari	iOS Safari	WeChat	Android Chrome
Player dimension configuration	✓	✓	✓	✓	✓	✓	✓	✓
Resuming playback	✓	✓	✓	✓	✓	✓	✓	✓
Playback speed change	✓	✓	✓	✓	✓	✓	✓	✓
Preview thumbnails	✓	✓	✓	✓	-	-	-	-
Changing `fileID` for playback	✓	✓	✓	✓	✓	✓	✓	✓
Flipping videos	✓	✓	✓	✓	✓	✓	✓	✓
Progress bar marking	✓	✓	✓	✓	✓	-	-	-
HLS adaptive bitrate	✓	✓	✓	✓	✓	✓	✓	✓
Referer hotlink protection	✓	✓	✓	✓	✓	✓	✓	-
Definition change notifications	✓	✓	✓	✓	-	-	-	✓
Preview	✓	✓	✓	✓	✓	✓	✓	✓

Playing HLS videos encrypted using standard schemes	✓	✓	✓	✓	✓	✓	✓	✓
Playing HLS videos encrypted using private protocols	✓	✓	✓	-	-	-	Android:✓ iOS: -	✓
Video statistics	✓	✓	✓	✓	-	-	-	-
Video data monitoring	✓	✓	✓	✓	-	-	-	-
Custom UI messages	✓	✓	✓	✓	✓	✓	✓	✓
Custom UI	✓	✓	✓	✓	✓	✓	✓	✓
On-screen comments	✓	✓	✓	✓	✓	✓	✓	✓
Watermark	✓	✓	✓	✓	✓	✓	✓	✓
Ghost watermark	✓	✓	✓	✓	✓	✓	✓	✓
Playlist	✓	✓	✓	✓	✓	✓	✓	✓
Frame sync under poor network conditions	✓	✓	✓	✓	✓	✓	✓	✓

Note:

Only H.264 encoding is supported.

Chrome and Firefox for Windows and macOS are supported.

Chrome, Firefox, Edge, and QQ Browser need to load `hls.js` to play HLS.

The Referer hotlink protection feature is based on the Referer field of HTTP request headers. Some HTTP requests initiated by Android browsers do not carry the Referer field.

The player is compatible with mainstream browsers and can automatically select the optimal playback scheme depending on the browser used. For example, for modern browsers such as Chrome, the player uses the HTML5 technology for playback, and for mobile browsers, it uses the HTML5 technology or the browser's built-in capabilities.

Preparations

From version 5.0.0, the TCPlayer SDK for Web (TCPlayer) requires access to a License authorization for use. If you don't need the new premium functions, you can apply for a basic License to **try TCPlayer for free;** if you want to use the newly added premium functions you need to purchase a premium License. The detailed information is as follows:

TCPlayer feature	Feature Scope	Required License	Pricing	Authorization Unit
Basic Functions	Includes all features provided in versions prior to 5.0.0, see Product Features for details	Web Player Basic Version License	**0 CNY** Free Application	An exact domain (1 License can authorize up to 10 exact domains)
Premium Functions	Basic Version Features, VR Playback, Security Check	Web Player Premium Version License	399 CNY/month Buy Now	Wildcard Domain (1 License can authorize up to 1 wildcard domain)

Note:
1. Web Player Basic Version License can be applied for free, with a default validity of 1 year; if the remaining validity period is less than 30 days, it can be renewed for free.
2. To facilitate local development, the player won't authenticate localhost or 127.0.01; hence, these types of local service domain names need not be applied for when requesting a License.

Integration Guide

By following these steps, you can add a video player to your website.

Step 1. Import files into the page

The Player SDK supports two integration methods: CDN and NPM:

1. Integration through CDN

Create a new index.html file in your local project and import the player style and script files into the HTML page:

```
<link href="https://web.sdk.qcloud.com/player/tcplayer/release/v5.1.0/tcplayer.min
<!--Player script file-->
<script src="https://web.sdk.qcloud.com/player/tcplayer/release/v5.1.0/tcplayer.v5
```

It is recommended to deploy resources on your own when using the Player SDK, click Download player resources. Deploy the unzipped folder without altering its directory structure to prevent cross-referencing issues between resources.

If the deployment address is `aaa.xxx.ccc`, import the player style and script files at the appropriate places. When deploying on your own, you need to manually reference the dependency files under the libs folder of the resource package, otherwise, the Tencent Cloud CDN files will be requested by default.

```
<link href="aaa.xxx.ccc/tcplayer.min.css" rel="stylesheet"/>
<!--To play HLS format videos in modern browsers like Chrome and Firefox through H
<script src="aaa.xxx.ccc/libs/hls.min.x.xx.m.js"></script>
<!--Player script file-->
<script src="aaa.xxx.ccc/tcplayer.vx.x.x.min.js"></script>
```

2. Integration through npm

First, install the tcplayer npm package:

```
npm install tcplayer.js
```

Import the SDK and style files:

```
import TCPlayer from 'tcplayer.js';
import 'tcplayer.js/dist/tcplayer.min.css';
```

Step 2. Place the player container

Add the player container to the location on the page where the player is to be displayed. For example, add the following code in index.html (the container ID and dimensions can be custom defined).

```
<video id="player-container-id" width="414" height="270" preload="auto" playsinline
</video>
```

Note:

The player container must be a `<video>` tag.

In the example, `player-container-id` is the ID of the player container, which you can set yourself.

We recommend you set the size of the player container zone through CSS, which is more flexible than the attribute and can achieve effects such as fit to full screen and container adaption.

The `preload` attribute in the example specifies whether to load the video after the page is loaded. It is usually set to `auto` for faster video playback. Other options are: `meta` (to only load metadata after the page loads) and `none` (to not load the video after the page loads). Videos will not automatically load on mobile devices due to system restrictions.

The attributes `playsinline` and `webkit-playsinline` are used to achieve inline playback in standard mobile browsers without hijacking video playback. This is just an example, please use as needed.

Setting the `x5-playsinline` attribute in the TBS kernel will utilize the X5 UI player.

Step 3. Initialize the player

After page initialization, you can play video resources. The player supports both video on demand (VOD) and live streaming playback scenarios as follows:

VOD playback: The player can play Tencent Video on Demand media resources through FileID. For the specific VOD process, please refer to the Using the Player for Playback document.

Live playback: The player can pull a live audio/video stream for playback by passing in a URL. For information on generating a Tencent Cloud Streaming Services URL, see Splicing Live Streaming URLs.

URL playback (VOD and live)

File ID playback (VOD)

After page initialization, call the method in the player instance to pass in the URL to the method.

```
// `player-container-id` is the player container ID, which must be the same as in H
var player = TCPlayer('player-container-id', {
    sources: [{
      src: 'Please replace with your playback URL',
    }],
    licenseUrl: 'Please replace with your licenseUrl', // License URL, see the prep
    language: 'Please replace with your Setting language', // Setting language en |
});

// player.src(url); // URL playback address
```

In the initialization code on the index.html page, add the following initialization script. Pass in the video ID and appID obtained during the preparation phase from the fileID (in Media Management) and (peek in **Account Information** > Basic Information).

```
var player = TCPlayer('player-container-id', { // player-container-id is the player
    fileID: 'Please enter your fileID', // Enter the fileID of the video to be play
    appID: 'Please enter your appID', // Enter the appID of your VOD account
    // Enter the player Signature psign, for information on the Signature and how t
    psign: 'Please enter your player Signature psign',
    licenseUrl: 'Please enter your licenseUrl', // Refer to the preparation section
    language: 'Please replace with your Setting language', // Setting language en |
});
```

Note:

Not all videos can be played successfully in a browser. We recommend you use Tencent Cloud's services to transcode a video before playing it.

Step 4. Implement more features

You can utilize the server-side capabilities of Video on Demand (VOD) for advanced features, such as automatic switching of adaptive streams, video thumbnail previews, and adding video marker information. These features are detailed in Play back a long video, which you can refer to for implementation.
Additionally, the player offers more features. For a list of features and instructions on how to use them, please see the Feature Demonstration page.

TCPlayer Resolution Configuration Guide

Last updated：2024-04-11 16:50:03

During playback, you can switch the video resolution automatically or manually to accommodate different sizes of playback devices and network conditions, thus enhancing the viewing experience. This article will explain several scenarios.

Live Streaming

Live streaming videos are played in the form of URLs. When initializing the player, you can specify the URL to be played through the sources field. Alternatively, after initializing the player, you can play by calling the src method on the player instance.

1. Adaptive Bitrate (ABR)

Adaptive bitrate URLs can seamlessly transition during switches without causing interruptions or jumps, ensuring a smooth transition in both visual and auditory experiences. This technology is also relatively simple to use; you just need to pass the playback URL to the player, which will automatically parse the sub-streams and render the resolution switching component on the control bar.

Example 1: Playing HLS adaptive bitrate URLs

During the player's initialization, when an adaptive bitrate URL is passed in, the player will automatically generate a resolution switching component and switch automatically based on network conditions.

```
const player = TCPlayer('player-container-id', { // player-container-id is the play
  sources: [{
    src: 'https://hls-abr-url', // hls adaptive bitrate URL
  }],
});
```

Note:

Parsing the substreams of an HLS adaptive bitrate requires the dependency on the MSE API of the playback environment. In browsers not supporting MSE (e.g., Safari on iOS), the browser internally handles this by automatically switching resolutions based on network conditions, but it won't be able to parse multiple resolutions for manual switching.

Example 2: Playing WebRTC adaptive bitrate URLs

In the WebRTC adaptive bitrate scenario, after the address is input, the player will automatically decompose the substream addresses based on the ABR template in the address.

```
const player = TCPlayer('player-container-id',{
  sources: [{
    src: 'webrtc://global-lebtest-play.myqcloud.com/live/lebtest?txSecret=f22a813b2
  }],

  webrtcConfig: {
    // Whether to render multiple resolutions switch, enabled by default, optional
    enableAbr: true,
    // The label name corresponding to the template name, optional
    abrLabels: {
      d1080p: 'FHD',
      d540p: 'HD',
      d360p: 'SD',
      auto: 'AUTO',
    },
  },
});
```

The following explanations are provided for the parameters in the WebRTC URL:

1. tabr_bitrates specifies the ABR template, and the number of templates will determine the number of rendered resolutions. If no separate resolution label is set, the template name (e.g., d1080p) will be used as the name of the resolution.

2. tabr_start_bitrate specifies the initial resolution setting.

3. tabr_control sets whether automatic resolution switching is enabled. Once enabled, the player will render an option for automatic resolution.

2. Manually setting the resolution

If the playback URL is not an adaptive bitrate URL, you can also manually set the resolution. See the following code:

```
const player = TCPlayer('player-container-id', { // player-container-id is the play
  multiResolution:{
    //  Configure multiple resolution URLs
    sources:{
      'SD':[{
        src: 'http://video-sd-url',
      }],
      'HD':[{
        src: 'http://video-hd-url',
      }],
      'FHD':[{
        src: 'http://video-fhd-url',
      }]
    },
    //  Configure the tag for each resolution
```

```
    labels:{
      'SD':'Standard Definition','HD':'High Definition','FHD':'Full High Definition
    },
    //  Configure the order of resolutions in the player component
    showOrder:['SD','HD','FHD'],
    //  Configure the default selected resolution
    defaultRes: 'SD',
  },
});
```

VOD Scenario

In the VOD scenario, if you play via fileID, which type of file to play (source file, transcoded file, adaptive bitrate file) and the resolution of substreams of the adaptive bitrate file are all set in the player signature. You can refer to the guide Play back an adaptive bitrate streaming video to understand the entire process of playing videos in the VOD scenario.

When calculating the player signature, you can set the display names of substreams of different resolutions through the resolutionNames in the contentInfo field. If you leave it blank or fill in an empty array, the default configuration is used.

```
resolutionNames: [{
  MinEdgeLength: 240,
  Name: '240P',
}, {
  MinEdgeLength: 480,
  Name: '480P',
}, {
  MinEdgeLength: 720,
  Name: '720P',
}, {
  MinEdgeLength: 1080,
  Name: '1080P',
}, {
  MinEdgeLength: 1440,
  Name: '2K',
}, {
  MinEdgeLength: 2160,
  Name: '4K',
}, {
  MinEdgeLength: 4320,
  Name: '8K',
}]
```

The number of substreams during playback depends on the number of substreams converted according to different adaptive bitrate templates during transcoding. These substreams will fall within the MinEdgeLength range set by resolutionNames based on short side length and then be displayed with the corresponding Name as the clarity name.
If you need to quickly experience generating the player signature, you can use the Tencent Video on Demand console's Player Signature Generation Tool.

TCPlayer Swift Live Streaming Downgrade Notice

Last updated：2024-04-11 16:50:14

Downgrade scenarios

Live Event Broadcasting is based on WebRTC and depends on the operating system and browser support for WebRTC.
Currently, the SDK has been tested on the following operating systems and browsers, with the test results as follows:

Operating system	OS Version	Browser	Browser version	Support for stream pull
Windows	win 10	Chrome	86+	✓
		Firefox	88+	✓
		Microsoft Edge	86+	✓
macOS	10.5+	Safari	13.1+	✓
		Chrome	86+	✓
		Firefox	88+	✓
		Microsoft Edge	86+	✓
iOS	13.1.1+	Safari	13.7+	✓
		Chrome	86+	✓
		Firefox	33+	✓
		Microsoft Edge	89	✓
		WeChat embedded	-	✓
Android	-	Chrome	86+	✓
		Firefox	88+	✓
		WeChat embedded	X5 core	✓

		WeChat embedded	XWeb core	✓

Additionally, in some browsers that support WebRTC, there may be decoding failures or server-side issues. In these cases, the player will convert the WebRTC URL to a more compatible HLS URL for playback. This behavior is known as downgrade processing.

To summarize, there are several scenarios that trigger downgrading:

The browser environment does not support WebRTC.

Failed to connect to the server, and the number of retries has exceeded the set value (internal status code -2004).

Decoding failure during playback (internal status code -2005).

Other WebRTC-related errors (internal status code -2001).

Downgrade method

1. Automatic downgrade

During player initialization, the Live Event Broadcasting address is passed through the sources field. In environments requiring downgrade processing, the player automatically converts the protocol, converting the Live Event Broadcasting address to an HLS protocol address.

For example, Live Event Broadcasting address:

```
webrtc://global-lebtest-play.myqcloud.com/live/lebtest?txSecret=f22a813b284137ed10d
```

It will automatically convert to:

```
https://global-lebtest-play.myqcloud.com/live/lebtest.m3u8?txSecret=f22a813b284137e
```

2. Specified downgrade

In Adaptive Bit Rate (ABR) playback scenarios, if downgrading is necessary, one cannot simply convert formats to obtain the adaptive bitrate HLS address; it must be manually specified. Or in other scenarios where the user wishes to manually specify, downgrade addresses can be set in the following manner. The address is not limited to HLS protocol; it can also be of other protocols:

```
var player = TCPlayer('player-container-id',{
    sources: 'webrtc://global-lebtest-play.myqcloud.com/live/lebtest?txSecret=f22a8
    webrtcConfig: {
      fallbackUrl: 'https://global-lebtest-play.myqcloud.com/live/lebtest_HLSABR.m3
    },
});
```

Downgrade callback

When a downgrade is triggered, the player will initiate a callback:

```
player.on('webrtcfallback', function(event) {
    console.log(event);
});
```

Downgrade scenarios

Live Event Broadcasting is based on WebRTC and depends on the operating system and browser support for WebRTC.
Currently, the SDK has been tested on the following operating systems and browsers, with the test results as follows:

Operating system	OS Version	Browser	Browser version	Support for stream pull
Windows	win 10	Chrome	86+	✓
		Firefox	88+	✓
		Microsoft Edge	86+	✓
macOS	10.5+	Safari	13.1+	✓
		Chrome	86+	✓
		Firefox	88+	✓
		Microsoft Edge	86+	✓
iOS	13.1.1+	Safari	13.7+	✓
		Chrome	86+	✓
		Firefox	33+	✓
		Microsoft Edge	89	✓
		WeChat embedded	-	✓
Android	-	Chrome	86+	✓
		Firefox	88+	✓
		WeChat embedded	X5 core	✓

		WeChat embedded	XWeb core	✓

Additionally, in some browsers that support WebRTC, there may be decoding failures or server-side issues. In these cases, the player will convert the WebRTC URL to a more compatible HLS URL for playback. This behavior is known as downgrade processing.
To summarize, there are several scenarios that trigger downgrading:
The browser environment does not support WebRTC.
Failed to connect to the server, and the number of retries has exceeded the set value (internal status code -2004).
Decoding failure during playback (internal status code -2005).
Other WebRTC-related errors (internal status code -2001).

Downgrade method

1. Automatic downgrade

During player initialization, the Live Event Broadcasting address is passed through the sources field. In environments requiring downgrade processing, the player automatically converts the protocol, converting the Live Event Broadcasting address to an HLS protocol address.
For example, Live Event Broadcasting address:

```
webrtc://global-lebtest-play.myqcloud.com/live/lebtest?txSecret=f22a813b284137ed10d
```

It will automatically convert to:

```
https://global-lebtest-play.myqcloud.com/live/lebtest.m3u8?txSecret=f22a813b284137e
```

2. Specified downgrade

In Adaptive Bit Rate (ABR) playback scenarios, if downgrading is necessary, one cannot simply convert formats to obtain the adaptive bitrate HLS address; it must be manually specified. Or in other scenarios where the user wishes to manually specify, downgrade addresses can be set in the following manner. The address is not limited to HLS protocol; it can also be of other protocols:

```
var player = TCPlayer('player-container-id',{
    sources: 'webrtc://global-lebtest-play.myqcloud.com/live/lebtest?txSecret=f22a8
    webrtcConfig: {
      fallbackUrl: 'https://global-lebtest-play.myqcloud.com/live/lebtest_HLSABR.m3
    },
});
```

Downgrade callback

When a downgrade is triggered, the player will initiate a callback:

```
player.on('webrtcfallback', function(event) {
    console.log(event);
});
```

iOS Integration
Integration Guide

Last updated：2024-04-26 11:09:31

This document describes how to quickly integrate RT-Cube's LiteAVSDK_Player for iOS into your project.

Environment Requirements

Xcode 9.0 or later
iPhone or iPad with iOS 9.0 or later
A valid developer signature for your project

Integrating the SDK

You can use CocoaPods to automatically load the SDK or manually download the SDK and import it into your project.

Integration via CocoaPods

1. **Install CocoaPods.**

Enter the following command in a terminal window (you need to install Ruby on your Mac first):

```
sudo gem install cocoapods
```

2. **Create a Podfile.**

Go to the directory of your project and enter the following command to create a Podfile in the directory.

```
pod init
```

3. **Edit the Podfile.**

Use CocoaPod's official source, which allows version selection. Edit the Podfile:

Directly integrate the latest version of `TXLiteAVSDK_Player_Premium` as a Pod:

```
platform :ios, '9.0'
source 'https://github.com/CocoaPods/Specs.git'

target 'App' do
pod 'TXLiteAVSDK_Player_Premium'
end
```

To specify a version, you can add the following dependency to the `podfile` file:

```
pod 'TXLiteAVSDK_Player_Premium', '~> 110.8.29000'
```

4. **Update the local repository and install the SDK.**

Enter the following command in a terminal window to update the local repository file and install LiteAVSDK:

```
pod install
```

Or, run this command to update the local repository:

```
pod update
```

An XCWORKSPACE project file integrated with LiteAVSDK will be generated. Double-click to open the file.

Manual SDK integration

1. Download the package of the SDK and demo on the latest version of TXLiteAVSDK_Player_Premium.

2. Add `SDK/TXLiteAVSDK_Player_Premium.framework` to the project to be integrated and select **Do Not Embed**.

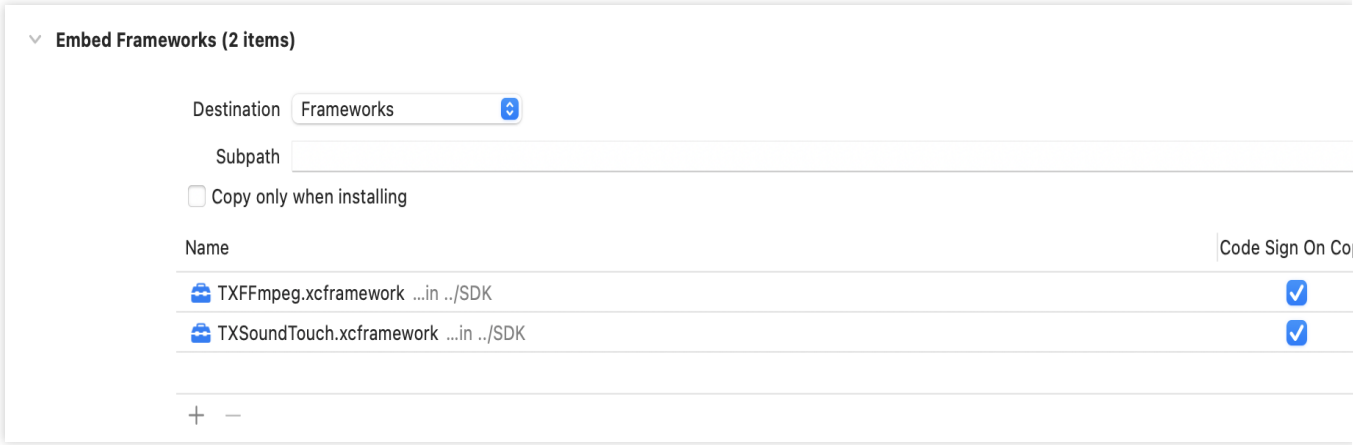
3. You need to configure `-ObjC` of the project target; otherwise, the SDK will crash as the SDK category cannot be loaded.

```
Open Xcode, select the target, select the **Build Settings** tab, search for "Other
```

4. Add library files (in the SDK directory)

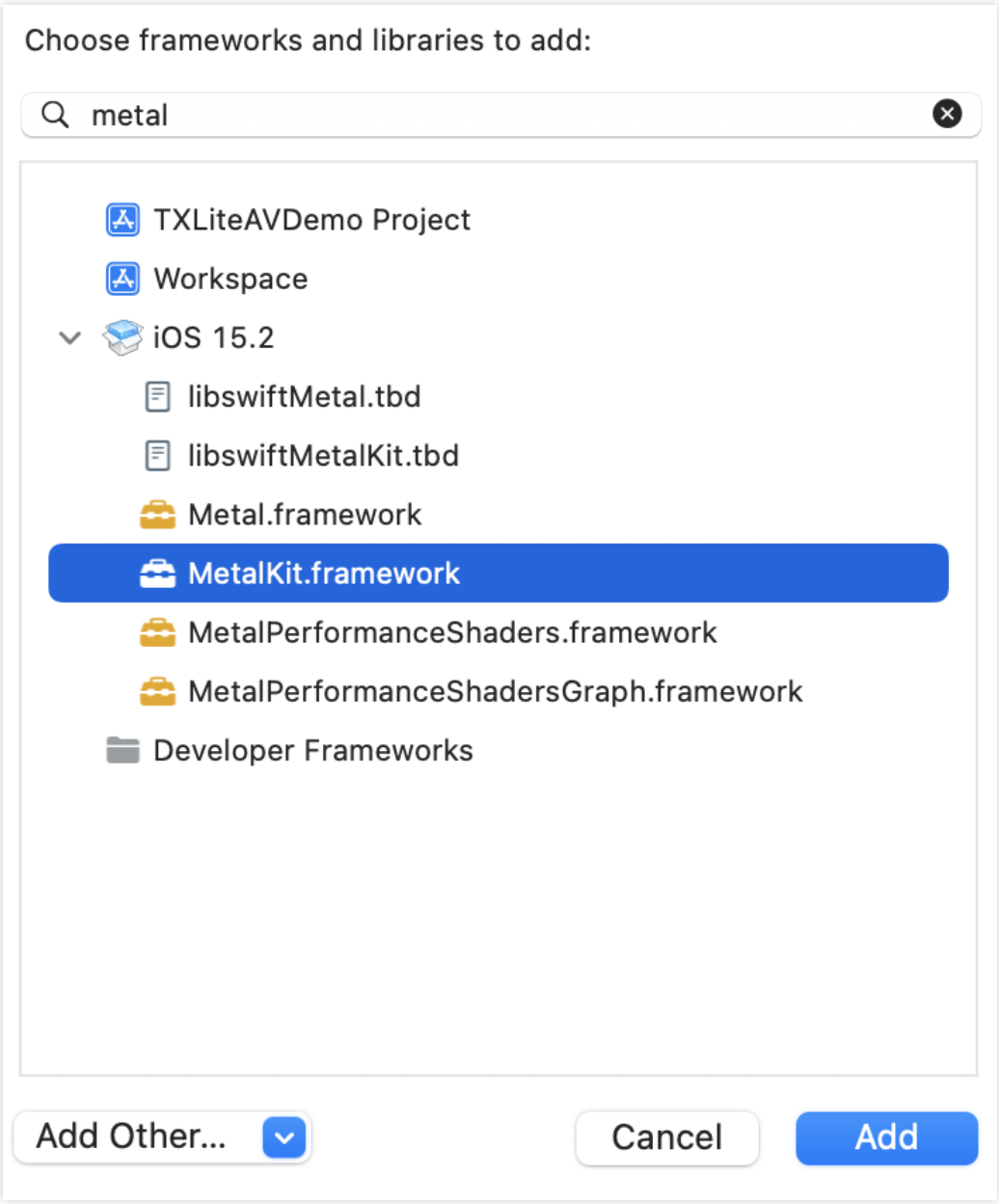
TXFFmpeg.xcframework: Add the .xcframework file to the project, set it to **Embed & Sign** in **General** > **Frameworks, Libraries, and Embedded Content**, and check whether **Code Sign On Copy** is selected in **Build Phases** > **Embed Frameworks** in your **project settings** as shown below:

TXSoundTouch.xcframework: Add the .xcframework file to the project, set it to **Embed & Sign** in **General** > **Frameworks, Libraries, and Embedded Content**, and check whether **Code Sign On Copy** is selected in **Build Phases** > **Embed Frameworks** in your **project settings** as shown below:

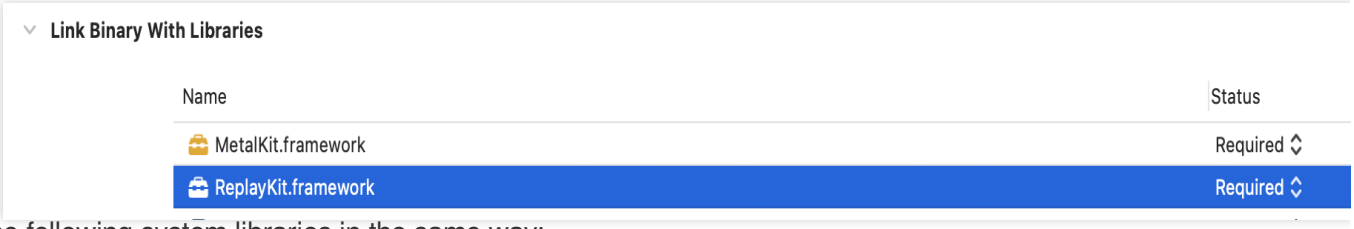


Then, select **Build Settings** > **Search Paths** in Xcode and add the path of the above frameworks in **Framework Search Paths**.

MetalKit.framework: Open Xcode, go to your **project settings**, select **Build Phases** > **Link Binary With Libraries**, click **+** in the bottom-left corner, and enter "MetalKit" to add it to the project as shown below:



ReplayKit.framework: Open Xcode, go to your **project settings**, select **Build Phases** > **Link Binary With Libraries**, click **+** in the bottom-left corner, and enter "ReplayKit" to add it to the project as shown below:



Add the following system libraries in the same way:

System frameworks: SystemConfiguration, CoreTelephony, VideoToolbox, CoreGraphics, AVFoundation, Accelerate, and MobileCoreServices.

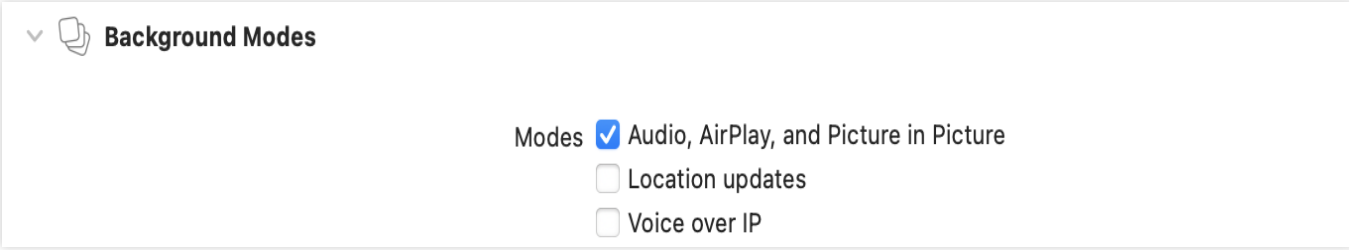
System libraries: libz, libresolv, libiconv, libc++, and libsqlite3.

Picture-in-picture (PiP) feature

To use the PiP feature, configure as shown below. If you don't need the PiP feature, skip this part.

1. To use the PiP feature of iOS, upgrade the SDK to 10.3 or later.

2. To use the PiP feature, you need to enable the background mode. In Xcode, select the target, click **Signing & Capabilities** > **Background Modes**, and select **Audio, AirPlay, and Picture in Picture** as shown below:



Importing the SDK

There are two ways to import the SDK in your project code.

Method 1: import the SDK module in the files that need to use the SDK's APIs in your project

```
@import TXLiteAVSDK_Player_Premium;
// If you are using the basic version of the player, please use: @import
TXLiteAVSDK_Player;
```

Method 2: import a specific header file in the files that need to use the SDK's APIs in your project

```
#import "TXLiteAVSDK_Player_Premium/TXLiteAVSDK.h"
// If you are using the basic version of the player, please use: #import
"TXLiteAVSDK_Player/TXLiteAVSDK.h"
```

Configuring License

1. Enter VOD console. to apply for a trial license as instructed in Adding and Renewing a License. If no license is configured, video playback will fail. You will get two strings: a license URL and a decryption key.

2. After obtaining the License information, you need to initialize and configure the License before calling the relevant interfaces of the SDK. For detailed tutorials, please see Configuring View License.

FAQs

What should I do if a duplicate symbol error occurs because my project integrates multiple editions of LiteAVSDK such as CSS, TRTC, and Player?

If you integrate two or more editions of LiteAVSDK (MLVB, Player, TRTC, UGSV), a library conflict error will occur when you build your project. This is because some symbol files are shared among the underlying libraries of the SDKs. To solve the problem, we recommend you integrate the All-in-One SDK, which includes the features of MLVB, Player, TRTC, and UGSV. For details, see SDK Download.

How to call the API method of the SDK in Swift project?

If you want to call the API interface of the SDK in the Swift project, there are two ways:

Method 1: Using a bridging header file

1. Create a bridging header file, for example, ***-Bridging-Header.h, and add the following code: #import <TXLiteAVSDK_Player_Premium/TXLiteAVSDK.h>.
2. Configure the Objective-c Bridging header option in the Build Setting of the project. Set the path of the bridging file and add it to Objective-c Bridging header (e.g., $(SRCROOT)/SwiftCallOC/***-Bridging-Header.h, depending on the specific path of the project). Compile and run the project.

Method 2: Using the module.modulemap file in the SDK

1. Check if the TXLiteAVSDK_Player_Premium.framework contains the Modules - module.modulemap file (Player SDK provides it by default).
2. Configure the Swift Compiler - Search Paths option in the Build Setting of the project. Add the directory path where the module.modulemap file is located or the parent directory path. Here, it can be: ${PODS_ROOT}/TXLiteAVSDK_Player_Premium/TXLiteAVSDK_Player_Premium/TXLiteAVSDK_Player_Premium.framework/Modules (depending on the specific path of the project).
3. At the top of the class where you need to call the method, use import TXLiteAVSDK_Player_Premium to import and call the relevant methods.

For the above integration methods and demo, please refer to the GitHub demo for details.

VOD Scenario

Last updated：2025-06-23 11:43:04

Limits

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

This Document Describes

How to integrate the Tencent Cloud Player SDK for iOS.
How to use the Player SDK for VOD playback.
How to use the underlying capabilities of the Player SDK to implement more features.

SDK Integration

Step 1. Integrate the SDK ZIP file

Download and integrate the SDK ZIP file as instructed in Integration Guide.

Step 2. Configure the license

If you have obtained a license, you can view the license URL and key in the VOD console.
If you don't have the required license yet, you can get it as instructed in Adding and Renewing a License.
After obtaining the License information, you need to initialize and configure the License before calling the relevant interfaces of the SDK. For detailed tutorials, please see Configuring View License.

Step 3. Create a player object

The `TXVodPlayer` module of the Player SDK is used to implement the VOD feature.

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

Step 4. Create a rendering view

In iOS, a view is used as the basic UI rendering unit. Therefore, you need to configure a view, whose size and position you can adjust, for the player to display video images on.

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

Technically, the player does not render video images directly on the view (`_myView` in the sample code) you provide. Instead, it creates a subview for OpenGL rendering over the view.
You can adjust the size of video images by changing the size and position of the view. The SDK will make changes to the video images accordingly.

How to make an animation

You are allowed great flexibility in view animation, but note that you need to modify the `transform` rather than `frame` attribute of the view.

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

Step 5. Start playback

`TXVodPlayer` supports two playback modes for you to choose as needed:
Through URL
Through `fileId`
`TXVodPlayer` will internally recognize the playback protocol automatically. You only need to pass in your playback URL to the `startPlay` function.

```
// Play back a video resource at a URL
NSString* url = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
[_txVodPlayer startVodPlay:url];

// Play back a local video resource in the sandbox
// Get the `Documents` path
NSString *documentPath = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
// Get the local video path
NSString *videoPath = [NSString stringWithFormat:@"%@/video1.m3u8",documentPath];
[_txVodPlayer startVodPlay:videoPath];
```

```
TXPlayerAuthParams *p = [TXPlayerAuthParams new];
p.appId = 1252463788;
```

```
p.fileId = @"4564972819220421305";
// `psign` is a player signature. For more information on the signature and how to
p.sign = @"psignxxxxx"; // The player signature
[_txVodPlayer startVodPlayWithParams:p];
```

You can go to Media Assets and find it. After clicking it, you can view its `fileId` in the video details on the right. Play back the video through the `fileId`, and the player will request the backend for the real playback URL. If the network is abnormal or the `fileId` doesn't exist, the `PLAY_ERR_GET_PLAYINFO_FAIL` event will be received; otherwise, `PLAY_EVT_GET_PLAYINFO_SUCC` will be received, indicating that the request succeeded.

Step 6. Stop playback

Remember to use **removeVideoWidget** to terminate the view control before exiting the current UI when stopping playback. This can prevent memory leak and screen flashing issues.

```
// Stop playback
[_txVodPlayer stopPlay];
[_txVodPlayer removeVideoWidget]; // Remember to terminate the view control
```

Basic Feature Usage

1. Playback control

Starting playback

```
// Start playback
[_txVodPlayer startVodPlay:url];
```

Pausing playback

```
// Pause the video
[_txVodPlayer pause];
```

Resuming playback

```
// Resume the video
[_txVodPlayer resume];
```

Stopping playback

```
// Stop the video
```

```
[_txVodPlayer stopPlay];
```

Adjusting playback progress (seek)

When the user drags the progress bar, `seek` can be called to start playback at the specified position. The Player SDK supports accurate seek.

```
int time = 600; // In seconds if the value is of `int` type
// Adjust the playback progress
[_txVodPlayer seek:time];
```

Precise and Imprecise Seek

Starting from version 11.8 of the player SDK, it is supported to specify precise or imprecise seek when calling the seek interface.

```
float time = 600; // float type, unit is seconds
// Adjust progress
[_txVodPlayer seek:time accurateSeek:YES]; // Accurate seek
[_txVodPlayer seek:time accurateSeek:NO]; // Non-accurate seek
```

Seek to the specified Program Date Time (PDT) point in the video stream

To seek to the specified Program Date Time (PDT) point in the video stream, which enables functions such as fast-forward, rewind, and progress bar jumping, currently only HLS video format is supported.
Note: Starting from version 11.6 of the player's advanced edition, this function is supported.

```
long long pdtTimeMs = 600; // Unit is milliseconds
[_txVodPlayer seekToPdtTime:time];
```

Specifying playback start time

You can specify the playback start time before calling `startVodPlay` for the first time.

```
float startTimeInSecond = 60; // Unit: Second
[_txVodPlayer setStartTime:startTimeInSecond];  // Set the playback start time
[_txVodPlayer startVodPlay:url];
```

2. Image adjustment

view: size and position

You can modify the size and position of the view by adjusting the size and position of the parameter `view` of setupVideoWidget. The SDK will automatically adjust the size and position of the view based on your configuration.

setRenderMode: Aspect fill or aspect fit

Value	Description
RENDER_MODE_FILL_SCREEN	Images are scaled to fill the entire screen, and the excess parts are cropped. There are no black bars in this mode, but images may not be displayed in whole.
RENDER_MODE_FILL_EDGE	Images are scaled as large as the longer side can go. Neither side exceeds the screen after scaling. Images are centered, and there may be black bars.

setRenderRotation: Image rotation

Value	Description
HOME_ORIENTATION_RIGHT	The Home button is on the right of the video image
HOME_ORIENTATION_DOWN	The Home button is below the video image
HOME_ORIENTATION_LEFT	The Home button is on the left of the video image
HOME_ORIENTATION_UP	The Home button is above the video image

3. Adjustable-Speed playback

The VOD player supports adjustable-speed playback. You can use the `setRate` API to set the VOD playback speed, such as 0.5x, 1.0x, 1.2x, and 2x speed.

```
// Set playback at 1.2X rate
[_txVodPlayer setRate:1.2];
// Start playback
[_txVodPlayer startVodPlay:url];
```

4. Playback loop

```
// Set playback loop
[_txVodPlayer setLoop:true];
// Get the current playback loop status
[_txVodPlayer loop];
```

5. Muting/Unmuting

```
// Mute or unmute the player. true: Mute; false: Unmute
[_txVodPlayer setMute:true];
```

6. Screencapturing

Call **snapshot** to take a screenshot of the current video frame. This method captures only the video frame. To capture the UI, use the corresponding API of the iOS system.

7. Roll image ad

The Player SDK allows you to add roll images on the UI for advertising as follows:
If `autoPlay` is set to `NO`, the player will load the video normally but will not immediately start playing it back. Users can see the roll image ad on the player UI after the player is loaded and before the video playback starts. When the ad display stop conditions are met, the `resume` API will be called to start video playback.

8. HTTP-REF

`headers` in `TXVodPlayConfig` can be used to set HTTP request headers, such as the `Referer` field commonly used to prevent the URL from being copied arbitrarily (Tencent Cloud provides a more secure signature-based hotlink protection solution) and the `Cookie` field for client authentication.

```
NSMutableDictionary<NSString *, NSString *> *httpHeader = [[NSMutableDictionary all
[httpHeader setObject:@"${Referer Content}" forKey:@"Referer"];
[_config setHeaders:httpHeader];
[_txVodPlayer setConfig:_config];
```

9. Hardware acceleration

It is extremely difficult to play back videos of the Blu-ray (1080p) or higher image quality smoothly if only software decoding is used. Therefore, if your main scenario is game live streaming, we recommend you use hardware acceleration.
Before switching between software and hardware decoding, you need to call **stopPlay** first. After the switch, you need to call **startVodPlay**; otherwise, severe blurs will occur.

```
[_txVodPlayer stopPlay];
_txVodPlayer.enableHWAcceleration = YES;
[_txVodPlayer startVodPlay:_flvUrl type:_type];
```

10. Definition settings

The SDK supports the multi-bitrate format of HLS, so users can switch between streams at different bitrates. You can get the array of multiple bitrates as follows:

```
NSArray *bitrates = [_txVodPlayer supportedBitrates]; // Get the array of multiple
// TXBitrateItem class field meaning: index-bitrate subscript; width-video width; h
TXBitrateItem *item = [bitrates objectAtIndex:i];
[_txVodPlayer setBitrateIndex:item.index]; // Switch bit rate to desired definition
```

```
// Get the bit rate subscript of the current playback, the return value -1000 is th
int index = [_txVodPlayer bitrateIndex];
```

During playback, you can call `-[TXVodPlayer setBitrateIndex:]` at any time to switch the bitrate. During switch, the data of another stream will be pulled. The SDK is optimized for Tencent Cloud multi-bitrate files to implement smooth switch.

If you know the resolution information of the video stream in advance, you can specify the resolution of the video to be played before starting the broadcast, so as to avoid switching the stream after playback. For detailed methods, refer to Player configuration#Specify resolution before starting broadcast.

11. Adaptive bitrate streaming

The SDK supports adaptive bitrate streaming of HLS. After this capability is enabled, the player can dynamically select the most appropriate bitrate for playback based on the current bandwidth. You can enable adaptive bitrate streaming as follows:

```
[_txVodPlayer setBitrateIndex:-1]; // Pass in `-1` for the `index` parameter
```

During playback, you can call `-[TXVodPlayer setBitrateIndex:]` at any time to switch to another bitrate. After the switch, adaptive bitrate streaming will be disabled.

12. Enabling smooth bitrate switch

Before starting playback, you can enable smooth bitrate switch to seamlessly switch between different definitions (bitrates) during playback. If smooth bitrate switch is enabled, the transition between different bitrates will be smoother but will be more time-consuming. Therefore, this feature can be configured as needed.

```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
// If it is set to `YES`, the bitrate can be switched smoothly when IDR frames are
[_config setSmoothSwitchBitrate:YES];
[_txVodPlayer setConfig:_config];
```

13. Playback progress listening

There are two metrics for the VOD progress: **loading progress** and **playback progress**. Currently, the SDK notifies the two progress metrics in real time through event notifications. For more information on the event notification content, see Event Listening.

```
-(void) onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary*
    if (EvtID == PLAY_EVT_PLAY_PROGRESS) {
            // Loading progress in seconds. The decimal part is in ms.
            float playable = [param[EVT_PLAYABLE_DURATION] floatValue];
                [_loadProgressBar setValue:playable];

            // Playback progress in seconds. The decimal part is in ms.
```

```
            float progress = [param[EVT_PLAY_PROGRESS] floatValue];
                [_seekProgressBar setValue:progress];

            // Total video duration in seconds. The decimal part is in ms.
            float duration = [param[EVT_PLAY_DURATION] floatValue];
            // It can be used to set duration display, etc.

            // Get the PDT time. This function is supported starting from version 1
            long long pdt_time_ms = [param[VOD_PLAY_EVENT_PLAY_PDT_TIME_MS] longLon
    }
}
```

14. Playback network speed listening

You can display the current network speed when the video is lagging by listening on events.

You can use the `NET_SPEED` of `onNetStatus` to get the current network speed. For detailed directions, see Playback status feedback (onNetStatus).

After the `PLAY_EVT_PLAY_LOADING` event is detected, the current network speed will be displayed.

After the `PLAY_EVT_VOD_LOADING_END` event is received, the view showing the current network speed will be hidden.

15. Video resolution acquisition

The Player SDK plays back a video through a URL string. The URL doesn't contain the video information, and you need to access the cloud server to load such information. Therefore, the SDK can only send the video information to your application as event notifications. For more information, see Event Listening.

Resolution information

Method 1

Method 2

Use the `VIDEO_WIDTH` and `VIDEO_HEIGHT` of `onNetStatus` to get the video width and height. For detailed directions, see Status feedback (onNetStatus).

Directly call `-[TXVodPlayer width]` and `-[TXVodPlayer height]` to get the current video width and height.

16. Player buffer size

During normal video playback, you can control the maximum size of the data buffered from the network in advance. If the maximum buffer size is not configured, the player will use the default buffer policy to guarantee a smooth playback experience.

```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
[_config setMaxBufferSize:10];  // Maximum buffer size during playback in MB
[_txVodPlayer setConfig:_config];  // Pass in `config` to `_txVodPlayer`
```

17. Local video cache

In short video playback scenarios, the local video file cache is required, so that general users don't need to consume traffic again to reload an already watched video.

Supported format: The SDK supports caching videos in two common VOD formats: HLS (M3U8) and MP4.

Enablement time: The SDK doesn't enable the caching feature by default. We recommend you do not enable it for scenarios in which most videos are watched only once.

Enablement method: To enable it, you need to configure two parameters: local cache directory and cache size.

```
// Set the global cache directory of the playback engine
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDom
NSString *documentsDirectory = [paths objectAtIndex:0];
NSString *preloadDataPath = [documentsDirectory stringByAppendingPathComponent:@"/p
if (![[NSFileManager defaultManager] fileExistsAtPath:preloadDataPath]) {
   [[NSFileManager defaultManager] createDirectoryAtPath:preloadDataPath
                            withIntermediateDirectories:NO
                                             attributes:nil
                                                  error:&error];
[TXPlayerGlobalSetting setCacheFolderPath:preloadDataPath];
// Set the playback engine cache size
[TXPlayerGlobalSetting setMaxCacheSize:200];
// Start playback
[_txVodPlayer startVodPlay:url];
```

Note:

The `TXVodPlayConfig#setMaxCacheItems` API used for configuration on earlier versions has been deprecated and is not recommended.

18.Screen control (screen on and off)

Due to the frequent personalized settings in mobile phones, the screen lock time is often set, which may cause the screen to turn off (or be locked) during video playback, greatly affecting the user experience. Therefore, to solve this problem, the following code needs to be added at relevant times during the playback process to keep the screen always on.

```
 (1) Bright screen (no screen off)
// start playing (startVodPlay / startPlayDrm / startVodPlayWithParams)
// resume playback (resume)
 [[UIApplication sharedApplication] setIdleTimerDisabled:YES];

 (2) Off screen (restore off screen)
// stop (stopPlay)
// pause (pause)
[[UIApplication sharedApplication] setIdleTimerDisabled:NO];
```

Note:

Please pay attention to calling the above interfaces in the main thread.

19.DRM encrypted video playback

Note：

This feature requires the premium version of the player to be supported.

The advanced version of the player SDK supports playback of commercial-grade DRM-encrypted videos, and currently supports two DRM schemes, WideVine and Fairplay. For more commercial-grade DRM information, please refer to the product introduction.

DRM-encrypted videos can be played in the following 2 ways:

Play by FiledId

Custom configuration play

```
TXPlayerAuthParams *p = [TXPlayerAuthParams new];
p.appId = ${appId}; // appId of the Tencent Cloud account
p.fileId = @"${fieId}"; // fileId of DRM encrypted video
// psign is the signature of the player. Please refer to the link for signature int
p.sign = @"${psgin}"; // Player signature for encrypted video
[_txVodPlayer startVodPlayWithParams:p];
```

Playing via FileId is suitable for accessing the cloud on-demand background. This method is no different from playing ordinary FileId files. You need to configure the resource as a DRM type in Cloud VOD, and the SDK will recognize and process it internally.

```
// Play through the TXVodPlayer#startPlayDrm interface
// @param certificateUrl certificate provider url
// @param licenseUrl decrypted key url
// @param videoUrl Url address of the video to be played
TXPlayerDrmBuilder *builder = [[TXPlayerDrmBuilder alloc] initWithDeviceCertificate
[_txVodPlayer startPlayDrm:builder];
```

20.External subtitles

Note：

This feature requires the premium version of the player to be supported.

The advanced version of the player SDK supports adding and switching external subtitles, and now supports subtitles in two formats: SRT and VTT.

Best practice: It is recommended to add subtitles and configure subtitle styles before calling startVodPlay. After receiving the PLAY_EVT_VOD_PLAY_PREPARED event, call selectTrack to choose the subtitle. Adding subtitles does not automatically load them. After calling selectTrack, the subtitles will be loaded. The successful selection of subtitles will trigger the VOD_PLAY_EVT_SELECT_TRACK_COMPLETE event callback.

The usage is as follows：

Step 1: Add external subtitles

```
// Pass in subtitle url, subtitle name, subtitle type. It is recommended to add sub
[_txVodPlayer addSubtitleSource:@"https://mediacloud-76607.gzc.vod.tencent-cloud.co
```

Step 2: Switch subtitles after playback.

```
// After starting to play the video, select the added external subtitles. Please ca
NSArray<TXTrackInfo *> *subtitlesArray = [_txVodPlayer getSubtitleTrackInfo];
for (int i = 0; i < subtitlesArray.count; i++) {
    TXTrackInfo *info = subtitlesArray[i];
    if (info.trackIndex == 0) {
        [_txVodPlayer selectTrack:info.trackIndex];  // check subtitles
    } else {
    // If other subtitles are not needed, perform deselectTrack
    [_txVodPlayer deselectTrack:info.trackIndex];
    }
}

// Listen for track switch messages
- (void)onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary
    if (EvtID == VOD_PLAY_EVT_SELECT_TRACK_COMPLETE) {
        int trackIndex = [(NSNumber *)[param valueForKey:EVT_KEY_SELECT_TRACK_INDEX
        int errorCode = [(NSNumber *)[param valueForKey:EVT_KEY_SELECT_TRACK_ERROR_
        NSLog(@"receive VOD_PLAY_EVT_SELECT_TRACK_COMPLETE, trackIndex=%d , errorCo
    }
}
```

Step 3: Configure subtitle style.

The subtitle style supports configuration before or during playback.

```
// For detailed parameter configuration, please refer to the API documentation
TXPlayerSubtitleRenderModel *model = [[TXPlayerSubtitleRenderModel alloc] init];
model.canvasWidth = 1920;  // The width of the subtitle rendering canvas
model.canvasHeight = 1080;  // The height of the subtitle rendering canvas
model.isBondFontStyle = NO;  // Set whether the subtitle font is bold
model.fontColor = 0xFF000000; // Set the subtitle font color, the default is white
[_txVodPlayer setSubtitleStyle:model];
```

21.Subtitle text callback

Note：

This feature is supported starting from Player Premium 12.3.

The default configuration of the player premium SDK is to render and display subtitles through the built-in engine. You can modify the configuration to support callback text. The business can render and display the subtitle text by itself after obtaining it. Subtitles in SRT and VTT formats are currently supported.

The detailed usage is as follows:

Step 1: Set up the subtitle text callback

```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
NSMutableDictionary<NSString *, id> *extInfoMap = [NSMutableDictionary dictionary];
[extInfoMap setObject:@(0) forKey:@"450"];
[_config setExtInfoMap:extInfoMap];
[_txVodPlayer setConfig:_config];
```

Step 2: Add and select subtitles

To add and select subtitle files, please refer to the External Subtitles section.

Step 3: Register to listen for subtitle text callback

After selecting the subtitle, you can register the following interface to listen to the subtitle text content. The meaning of the relevant fields: `TXVodSubtitleData#trackIndex,` the track index of the current subtitle; `TXVodSubtitleData#subtitleData,` the actual subtitle text content. When the callback subtitleData is empty, it means that the subtitle is empty. The business can be encapsulated and displayed; the other fields of the `TXVodSubtitleData` class have no practical meaning for the time being, so don't pay attention to them.

```
// protocol TXVodPlayListener
- (void)onPlayer:(TXVodPlayer *)player subtitleData:(TXVodSubtitleData *)subtitleDa
    long trackIndex = subtitleData.trackIndex; // The track index of the current su
    NSString *data = subtitleData.subtitleData;   // The actual subtitle text conte
    // Display the data subtitle text content as needed
}
```

22.Switching between multiple audio tracks

The advanced version of the player SDK supports switching between multiple audio tracks built into the video. The usage is as follows:

```
NSArray<TXTrackInfo *> *soundTrackArray = [_txVodPlayer getAudioTrackInfo];
for (int i = 0; i < soundTrackArray.count; i++) {
    TXTrackInfo *info = soundTrackArray[i];
    if (info.trackIndex == 0) {
        // Switch to the desired audio track by determining the trackIndex or name.
        [_txVodPlayer selectTrack:info.trackIndex];
    } else {
        // If other subtitles are not required, proceed with deselectTrack.
        [_txVodPlayer deselectTrack:info.trackIndex];
    }
}
```

Using Advanced Features

1. Video preloading

Step 1. Use video preloading

In UGSV playback scenarios, the preloading feature contributes to a smoother viewing experience: While watching a video, you can load the URL of the next video to be played back on the backend. When the next video is switched to, it will be preloaded and can be played back immediately.
Video preloading can deliver an instant playback effect but has certain performance overheads. It will occupy download bandwidth and thread resources. It is recommended that the number of concurrent video pre-playbacks be controlled within 3. If your business needs to preload many videos, we recommend you use this feature together with video predownloading.
This is how seamless switch works in video playback. You can use `isAutoPlay` in `TXVodPlayer` to implement the feature as follows:

```
// Play back video A: If `isAutoPlay` is set to `YES`, the video will be immediatel
NSString* url_A = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f10.mp4";
_player_A.isAutoPlay = YES;
[_player_A startVodPlay:url_A];

// To preload video B when playing back video A, set `isAutoPlay` to `NO`
NSString* url_B = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
_player_B.isAutoPlay = NO;
[_player_B startVodPlay:url_B];
```

After video A ends and video B is automatically or manually switched to, you can call the `resume` function to immediately play back video B.

Note:

After `autoPlay` is set to `false`, make sure that video B has been prepared before calling `resume`, that is, you should call it only after the `PLAY_EVT_VOD_PLAY_PREPARED` event of video B (2013: the player has been prepared, and the video can be played back) is detected.

```
-(void) onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary*
{
    // When video A ends, directly start playing back video B for seamless switch
    if (EvtID == PLAY_EVT_PLAY_END) {
            [_player_A stopPlay];
            [_player_B setupVideoWidget:mVideoContainer insertIndex:0];
```

```
            [_player_B resume];
        }
}
```

Step 2. Configure the video preloading buffer

You can set a large buffer to play back videos more smoothly under unstable network conditions.
You can set a smaller buffer to reduce the traffic consumption.

Preloading buffer size

This API is used to control the maximum buffer size before the playback starts in preloading scenarios (that is, `AutoPlay` of the player is set to `false` before video playback starts).

```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
[_config setMaxPreloadSize:(2)];;  // Maximum preloading buffer size in MB. Set it
[_txVodPlayer setConfig:_config];  // Pass in `config` to `_txVodPlayer`
```

Playback buffer size

During normal video playback, you can control the maximum size of the data buffered from the network in advance. If the maximum buffer size is not configured, the player will use the default buffer policy to guarantee a smooth playback experience.

```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
[_config setMaxBufferSize:10];  // Maximum buffer size during playback in MB
[_txVodPlayer setConfig:_config];  // Pass in `config` to `_txVodPlayer`
```

2. Video predownloading

You can download part of the video content in advance without creating a player instance, so as to start playing back the video faster when using the player. This helps deliver a better playback experience.
Before using the playback service, make sure that video cache has been set.

Note:

Video pre-downloading will occupy download bandwidth and thread resources. It is recommended to control the queue and limit the number of concurrent downloads to less than 3.
`TXPlayerGlobalSetting` is the global cache setting API, and the original `TXVodConfig` API has been deprecated.
The global cache directory and size settings have a higher priority than those configured in `TXVodConfig` of the player.
Pre-download by media URL
Pre-download by media filedId
An example code for pre-downloading a video via a media asset URL is as follows:

```
// Set the global cache directory of the playback engine
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDom
NSString *documentsDirectory = [paths objectAtIndex:0];
NSString *preloadDataPath = [documentsDirectory stringByAppendingPathComponent:@"/p
if (![[NSFileManager defaultManager] fileExistsAtPath:preloadDataPath]) {
    [[NSFileManager defaultManager] createDirectoryAtPath:preloadDataPath
                               withIntermediateDirectories:NO
                                                attributes:nil
                                                     error:&error]; //Create folder
}
[TXPlayerGlobalSetting setCacheFolderPath:preloadDataPath];

// Set the playback engine cache size
[TXPlayerGlobalSetting setMaxCacheSize:200];
NSString *m3u8url = "http://****";
int taskID = [[TXVodPreloadManager sharedManager] startPreload:m3u8url
                                                   preloadSize:10
                                          preferredResolution:1920*1080
                                                      delegate:self];


// Cancel predownloading
[[TXVodPreloadManager sharedManager] stopPreload:taskID];
```

Note:

Pre-download by fileId is supported since version 11.3.

Pre-downloading by fileId is a time-consuming operation, please do not call it in the main thread, otherwise an illegal call exception will be thrown. The preferredResolution passed in during startPreload must be consistent with the preferred resolution set when starting the broadcast, otherwise the expected effect will not be achieved. An example of use is as follows:

```
//Set the global cache directory of the playback engine
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDom
NSString *documentsDirectory = [paths objectAtIndex:0];
NSString *preloadDataPath = [documentsDirectory stringByAppendingPathComponent:@"/p
if (![[NSFileManager defaultManager] fileExistsAtPath:preloadDataPath]) {
    [[NSFileManager defaultManager] createDirectoryAtPath:preloadDataPath
                               withIntermediateDirectories:NO
                                                attributes:nil
                                                     error:&error]; //Create folder
}
[TXPlayerGlobalSetting setCacheFolderPath:preloadDataPath];
//Set playback engine cache size
[TXPlayerGlobalSetting setMaxCacheSize:200];

TXPlayerAuthParams *params = [[TXPlayerAuthParams alloc] init];
```

```
params.appId = ${appId};
params.fileId = @"${fileId}";
params.sign = @"${psign}";
// Note: Time-consuming operation, please do not call it in the main thread! Callin
int taskID = [[TXVodPreloadManager sharedManager] startPreload:params
                                                    preloadSize:10
                                          preferredResolution:1920*1080
                                                      delegate:self]; // TXVodPrelo

// Set the playback engine cache size
[[TXVodPreloadManager sharedManager] stopPreload:taskID];
```

3. Video download

Video download allows users to download online videos and watch them offline. If the video is encrypted, the downloaded video through the player SDK will be kept in an encrypted state locally and can only be decrypted and played through Tencent Cloud Player SDK. This can effectively prevent illegal dissemination of downloaded videos and protect video security.

As HLS streaming media cannot be directly saved locally, you cannot download them and play back them as local files. You can use the video download scheme based on `TXVodDownloadManager` to implement offline HLS playback.

Note:

Video download supports downloading MP4 and HLS videos. For nested HLS videos, you need to specify the preferred resolution.

Step 1. Make preparations

When the SDK is initialized, set the global storage path for functions such as video download, preload, and cache. The usage is as follows:

```
NSString *cachesDir = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSU
NSString downloadPath = [NSString stringWithFormat:@"%@/txdownload",cachesDir];
[TXPlayerGlobalSetting setCacheFolderPath:downloadPath];
```

`TXVodDownloadManager` is designed as a singleton; therefore, you cannot create multiple download objects. It is used as follows:

```
TXVodDownloadManager *downloader = [TXVodDownloadManager shareInstance];
```

Set the httpHeader used for downloading

Configure according to business needs. When the player starts downloading, it will be sent to the server. Player version 12.2 starts to support it.

```
NSDictionary *httpHeader = [[NSMutableDictionary alloc] init];
```

```
[downloader setHeaders:httpHeader]; // Set the download httpHeader
```

Step 2. Start the download

You can start the download through the `fileid` or URL.

Through fileid

Through URL

You need to pass in `appId` and `fileId` at least for download through `fileid`. If you don't specify a value for `userName`, `default` will be used by default. **Note: You can download encrypted videos only through `Fileid` and must enter the `psign` parameter.**

```
TXVodDownloadDataSource *source = [[TXVodDownloadDataSource alloc] init];
source.appId = 1252463788;
source.fileId = @"4564972819220421305";
// // `psign` is a player signature. For more information on the signature and how
source.pSign = @"xxxxxxxxxx";

// Specify the download definition
// Valid values: `TXVodQualityOD` (original), `TXVodQualityFLU` (LD), `TXVodQuality
source.quality = TXVodQualityHD;  // HD

// **Note that if you use the legacy v2 protocol for download, set the `appId` and
// source.auth = auth;  **There is no need to set it by default.**

[downloader startDownload:dataSource];
```

You only need to pass in the download URL. Only the non-nested HLS, i.e., single-bitstream HLS, is supported. Use `fileid` in case of private encryption.

```
[downloader startDownloadUrl:@"http://1253131631.vod2.myqcloud.com/26f327f9vodgzp12
```

Step 3. Receive the task information

Before receiving the task information, you need to set the callback delegate first.

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

You may receive the following task callbacks:

Callback Message	Description
-[TXVodDownloadDelegate onDownloadStart:]	The task started, that is, the SDK started the download.
-[TXVodDownloadDelegate onDownloadProgress:]	Task progress. During download, the SDK will frequently call back this API. You can update the displayed progress here.

-[TXVodDownloadDelegate onDownloadStop:]	The task stopped. When you call stopDownload to stop the download, if this message is received, the download is stopped successfully.
-[TXVodDownloadDelegate onDownloadFinish:]	Download was completed. If this callback is received, the entire file has been downloaded, and the downloaded file can be played back by `TXVodPlayer`.
-[TXVodDownloadDelegate onDownloadError:errorMsg:]	A download error occurred. If the network is disconnected during download, this API will be called back and the download task will stop. For all error codes, see TXDownloadError.

Download error code

error code	value	Meaning
TXDownloadSuccess	0	Download successful
TXDownloadAuthFaild	-5001	Failed to request video information from the cloud on-demand console, it is recommended to check whether the fileId and psign parameters are correct
TXDownloadNoFile	-5003	No file for this resolution
TXDownloadFormatError	-5004	The download file format is not supported
TXDownloadDisconnet	-5005	The network is disconnected, it is recommended to check whether the network is normal
TXDownloadHlsKeyError	-5006	Failed to get HLS decryption key
TXDownloadPathError	-5007	Download directory access failed, it is recommended to check whether you have permission to access the download directory

As the downloader can download multiple files at a time, the callback API carries the `TXVodDownloadMediaInfo` object. You can access the URL or `dataSource` to determine the download source and get other information such as download progress and file size.

Step 4. Stop the download

You can call the `-[TXVodDownloadManager stopDownload:]` method to stop the download. The parameter is the object returned by `-[TXVodDownloadManager sartDownloadUrl:]`. **The SDK supports checkpoint restart.** If the download directory is not changed, when you resume downloading a file, the download will start from the point where it stopped.

Step 5. Manage downloads

1. You can get the download lists of all accounts or the specified account.

```
// s a time-consuming function. Please do not call it in the main thread
NSArray<TXVodDownloadMediaInfo *> *array = [[[TXVodDownloadManager shareInstance] g
// Get the download list of the `default` user
for (TXVodDownloadMediaInfo *info in array) {
    if ([info.userName isEqualToString:@"default"]) {
        // Save the download list of the `default` user
    }
}
```

2. Get the download information of a `FileId` or URL:

2.1 To get the download information of a `Fileid` through the `-[TXVodDownloadManager getDownloadMediaInfo:]` API, such as the current download status and progress, you need to pass in `AppID`, `Fileid`, and `qualityId`.

```
// Get the download information of a `fileId`
TXVodDownloadMediaInfo *sourceMediaInfo = [[TXVodDownloadMediaInfo alloc] init];
TXVodDownloadDataSource *dataSource = [[TXVodDownloadDataSource alloc] init];
dataSource.appId = 1252463788;
dataSource.fileId = @"4564972819220421305";
dataSource.pSign = @"psignxxxx";
dataSource.quality = TXVodQualityHD;
sourceMediaInfo.dataSource = dataSource;
TXVodDownloadMediaInfo *downlaodMediaInfo =  [[TXVodDownloadManager shareInstance]

// Get the total size of the file being downloaded in bytes. This API takes effect
// Note: The total size refers to the size of the original file uploaded to the VOD
downlaodMediaInfo.size; //  Get the total size of the file being downloaded
downlaodMediaInfo.duration;   // Get the total duration
downlaodMediaInfo.playableDuration; // Get the playable duration of the downloaded
downlaodMediaInfo.progress;   // Get the download progress
downlaodMediaInfo.playPath;   // Get the offline playback path, which can be passed
downlaodMediaInfo.downloadState;  // Get the download status. For more information,
[downlaodMediaInfo isDownloadFinished];  // If `YES` is returned, the download is c
```

2.2 To get the download information of a URL, you simply need to pass in the URL information.

```
// Get the download information of a `fileId`
TXVodDownloadMediaInfo *sourceMediaInfo = [[TXVodDownloadMediaInfo alloc] init];
mediaInfo.url = @"videoURL";
TXVodDownloadMediaInfo *downlaodMediaInfo =  [[TXVodDownloadManager shareInstance]
```

3. Delete the download information and relevant file:

If you don't need to resume the download, call the `-[TXVodDownloadManager deleteDownloadFile:]` method to delete the file to release the storage space.

Step 6: Play offline after downloading

The downloaded video can be played without internet connection, no internet connection is required. Once the download is complete, it can be played.

```
NSArray<TXVodDownloadMediaInfo *> *mediaInfoList = [[TXVodDownloadManager shareInst
TXVodDownloadMediaInfo *mediaInfo = [mediaInfoList firstObject]; // Find the curren
if (mediaInfo.downloadState == TXVodDownloadMediaInfoStateFinish) { // Determine wh
    [self.player startVodPlay:mediaInfo.playPath];
}
```

Note:

When downloading and playing offline, be sure to get the download list and play it through the PlayPath of the download list video object TXVodDownloadMediaInfo, do not save the PlayPath object directly.

4. Encrypted playback

The video encryption solution is used in scenarios where the video copyright needs to be protected, such as online education. To encrypt your video resources, you need to alter the player and encrypt and transcode video sources. For more information, see Media Encryption and Copyright Protection Overview.

After you get the `appId` as well as the encrypted video's `fileId` and `psign` in the Tencent Cloud console, you can play back the video as follows:

```
TXPlayerAuthParams *p = [TXPlayerAuthParams new];
p.appId = 1252463788; // The `appId` of the Tencent Cloud account
p.fileId = @"4564972819220421305"; // The video's `fileId`
// `psign` is a player signature. For more information on the signature and how to
p.sign = @"psignxxxxx"; // The player signature
[_txVodPlayer startVodPlayWithParams:p];
```

5. Player configuration

Before calling `statPlay`, you can call `setConfig` to configure the player parameters, such as player connection timeout period, progress callback interval, and maximum number of cached files. `TXVodPlayConfig` allows you to configure detailed parameters. For more information, see TXVodPlayConfig. Below is the configuration sample code:

```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
[_config setEnableAccurateSeek:true];  // Set whether to enable accurate seek. Defa
[_config setMaxCacheItems:5];  // Set the maximum number of cached files to 5
[_config setProgressInterval:200];  // Set the progress callback interval in ms
[_config setMaxBufferSize:50];  // Set the maximum preloading buffer size in MB
```

```
[_txVodPlayer setConfig:_config];  // Pass in `config` to `_txVodPlayer`
```

Specifying resolution when playback starts

When playing back an HLS multi-bitrate video source, if you know the video stream resolution information in advance, you can specify the preferred resolution before playback starts, and the player will select and play back the stream at or below the preferred resolution. In this way, after playback starts, you don't need to call `setBitrateIndex` to switch to the required bitstream.

```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
// The parameter passed in is the product of the video width and height. You can pa
[_config setPreferredResolution:720*1280];
[_txVodPlayer setConfig:_config];  // Pass in `config` to `_txVodPlayer`
```

Specifying media type before playback

When the media type to be played is known in advance, the playback type detection within the player SDK can be reduced and the startup speed can be improved by configuring TXVodPlayConfig#setMediaType.

Note:

TXVodPlayConfig#setMediaType is supported since version 11.2.

```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
[_config setMediaType:MEDIA_TYPE_FILE_VOD];  // Used to increase the speed of MP4 p
// [_config setMediaType:MEDIA_TYPE_HLS_VOD]; // Used to increase the speed of HLS
[_txVodPlayer setConfig:_config];
```

Setting playback progress callback interval

```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
[_config setProgressInterval:200];  // Set the progress callback interval in ms
[_txVodPlayer setConfig:_config];  // Pass in `config` to `_txVodPlayer`
```

Specify the priority audio track before starting playback

Note:

This feature is supported starting from the Premium Player 12.3 version.

When the name of the audio track to be played is known in advance, you can specify the priority audio track before starting playback by configuring TXVodPlayConfig#setPreferredAudioTrack.

```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
[_config setPreferredAudioTrack:@"audioTrackName"]; // audioTrackName is the actual
[_txVodPlayer setConfig:_config];  // Pass config to mVodPlayer
```

6. HttpDNS resolution service

HTTPDNS is a domain name resolution service that sends domain name resolution requests to DNS servers based on the HTTP protocol, replacing the traditional method of sending resolution requests to the operator's local DNS based on the DNS protocol. This method can avoid domain name hijacking and cross-network access issues caused by local DNS, and solve the problem of video playback failure caused by abnormal domain name resolution in mobile internet services.

Note:

HttpDNS resolution service is supported from version 10.9 onwards.

1. To enable HTTPDNS resolution service

you can choose Tencent Cloud or other cloud providers and open the service. Make sure to integrate the service into the playback SDK after successful activation.

2. Accessing HTTPDNS Resolution Service in the Playback SDK

Taking Tencent Cloud HTTPDNS as an example, the following steps demonstrate how to access the service in the player SDK:

It is not recommended for the business layer to cache domain name resolution results. For example, Tencent HTTPDNS has a built-in intelligent cache mechanism. Additional caching on the business side may result in untimely updates.

Disable the LocaIDNS fallback domain name resolution policy. Configure through the DnsConfig of the HttpDns sdk (void) WGSetExpiredIPEnabled:(NO) and (void) WGSetPersistCacheIPEnabled:(NO) to avoid obtaining non-optimal or unavailable IPs in the scenario where LocaIDNS has an exception (such as hijacking, pollution, or recursive resolution failure).

```
// Step 1: Enable HttpDNS Parsing Switch
[TXLiveBase enableCustomHttpDNS:YES];
// Step 2: Implement HttpDNS resolution proxy: TXLiveBaseDelegate#onCustomHttpDNS
- (void)onCustomHttpDNS:(NSString *)hostName ipList:(NSMutableArray<NSString *> *)l
    // The player SDK will callback the hostName to the business. The business can
    // If an empty ip is returned, the SDK will not use httpdns for this network re
    // After resolving the hostName to an ip address, save it to iPList and return
    // MSDKDnsResolver is the HTTPDNS SDK resolution API provided by Tencent Cloud
    // NSArray *result = [[MSDKDns sharedInstance] WGGetHostByName:hostName];
    NSString *ip = nil;
    if (result && result.count > 1) {
        if (![result[1] isEqualToString:@"0"]) {
            ip = result[1];
        } else {
            ip = result[0];
        }
    }
    [list addObject:ip];
}

// Step 3: Set HttpDNS resolution proxy
```

```
[TXLiveBase sharedInstance].delegate = self;
```

7. HEVC Adaptive Downgrade Play

The player supports playing links that contain both HEVC and other video encoding formats, such as H.264. When the player device does not support the HEVC format, it will automatically downgrade to playing videos in the configured alternative encoding format (such as H.264).
Note: Supported from player version 11.7 of the premium version.

```
#import <CoreMedia/CoreMedia.h> //Import header file

NSDictionary *dic = @{
    VOD_KEY_VIDEO_CODEC_TYPE:@(kCMVideoCodecType_HEVC),  // Specify the original HE
    VOD_KEY_BACKUP_URL:@"${backupPlayUrl}"}; // Set the backup playback link addres
[_txVodPlayer setExtentOptionInfo:dic];

// Set the original HEVC playback link
[_txVodPlayer startVodPlay:@"${hevcPlayUrl}"];
```

8. Volume Normalization

The player supports automatically adjusting the volume when playing audio to ensure that the volume of all audio is consistent. This can avoid problems with some audio being too loud or too quiet, providing a better auditory experience. Use `TXVodPlayer#setAudioNormalization` to set the volume normalization, with a loudness range of -70 to 0 (LUFS), and custom values are also supported.
Note: Supported from player version 11.7 of the premium version.

```
/**
 Can be set to preset values (related classes or files: Android: TXVodConstants; iO
 Off: AUDIO_NORMALIZATION_OFF
 On: AUDIO_NORMALIZATION_STANDARD (standard)
     AUDIO_NORMALIZATION_LOW (low)
     AUDIO_NORMALIZATION_HIGH (high)
 Custom values can be set: from low to high, range -70 to 0 LUFS
*/
[_txVodPlayer setAudioNormalization:AUDIO_NORMALIZATION_STANDARD];                //

[_txVodPlayer setAudioNormalization:AUDIO_NORMALIZATION_OFF];
```

9. MP4 video local encryption

After MP4 local encryption is turned on, the player will encrypt and store the data when caching files. Encrypted video files can only be decrypted and played by the player, and cannot be played by third-party players.

Note:

After MP4 local encryption is turned on and playback begins, the encryption settings cannot be changed unless the file is cleared and re-cached.

This feature is supported starting with the Advanced Player 12.2 version.

```
//Precondition: Set the global cache directory of the player. This configuration on
NSString *cachesDir = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSU
NSString downloadPath = [NSString stringWithFormat:@"%@/txdownload",cachesDir];
[TXPlayerGlobalSetting setCacheFolderPath:downloadPath];

TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
_config.encryptedMp4Level = MP4_ENCRYPTION_LEVEL_L2;     // Set to use mp4 local en
[_txVodPlayer setConfig:_config];
```

10. Picture-In-Picture Function

The Player SDK provides basic PIP and Advanced Picture-in-Picture capabilities. Basic PIP supports standard video playback, while Advanced Picture-in-Picture (only for player premium) offers a comprehensive upgrade on this basis, with additional support for encrypted video playback, offline playback, and optimized switch logic to achieve instant foreground-to-PIP switching with no waiting for loading, delivering a smoother experience. For more feature details, refer to the Advanced Picture-in-Picture Usage Guide.

Note:

The advanced picture-in-picture function requires player premium.

Player Event Listening

You can bind a `TXVodPlayListener` listener to the `TXVodPlayer` object to use `onPlayEvent` (event notification) and `onNetStatus` (status feedback) to sync information to your application.

Event notification (onPlayEvent)

Playback events

Event ID	Code	Description
PLAY_EVT_PLAY_BEGIN	2004	Video playback started.
PLAY_EVT_PLAY_PROGRESS	2005	Video playback progress. The current playback progress, loading progress, and total video duration will be notified of.
PLAY_EVT_PLAY_LOADING	2007	The video is being loaded. The

		`LOADING_END` event will be reported if video playback resumes.
PLAY_EVT_VOD_LOADING_END	2014	Video loading ended, and video playback resumed.
VOD_PLAY_EVT_SEEK_COMPLETE	2019	Seeking was completed. The seeking feature is supported by v10.3 or later.
VOD_PLAY_EVT_LOOP_ONCE_COMPLETE	6001	A round of loop was completed. The loop feature is supported by v10.8 or later.
VOD_PLAY_EVT_HIT_CACHE	2002	Cache hit event at startup (supported since version 11.2).
VOD_PLAY_EVT_VIDEO_SEI	2030	Received SEI frame event (supported from player version 11.6 of the premium version).
VOD_PLAY_EVT_HEVC_DOWNGRADE_PLAYBACK	2031	HEVC downgrade playback occurs (supported by the player's advanced version 12.0).
VOD_PLAY_EVT_VOD_PLAY_FIRST_VIDEO_PACKET	2017	The player receives the first frame data packet event (supported by version 12.0).

SEI frame

SEI (Supplemental Enhancement Information) frames are a type of frame used to transmit additional information. The premium version of the player will parse the SEI frames in the video stream and provide callbacks through the `VOD_PLAY_EVT_VIDEO_SEI` event.

Note: Supported from player version 11.6 of the premium version.

```
-(void) onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary*
    if (EvtID == VOD_PLAY_EVT_VIDEO_SEI) {
        int seiType = [param objectForKey:EVT_KEY_SEI_TYPE]; //  the type of video
        int seiSize = [param objectForKey:EVT_KEY_SEI_SIZE]; // the data size of vi
        NSData *seiData = [param objectForKey:EVT_KEY_SEI_DATA]; // the byte array
    }
}
```

Warning events

You can ignore the following events, which are only used to notify you of some internal events of the SDK.

Event ID	Code	Description
PLAY_WARNING_VIDEO_DECODE_FAIL	2101	Failed to decode the current video frame.
PLAY_WARNING_AUDIO_DECODE_FAIL	2102	Failed to decode the current audio frame.
PLAY_WARNING_RECONNECT	2103	The network was disconnected, and automatic reconnection was performed (the `PLAY_ERR_NET_DISCONNECT` event will be thrown after three failed attempts).
PLAY_WARNING_HW_ACCELERATION_FAIL	2106	Failed to start the hardware decoder, and the software decoder was used instead.

Connection events

The following server connection events are mainly used to measure and collect the server connection time:

Event ID	Code	Description
PLAY_EVT_VOD_PLAY_PREPARED	2013	The player has been prepared and can start playback. If `autoPlay` is set to `false`, you need to call `resume` after receiving this event to start playback.
PLAY_EVT_RCV_FIRST_I_FRAME	2003	The network received the first renderable video data packet (IDR).

Image quality events

The following events are used to get image change information:

Event ID	Code	Description
PLAY_EVT_CHANGE_RESOLUTION	2009	The video resolution changed.
PLAY_EVT_CHANGE_ROATION	2011	The MP4 video was rotated.

Video information events

Event ID	Code	Description
PLAY_EVT_GET_PLAYINFO_SUCC	2010	Obtained the information of the file played back successfully.

If you play back a video through `fileId` and the playback request succeeds, the SDK will notify the upper layer of some request information, and you can parse `param` to get the video information after receiving the

`PLAY_EVT_GET_PLAYINFO_SUCC` event.

Video Information	Description
EVT_PLAY_COVER_URL	Video thumbnail URL
EVT_PLAY_URL	Video playback URL
EVT_PLAY_DURATION	Video duration
EVT_KEY_WATER_MARK_TEXT	Ghost watermark text content (supported from version 11.6).

```
-(void) onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary*
{
    if (EvtID == PLAY_EVT_VOD_PLAY_PREPARED) {
        // The player preparation completion event is received, and you can call th
    } else if (EvtID == PLAY_EVT_PLAY_BEGIN) {
        // The playback start event is received
    } else if (EvtID == PLAY_EVT_PLAY_END) {
        // The playback end event is received
    }
}
```

Ghost watermark

The content of the ghost watermark is filled in the player signature and is ultimately displayed on the playback end through collaboration between the cloud and the player, ensuring the security of the watermark throughout the transmission process. Follow the tutorial to configure the ghost watermark in the player signature. The content of the ghost watermark can be obtained through `[param objectForKey:@"EVT_KEY_WATER_MARK_TEXT"]` after receiving the `VOD_PLAY_EVT_GET_PLAYINFO_SUCC` event from the player. For detailed usage tutorial, please refer to SuperPlayer Component > Ghost Watermark.
Note: Supported from player version 11.6.

Playback error event

Note:

[-6004 , -6010] Error events are supported since version 11.0.

Event ID	Value	Meaning
PLAY_ERR_NET_DISCONNECT	-2301	Video data errors that cannot be recovered by retrying the playback. For example, network anomalies or download data errors that cause demuxing timeouts or failures.

PLAY_ERR_HLS_KEY	-2305	HLS decryption key retrieval failure.
VOD_PLAY_ERR_SYSTEM_PLAY_FAIL	-6004	System player playback error.
VOD_PLAY_ERR_DECODE_VIDEO_FAIL	-6006	Video decoding error or unsupported video format.
VOD_PLAY_ERR_DECODE_AUDIO_FAIL	-6007	Audio decoding error or unsupported audio format.
VOD_PLAY_ERR_DECODE_SUBTITLE_FAIL	-6008	Subtitle decoding error.
VOD_PLAY_ERR_RENDER_FAIL	-6009	Video rendering error.
VOD_PLAY_ERR_PROCESS_VIDEO_FAIL	-6010	Video post-processing error.
VOD_PLAY_ERR_GET_PLAYINFO_FAIL	-2306	Failed to obtain the on-demand file information. It is recommended to check whether the AppId, FileId or Psign is filled in correctly.

Status feedback (onNetStatus)

The status feedback is triggered once every 0.5 seconds to provide real-time feedback on the current status of the pusher. It can act as a dashboard to inform you of what is happening inside the SDK so you can better understand the current video playback status.

Parameter	Description
CPU_USAGE	Current instantaneous CPU utilization
VIDEO_WIDTH	Video resolution - width
VIDEO_HEIGHT	Video resolution - height
NET_SPEED	Current network data reception speed in KBps
VIDEO_FPS	Current video frame rate of streaming media
VIDEO_BITRATE	Current video bitrate in bps of streaming media
AUDIO_BITRATE	Current audio bitrate in bps of streaming media
V_SUM_CACHE_SIZE	Buffer (`jitterbuffer`) size. If the current buffer length is 0, lag will occur soon.
SERVER_IP	Connected server IP

Below is the sample code of using `onNetStatus` to get the video playback information:

```
- (void)onNetStatus:(TXVodPlayer *)player withParam:(NSDictionary *)param {
    // Get the current CPU utilization
    float cpuUsage = [[param objectForKey:@"CPU_USAGE"] floatValue];
    // Get the video width
    int videoWidth = [[param objectForKey:@"VIDEO_WIDTH"] intValue];
    // Get the video height
    int videoHeight = [[param objectForKey:@"VIDEO_HEIGHT"] intValue];
    // Get the real-time speed
    int  speed = [[param objectForKey:@"NET_SPEED"] intValue];
    // Get the current video frame rate of streaming media
    int fps = [[param objectForKey:@"VIDEO_FPS"] intValue];
    // Get the current video bitrate in Kbps of streaming media
    int videoBitRate = [[param objectForKey:@"VIDEO_BITRATE"] intValue];
    // Get the current audio bitrate in Kbps of streaming media
    int audioBitRate = [[param objectForKey:@"AUDIO_BITRATE"] intValue];
    // Get the buffer (`jitterbuffer`) size. If the current buffer length is 0, lag
    int jitterbuffer = [[param objectForKey:@"V_SUM_CACHE_SIZE"] intValue];
    // Get the connected server IP
    NSString *ip = [param objectForKey:@"SERVER_IP"];
}
```

Other functions

HLS live video source playback

The premium version of the player supports playing HLS live video sources. Starting from version 11.8, it supports live video sources with HLS EVENT. Usage is as follows:

```
TXVodPlayConfig *_config = [[TXVodPlayConfig alloc]init];
[_config setMediaType:MEDIA_TYPE_HLS_LIVE];   // Specify the HLS live media type
[_txVodPlayer setConfig:_config];
[_txVodPlayer startVodPlay:${YOUR_HSL_LIVE_URL}];
```

Scenario-Specific Features

1. Dynamically Setting AudioSession

Sometimes it is necessary to dynamically set the audio output mode based on the scene, especially for iPhones, which natively support multiple audio playback and background modes. Therefore, we support the following three main modes based on the user's scenario:

AVAudioSessionCategoryPlayback: Exclusive playback in the background.
AVAudioSessionCategoryPlayAndRecord: Exclusive playback in the background.
AVAudioSessionCategoryAmbient: Mixed playback.
You can use the above modes to set the Category and Option of AudioSession according to the current scenario to achieve your purpose. The following are two examples of settings for different scenarios (the settings below can be dynamically adjusted and set according to your own scenario):
Scenario 1: Playlist scenario (video playback needs to support silent playback in the playlist and does not interrupt external audio playback).

```
[[AVAudioSession sharedInstance] setCategory:AVAudioSessionCategoryPlayback withOpt
[[AVAudioSession sharedInstance] setActive:YES error:nil];
```

Scenario 2: Playback details scenario (the video details have sound, and temporarily interrupt external audio. After the video playback is complete, the external audio will be resumed).

```
[[AVAudioSession sharedInstance] setCategory:AVAudioSessionCategoryAmbient withOpti
[[AVAudioSession sharedInstance] setActive:NO withOptions:AVAudioSessionSetActiveOp
```

2. SDK-based demo component

Based on the Player SDK, Tencent Cloud has developed a player component. 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 popular video apps.

3. Open-source GitHub projects

Based on the Player SDK, Tencent Cloud has developed immersive video player, video feed stream, and multi-layer reuse components and will provide more user scenario-based components on future versions. You can download the Player for iOS to try out the components.

Android Integration
Integration Guide

Last updated：2024-04-26 11:09:31

This document describes how to quickly integrate RT-Cube's player SDK into your project. Different versions of the SDK can be integrated in the same way.

Environment Requirements

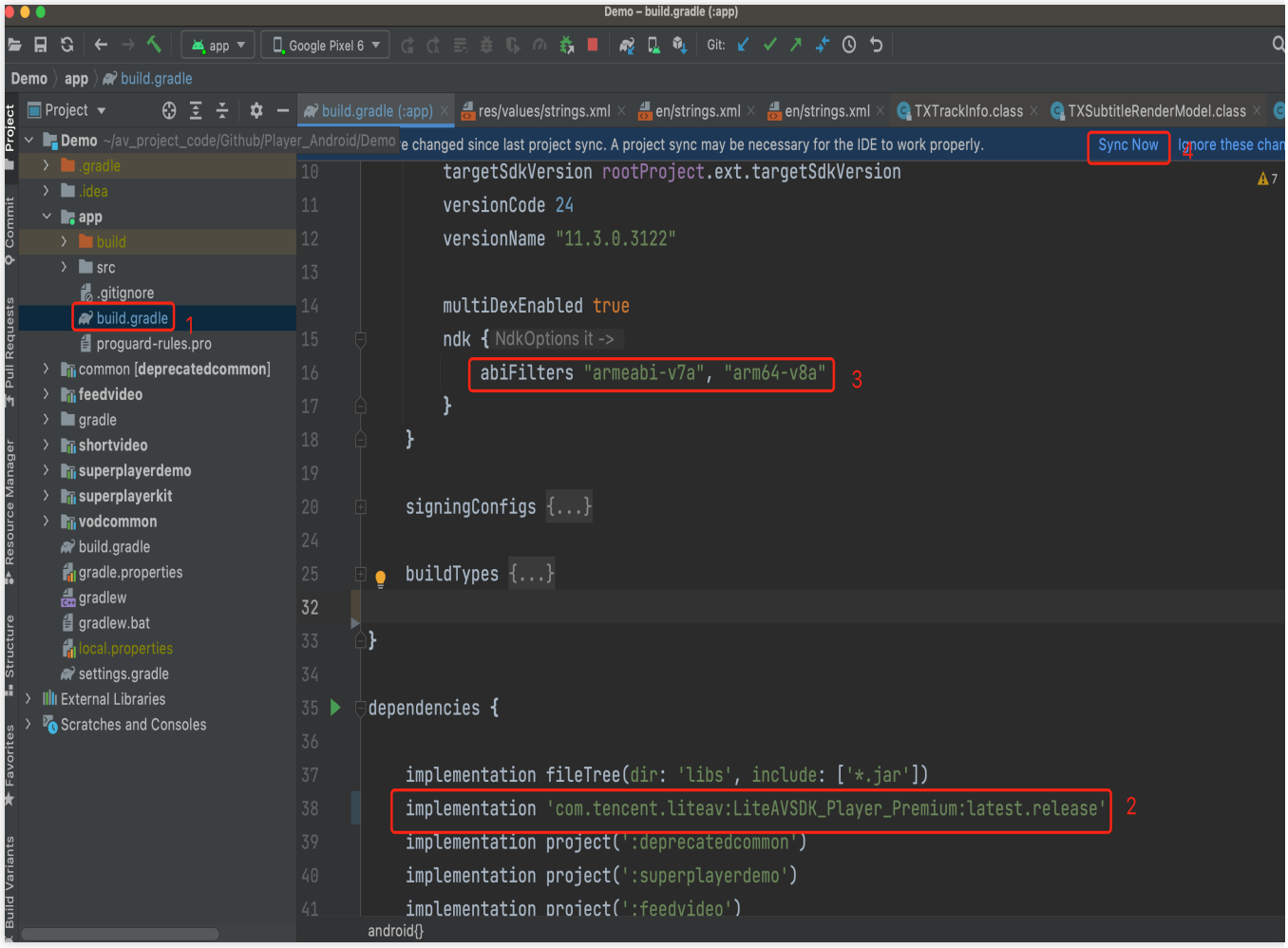
Android Studio 2.0 or above
Android 4.1 (SDK API level 16) or above

Integrating the SDK (AAR)

You can use Gradle to automatically load the AAR file or manually download the AAR file and import it into your project.

Method 1: automatic loading (AAR)

The player SDK has been released to the mavenCentral repository, and you can configure it in Gradle to download `LiteAVSDK_Player_Premium` updates automatically.
Open your project with Android Studio and modify the `build.gradle` file as described below to complete the integration.



1. Add the `mavenCentral` repository to the `build.gradle` in your project's root directory.

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

2. Open the `build.gradle` in the `app` directory and add the `LiteAVSDK_Player` dependencies to `dependencies`.

```
dependencies {
  // This configuration integrates the latest version of
`LiteAVSDK_Player_Premium` by default.
 implementation 'com.tencent.liteav:LiteAVSDK_Player_Premium:latest.release'
  // To integrate an earlier version such as 10.8.0.29000, configure as
follows:
  // implementation 'com.tencent.liteav:LiteAVSDK_Player_Premium:10.8.0.29000'
}
```

3. In `defaultConfig`, specify the CPU architecture to be used by the application. Currently, LiteAVSDK_Player supports armeabi, armeabi-v7a, and arm64-v8a.

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

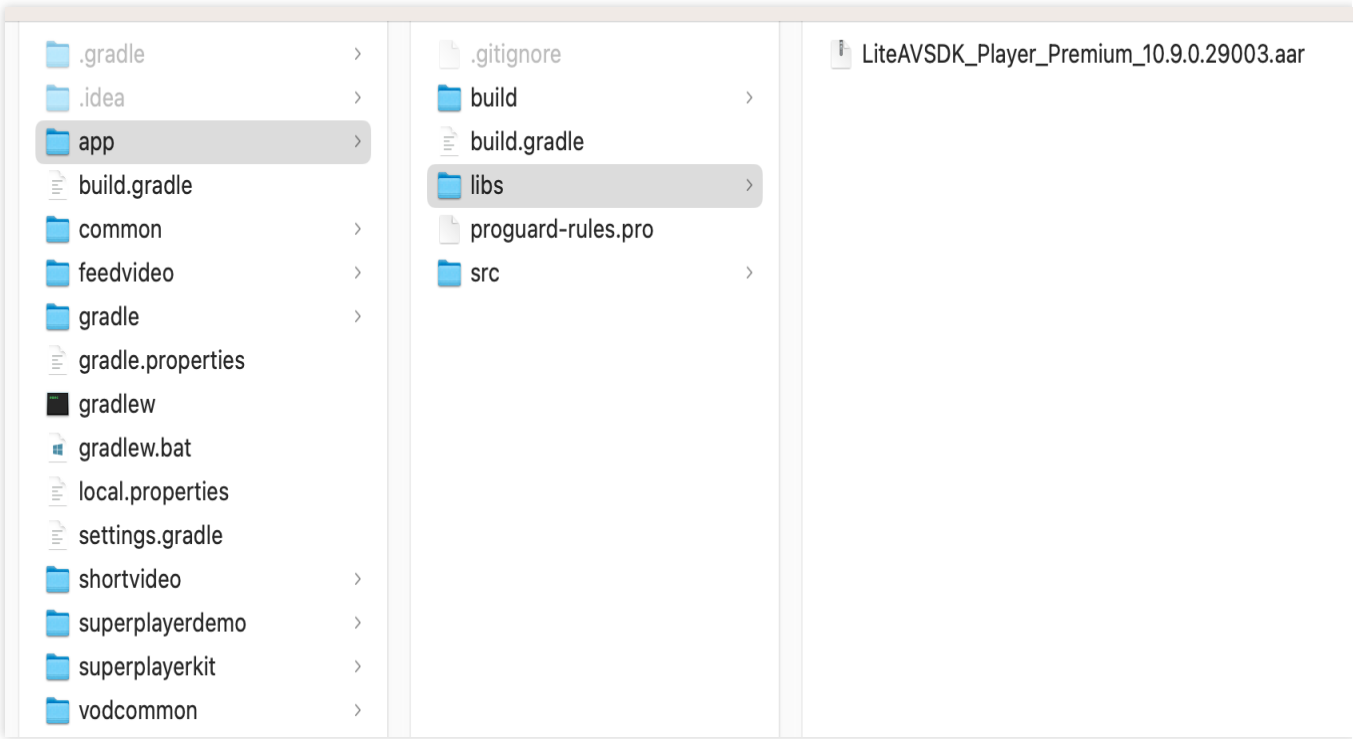
4. Click the **Sync Now** button

to sync the SDK. If you have no problem accessing Maven Central, the SDK will be downloaded and integrated into your project automatically.

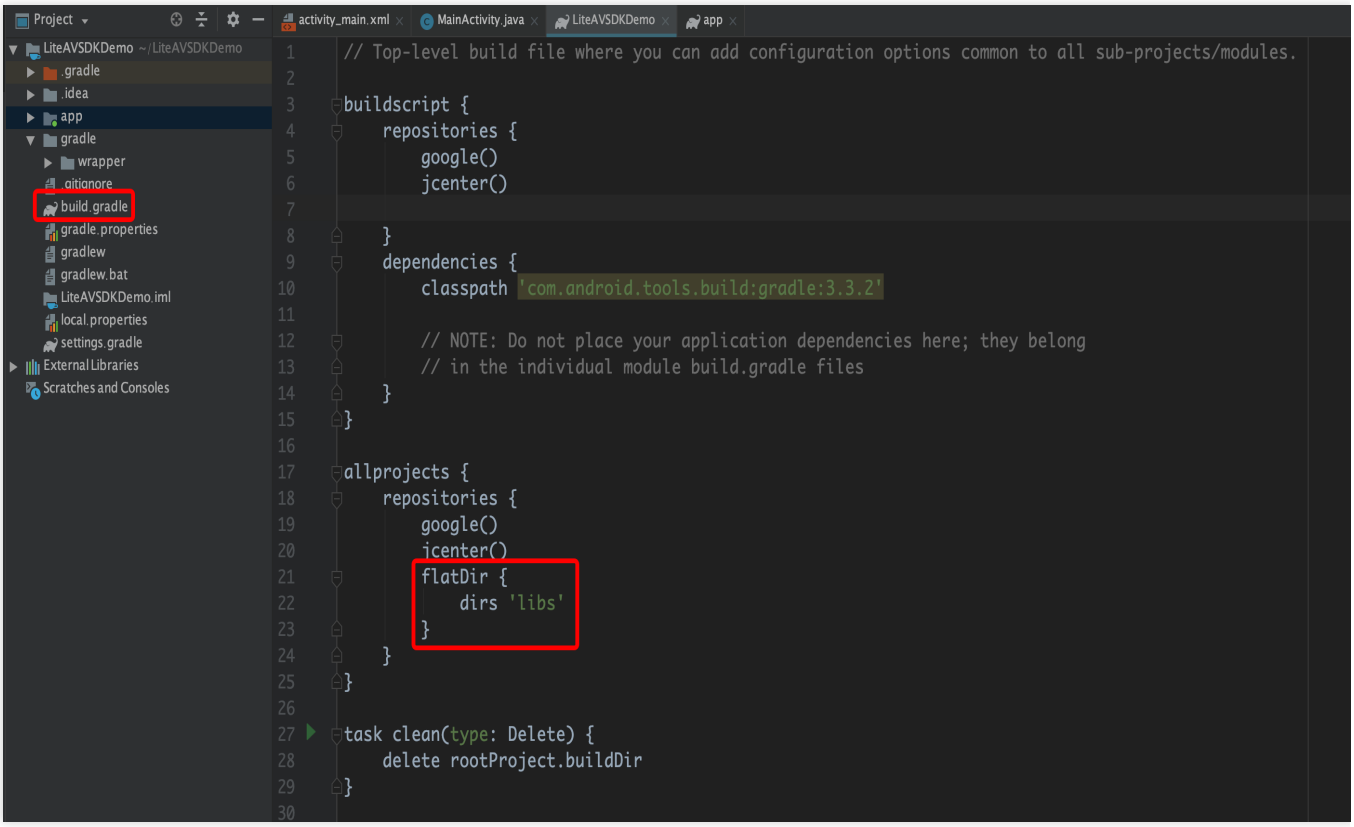
Method 2: manual download (AAR)

If you have problem accessing Maven Central, you can manually download the SDK and integrate it into your project.

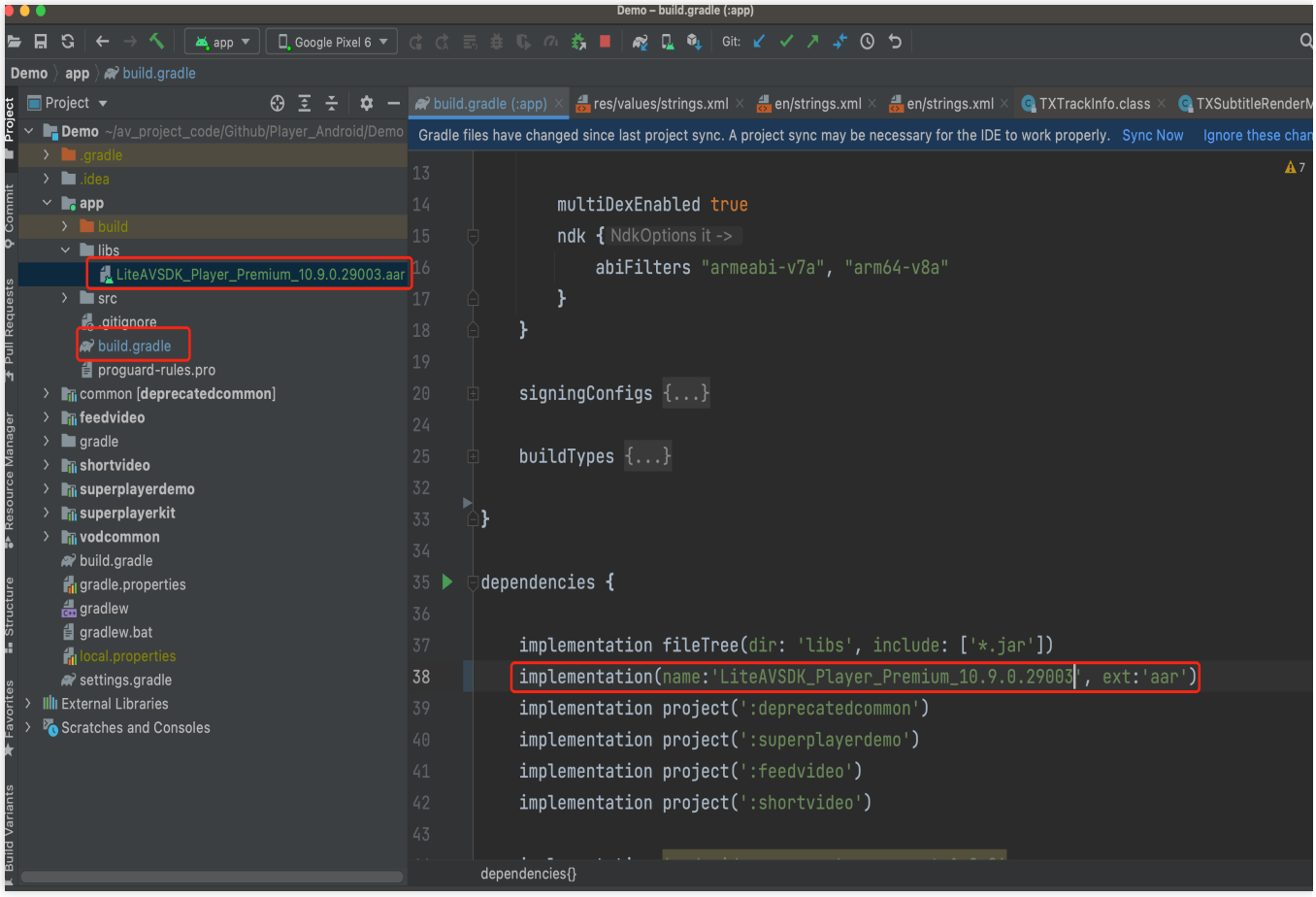
1. Download LiteAVSDK_Player_Premium and decompress the file.
2. Copy the AAR file in the SDK directory to the **app**/**libs** directory of your project.



3. Add **flatDir** to `build.gradle` under your project's root directory to specify the local path for the repository.



4. Add the LiteAVSDK_Player_Premium dependency and then add code that references the AAR file in `app/build.gradle`.




```
implementation(name:'LiteAVSDK_Player_Premium_10.9.0.29003', ext:'aar')
```

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

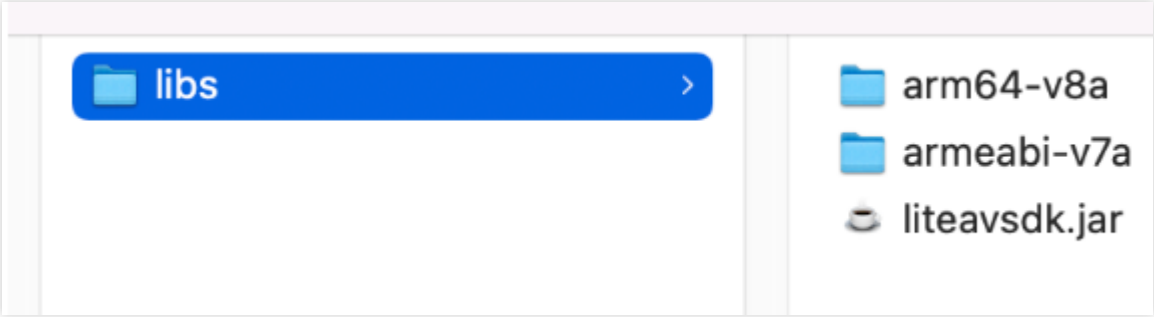
```
defaultConfig {
 ndk {
     abiFilters "armeabi", "armeabi-v7a", "arm64-v8a"
 }
}
```

6. Click **Sync Now** to complete the integration of LiteAVSDK.

Integrating the SDK (JAR)

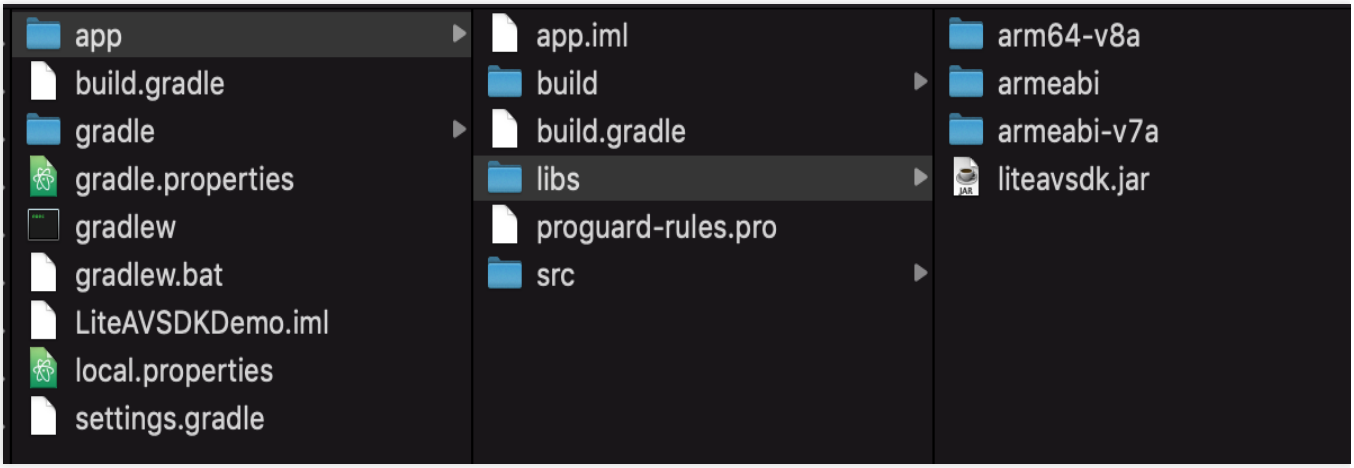
If you do not want to import the AAR library, you can also integrate LiteAVSDK by importing JAR and SO libraries.

1. Download LiteAVSDK_Player_Premium and decompress it. Find `LiteAVSDK_Player_Premium_xxx.zip` (`xxx` is the version number) in the SDK directory. After decompression, you can get the `libs` directory, which contains the JAR file and folders of SO files as listed below:

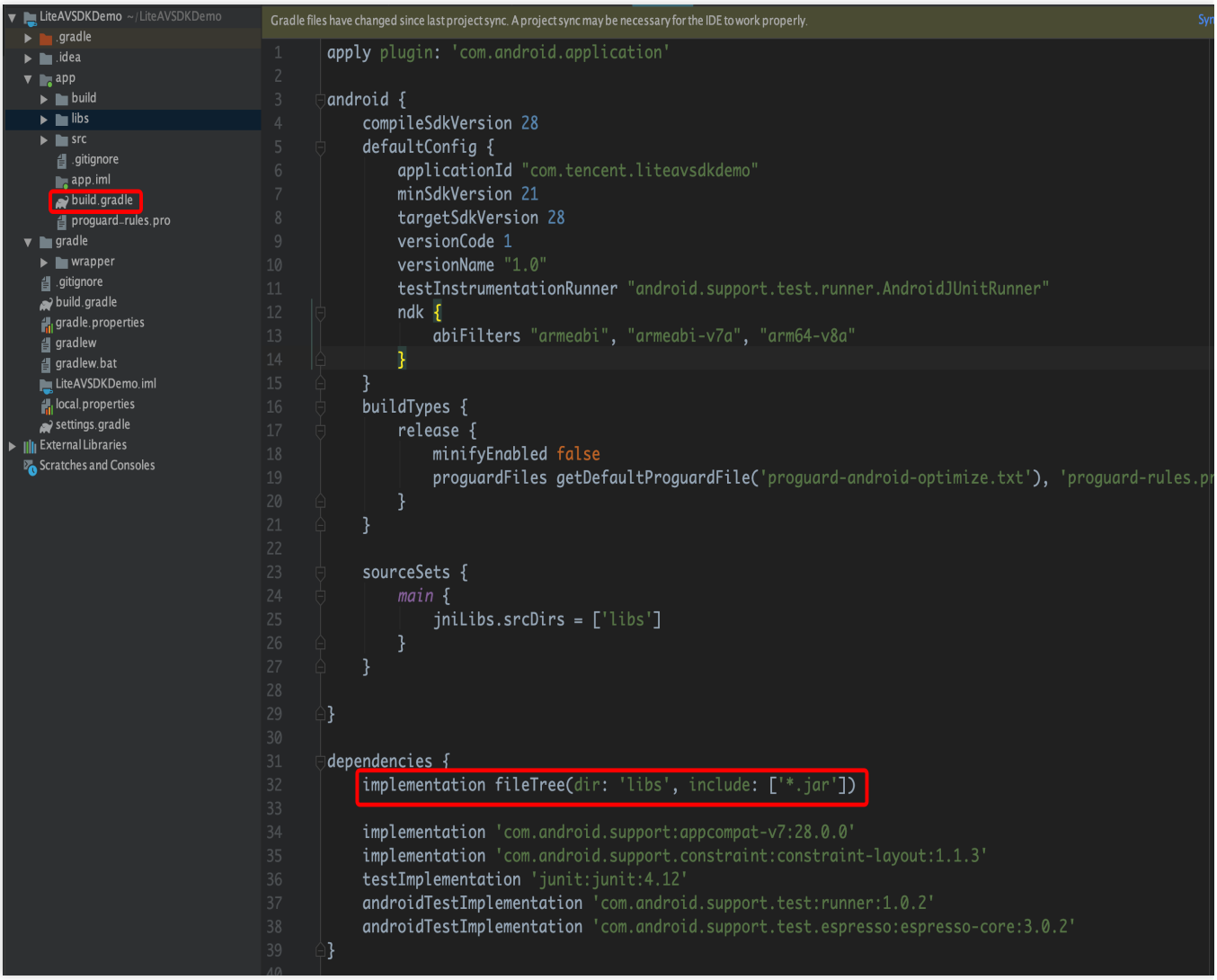


If you also need the .so file for the armeabi architecture, copy the `armeabi-v7a` directory and rename it `armeabi`.

2. Copy the JAR file and `armeabi`, `armeabi-v7a`, and `arm64-v8a` folders to the `app/libs` directory.

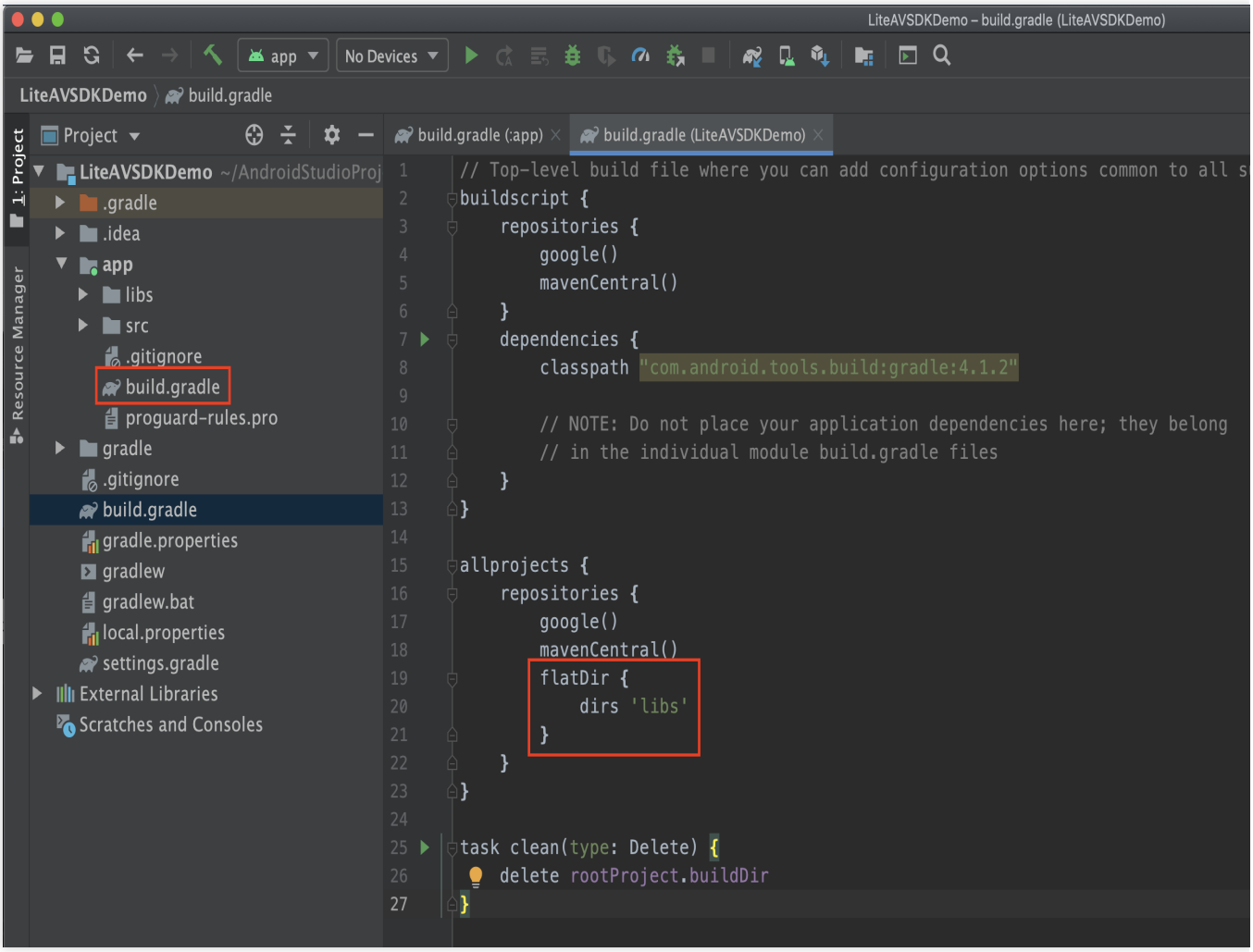


3. Add code that references the JAR library in `app/build.gradle`.

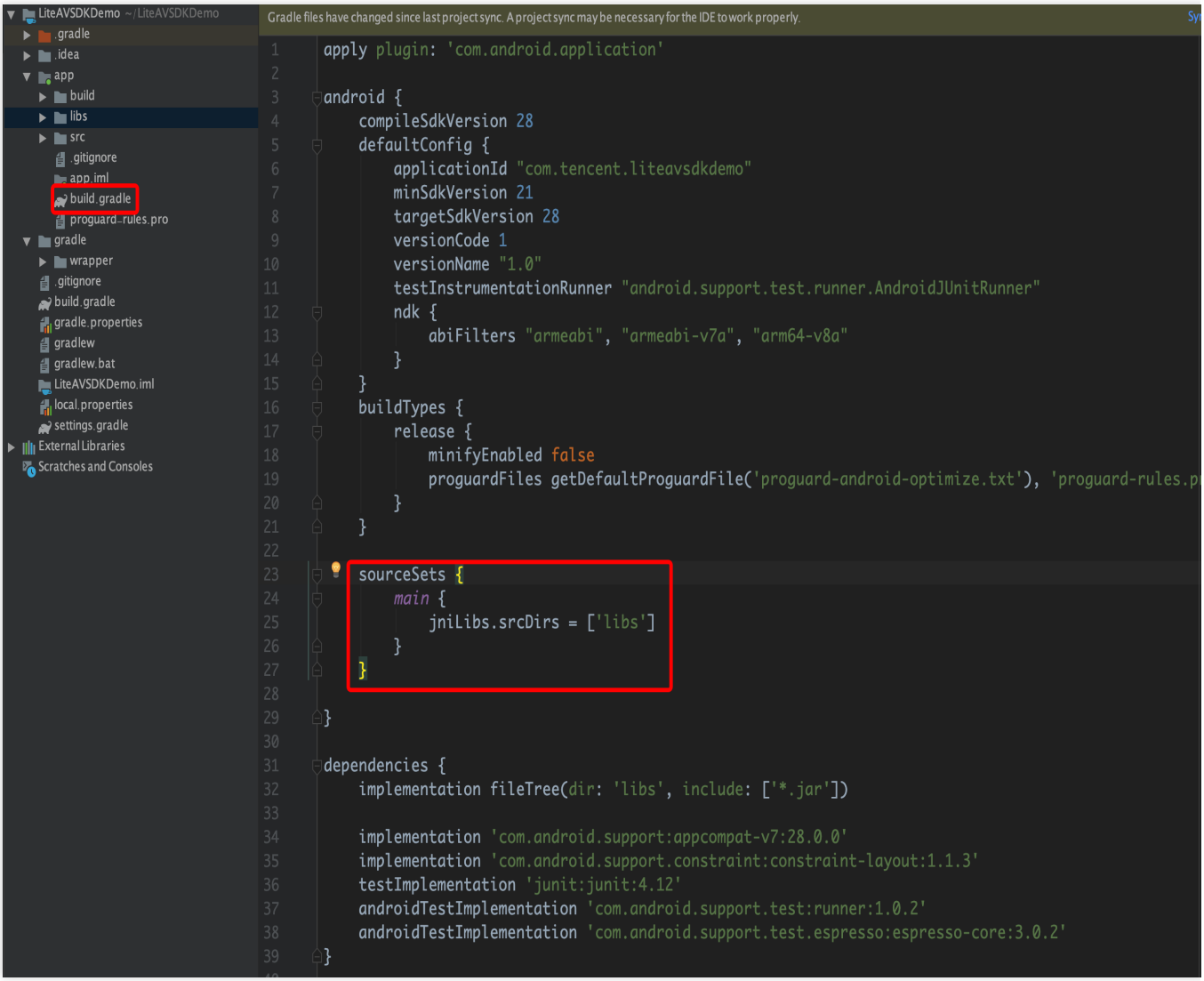



```
dependencies {
 implementation fileTree(dir:'libs',include:['*.jar'])
}
```

4. Add **flatDir** to `build.gradle` under the project's root directory to specify the local path for the repository.



5. In `app/build.gradle` , add code that references the SO libraries.



6. 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).

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

7. Click **Sync Now** to complete the integration.

Configuring 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" />
<!--storage-->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Network security configuration allows the app to send HTTP requests

For security reasons, starting from Android P, Google requires that requests from apps use encrypted connections. The player SDK will start a localsever to proxy HTTP requests. If your app's targetSdkVersion is greater than or equal to 28, you can enable sending HTTP requests to 127.0.0.1 through network security configuration. Otherwise, an error 'java.io.IOException: Cleartext HTTP traffic to 127.0.0.1 not permitted' will occur during playback, causing video playback to fail. The configuration steps are as follows:

1. Create a res/xml/network_security_config.xml file in the project and set the network security configuration.

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">127.0.0.1</domain>
    </domain-config>
</network-security-config>
```

2. Add the following attribute to the application tag in the AndroidManifest.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:networkSecurityConfig="@xml/network_security_config"
                    ... >
        ...
    </application>
</manifest>
```

Configuring Obfuscation Rules

In the `proguard-rules.pro` file, add LiteAVSDK classes to the "do not obfuscate" list.

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

Configuring License

Enter VOD console. to apply for a trial license as instructed in Adding and Renewing a License. If no license is configured, video playback will fail. You will get two strings: a license URL and a decryption key.
After obtaining the License information, you need to initialize and configure the License before calling the relevant interfaces of the SDK. For detailed tutorials, please see Configuring View License.

FAQs

What should I do if a duplicate symbol error occurs because my project integrates multiple editions of LiteAVSDK such as CSS, TRTC, and Player?

If you integrate two or more editions of LiteAVSDK (MLVB, Player, TRTC, UGSV), a library conflict error will occur when you build your project. This is because some symbol files are shared among the underlying libraries of the SDKs. To solve the problem, we recommend you integrate the All-in-One SDK, which includes the features of MLVB, Player, TRTC, and UGSV. For details, see SDK Download.

VOD Scenario

Last updated：2025-04-02 14:50:12

Limits

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.

This Document Describes

How to integrate the Tencent Cloud Player SDK for Android.
How to use the Player SDK for VOD playback.
How to use the underlying capabilities of the Player SDK to implement more features.

SDK Integration

Step 1. Integrate the SDK ZIP file

Download and integrate the SDK ZIP file as instructed in Integration Guide.

Step 2. Configure the license

If you have obtained a license, you can view the license URL and key in the VOD console.
If you don't have the required license yet, you can get it as instructed in Adding and Renewing a License.
After obtaining the License information, you need to initialize and configure the License before calling the relevant interfaces of the SDK. For detailed tutorials, please see Configuring View License.

Step 3. Add a view

The SDK provides `TXCloudVideoView` for video rendering by default. First, add the following code to the layout XML file:

```
<com.tencent.rtmp.ui.TXCloudVideoView
            android:id="@+id/video_view"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_centerInParent="true"
```

```
            android:visibility="gone"/>
```

Step 4. Create a player object

Create the **TXVodPlayer** object and use the `setPlayerView` API to associate the object with the **video_view** control just added to the UI.

```
// `mPlayerView` is the video rendering view added in step 3
TXCloudVideoView mPlayerView = findViewById(R.id.video_view);
// Create a player object
TXVodPlayer mVodPlayer = new TXVodPlayer(getActivity());
// Associate the player object with the video rendering view
mVodPlayer.setPlayerView(mPlayerView);
```

Step 5. Start playback

`TXVodPlayer` supports two playback modes for you to choose as needed:

Through URL

Through FileId

`TXVodPlayer` will internally recognize the playback protocol automatically. You only need to pass in your playback URL to the `startVodPlay` function.

```
// Play back a video resource at a URL
String url = "http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
mVodPlayer.startVodPlay(url);


// Play back a local video resource
String localFile = "/sdcard/video.mp4";
mVodPlayer.startVodPlay(localFile);

// Starting from version 11.8, the player supports content:// URI video resources a
// Play content:// URI video resources
String localFile = "content://xxx/xxx/video.mp4";
mVodPlayer.startVodPlay(localFile);

// Play asset catalog video resources, the passed address must start with asset://
String localFile = "asset://video.mp4";
mVodPlayer.startVodPlay(localFile
```

```
// We recommend you use the following new API:
// `psign` is a player signature. For more information on the signature and how to
TXPlayInfoParams playInfoParam = new TXPlayInfoParams(1252463788, // `appId` of the
    "4564972819220421305", // `fileId` of the video
```

```
    "psignxxxxxxx"); // The player signature
mVodPlayer.startVodPlay(playInfoParam);

// Legacy API, which is not recommended
TXPlayerAuthBuilder authBuilder = new TXPlayerAuthBuilder();
authBuilder.setAppId(1252463788);
authBuilder.setFileId("4564972819220421305");
mVodPlayer.startVodPlay(authBuilder);
```

Find the target video file in Media Assets, and you can view the `FileId` below the filename.

Play back the video through the `FileId`, and the player will request the backend for the real playback URL. If the network is abnormal or the `FileId` doesn't exist, the `TXLiveConstants.PLAY_ERR_GET_PLAYINFO_FAIL` event will be received; otherwise, `TXLiveConstants.PLAY_EVT_GET_PLAYINFO_SUCC` will be received, indicating that the request succeeded.

Step 6. Stop playback

Remember to terminate the view control when stopping the playback, especially before the next call of `startVodPlay`. This can prevent memory leak and screen flashing issues.

In addition, when exiting the playback UI, you need to call the `onDestroy()` function for the rendering view. This can prevent memory leak and "Receiver not registered" alarms.

```
@Override
public void onDestroy() {
    super.onDestroy();
    mVodPlayer.stopPlay(true); // `true` indicates to clear the last-frame image
    mPlayerView.onDestroy();
}
```

Note:

The boolean parameter of `stopPlay` indicates whether to clear the last-frame image. Early versions of the live player of the RTMP SDK don't have an official pause feature; therefore, this boolean value is used to clear the last-frame image.

If you want to retain the last-frame image after VOD stops, simply do nothing after receiving the playback stop event; playback will stop at the last frame by default.

Basic Feature Usage

1. Playback control

Starting playback

```
// Start playback
mVodPlayer.startVodPlay(url)
```

Pausing playback

```
// Pause the video
mVodPlayer.pause();
```

Resuming playback

```
// Resume the video
mVodPlayer.resume();
```

Stopping playback

```
// Stop the video
mVodPlayer.stopPlay(true);
```

Adjusting playback progress (seek)

When the user drags the progress bar, `seek` can be called to start playback at the specified position. The Player SDK supports accurate seek.

```
int time = 600; // In seconds if the value is of `int` type
// float time = 600; // In seconds if the value is of `float` type
// Adjust the playback progress
mVodPlayer.seek(time);
```

Precise and Imprecise Seek

Starting from version 11.8 of the player SDK, it is supported to specify precise or imprecise seek when calling the seek interface.

```
float time = 600; // Unit is seconds for float type
// Adjust progress
mVodPlayer.seek(time, true); // Precise seek
mVodPlayer.seek(time, false); // Imprecise seek
```

Seek to the specified Program Date Time (PDT) point in the video stream

To seek to the specified Program Date Time (PDT) point in the video stream, which enables functions such as fast-forward, rewind, and progress bar jumping, currently only HLS video format is supported.

Note:

Starting from version 11.6 of the player's premium edition, this function is supported.

```
long pdtTimeMs = 600; // Unit is milliseconds
mVodPlayer.seekToPdtTime(time);
```

Specifying playback start time

You can specify the playback start time before calling `startVodPlay` for the first time.

```
float startTimeInSecond = 60; // Unit: Second
mVodPlayer.setStartTime(startTimeInSecond);   // Set the playback start time
mVodPlayer.startVodPlay(url);
```

2. Image adjustment

view: size and position

You can modify the size and position of video images by adjusting the size and position of the `video_view` control added in the Add a view step during SDK integration.

setRenderMode: Aspect fill or aspect fit

Value	Definition
RENDER_MODE_FULL_FILL_SCREEN	Images are scaled to fill the entire screen, and the excess parts are cropped. There are no black bars in this mode, but images may not be displayed entirely.
RENDER_MODE_ADJUST_RESOLUTION	Images are scaled so that the long side of the video fits the screen. Neither side exceeds the screen after scaling. Images are centered, and there may be black bars visible.

setRenderRotation: Image rotation

Value	Definition
RENDER_ROTATION_PORTRAIT	Normal playback (the Home button is below the video image)
RENDER_ROTATION_LANDSCAPE	Clockwise rotation of the image by 270 degrees (the Home button is on the left of the video image)

```
 // Fill the screen at the original aspect ratio
mVodPlayer.setRenderMode(TXLiveConstants.RENDER_MODE_FULL_FILL_SCREEN);
// Normal playback (the Home button is below the video image)
mVodPlayer.setRenderRotation(TXLiveConstants.RENDER_ROTATION_PORTRAIT);
```

3. Adjustable-Speed playback

The VOD player supports adjustable-speed playback. You can use the `setRate` API to set the VOD playback speed, such as 0.5x, 1.0x, 1.2x, and 2x speed.

```
// Set playback at 1.2X rate
mVodPlayer.setRate(1.2);
```

4. Playback loop

```
// Set playback loop
mVodPlayer.setLoop(true);
// Get the current playback loop status
mVodPlayer.isLoop();
```

5. Muting/Unmuting

```
// Mute or unmute the player. true: Mute; false: Unmute
mVodPlayer.setMute(true);
```

6. Screencapturing

Call **snapshot** to take a screenshot of the current video frame. This method captures only the video frame. To capture the UI, use the corresponding API of the Android system.

```
// Take a screenshot
mVodPlayer.snapshot(new ITXSnapshotListener() {
    @Override
    public void onSnapshot(Bitmap bmp) {
        if (null != bmp) {
           // Get the screenshot bitmap
        }
    }
});
```

7. Roll image ad

The Player SDK allows you to add roll images on the UI for advertising as follows:
If `autoPlay` is set to `NO` , the player will load the video normally but will not immediately start playing it back. Users can see the roll image ad on the player UI after the player is loaded and before the video playback starts. When the ad display stop conditions are met, the `resume` API will be called to start video playback.

```
mVodPlayer.setAutoPlay(false);  // Set manual playback
```

```
mVodPlayer.startVodPlay(url);    // The video will be loaded after `startVodPlay` i
// ......
// Display the ad on the player UI
// ......
mVodPlayer.resume();  // Call `resume` to start playing back the video after the ad
```

8. HTTP-REF

`headers` in `TXVodPlayConfig` can be used to set HTTP request headers, such as the `Referer` field commonly used to prevent the URL from being copied arbitrarily (Tencent Cloud provides a more secure signature-based hotlink protection solution) and the `Cookie` field for client authentication

```
TXVodPlayConfig mPlayConfig = new TXVodPlayConfig();
Map<String, String> headers = new HashMap<>();
headers.put("Referer", "${Refer Content}");
mPlayConfig.setHeaders(headers);
mVodPlayer.setConfig(mPlayConfig);
```

9. Hardware acceleration

It is extremely difficult to play back videos of the Blu-ray (1080p) or higher image quality smoothly if only software decoding is used. Therefore, if your main scenario is game live streaming, we recommend you use hardware acceleration.

Before switching between software and hardware decoding, you need to call **stopPlay** first. After the switch, you need to call **startVodPlay**; otherwise, severe blurs will occur.

```
 mVodPlayer.stopPlay(true);
 mVodPlayer.enableHardwareDecode(true);
 mVodPlayer.startVodPlay(flvUrl, type);
```

10. Definition settings

The SDK supports the multi-bitrate format of HLS, so users can switch between streams at different bitrates to switch the video definition. You can set the definition as follows:

```
// Get the array of multiple bitrates. The TXBitrateItem class fields mean: index -
ArrayList<TXBitrateItem> bitrates = mVodPlayer.getSupportedBitrates();
int index = bitrates.get(i).index; // Specify the bitrate index to be played
mVodPlayer.setBitrateIndex(index); // Switch the bitrate to the desired clarity

// Get the index of the currently played bitrate. The return value of -1000 is the
int index = mVodPlayer.getBitrateIndex();
```

During playback, you can call `mVodPlayer.setBitrateIndex(int)` at any time to switch the bitrate. During the switch, the data of another stream will be pulled. The SDK is optimized for Tencent Cloud multi-bitrate files to implement smooth switching.
If you know the resolution information of the video stream in advance, you can specify the video resolution to be played before starting playback to avoid switching streams after playback. For detailed methods, refer to Player Configuration#Specifying Resolution Before Playback.

11. Adaptive bitrate streaming

The SDK supports adaptive bitrate streaming of HLS. After this capability is enabled, the player can dynamically select the most appropriate bitrate for playback based on the current bandwidth. You can enable adaptive bitrate streaming as follows:

```
mVodPlayer.setBitrateIndex(-1); // Pass in `-1` for the `index` parameter
```

During playback, you can call `mVodPlayer.setBitrateIndex(int)` at any time to switch to another bitrate. After the switch, adaptive bitrate streaming will be disabled.

12. Enabling smooth bitrate switch

Before starting playback, you can enable smooth bitrate switch to seamlessly switch between different definitions (bitrates) during playback. If smooth bitrate switch is enabled, the transition between different bitrates will be smoother but will be more time-consuming. Therefore, this feature can be configured as needed.

```
TXVodPlayConfig mPlayConfig = new TXVodPlayConfig();
// If it is set to `true`, the bitrate can be switched smoothly when IDR frames are
mPlayConfig.setSmoothSwitchBitrate(true);
mVodPlayer.setConfig(mPlayConfig);
```

13. Playback progress listening

There are two metrics for the VOD progress: **loading progress** and **playback progress**. Currently, the SDK notifies the two progress metrics in real time through event notifications. For more information on the event notification content, see Event Listening.
You can bind a **TXVodPlayerListener** listener to the `TXVodPlayer` object, and the progress notification will be called back to your application through the **PLAY_EVT_PLAY_PROGRESS** event. The event information contains the above two progress metrics.

```
mVodPlayer.setVodListener(new ITXVodPlayListener() {
    @Override
    public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
        if (event == TXLiveConstants.PLAY_EVT_PLAY_PROGRESS) {
            // Loading progress in ms
            int playable_duration_ms = param.getInt(TXLiveConstants.EVT_PLAYABLE_DU
```

```
            mLoadBar.setProgress(playable_duration_ms); // Set the loading progress

            // Playback progress in ms
            int progress_ms = param.getInt(TXLiveConstants.EVT_PLAY_PROGRESS_MS);
            mSeekBar.setProgress(progress_ms);  // Set the playback progress bar

            // Total video duration in ms
            int duration_ms = param.getInt(TXLiveConstants.EVT_PLAY_DURATION_MS);
            // It can be used to set duration display, etc.

            // Get the PDT time. This function is supported starting from version 1
            long pdt_time_ms = param.getLong(TXVodConstants.EVT_PLAY_PDT_TIME_MS);
        }
    }

    @Override
    public void onNetStatus(TXVodPlayer player, Bundle bundle) {
    }
});
```

14. Playback network speed listening

You can display the current network speed when the video is lagging by listening on events.

You can use the `NET_STATUS_NET_SPEED` of `onNetStatus` to get the current network speed. For detailed directions, see Playback status feedback (onNetStatus).

After the `PLAY_EVT_PLAY_LOADING` event is detected, the current network speed will be displayed.

After the `PLAY_EVT_VOD_LOADING_END` event is received, the view showing the current network speed will be hidden.

```
mVodPlayer.setVodListener(new ITXVodPlayListener() {
    @Override
    public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
        if (event == TXLiveConstants.PLAY_EVT_PLAY_LOADING) {
            // Show the current network speed

        } else if (event == TXLiveConstants.PLAY_EVT_VOD_LOADING_END) {
            // Hide the view showing the current network speed
        }
    }

    @Override
    public void onNetStatus(TXVodPlayer player, Bundle bundle) {
        // Get the real-time speed in Kbps
        int speed = bundle.getInt(TXLiveConstants.NET_STATUS_NET_SPEED);
    }
```

```
});
```

15. Video resolution acquisition

The Player SDK plays back a video through a URL string. The URL doesn't contain the video information, and you need to access the cloud server to load such information. Therefore, the SDK can only send the video information to your application as event notifications. For more information, see Event Listening.

You can get the resolution information in the following two methods:

Method 1: Use the `NET_STATUS_VIDEO_WIDTH` and `NET_STATUS_VIDEO_HEIGHT` of `onNetStatus` to get the video width and height. For detailed directions, see Playback status feedback (onNetStatus).

Method 2: Directly call `TXVodPlayer.getWidth()` and `TXVodPlayer.getHeight()` to get the current video width and height after receiving the `PLAY_EVT_VOD_PLAY_PREPARED` event callback from the player.

```
mVodPlayer.setVodListener(new ITXVodPlayListener() {
    @Override
    public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
    }

    @Override
    public void onNetStatus(TXVodPlayer player, Bundle bundle) {
        // Get the video width
        int videoWidth = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_WIDTH);
        // Get the video height
        int videoHeight = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_HEIGHT);
    }
});

// Get the video width and height. The values can be returned only after the `PLAY_
mVodPlayer.getWidth();
mVodPlayer.getHeight();
```

16. Player buffer size

During normal video playback, you can control the maximum size of the data buffered from the network in advance. If the maximum buffer size is not configured, the player will use the default buffer policy to guarantee a smooth playback experience.

```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setMaxBufferSize(10);  // Maximum buffer size during playback in MB
mVodPlayer.setConfig(config);  // Pass in `config` to `mVodPlayer`
```

17. Local video cache

In short video playback scenarios, the local video file cache is required, so that general users don't need to consume traffic again to reload an already watched video.

Supported format: The SDK supports caching videos in two common VOD formats: HLS (M3U8) and MP4.

Enablement time: The SDK doesn't enable the caching feature by default. We recommend you do not enable it for scenarios in which most videos are watched only once.

Enablement method: This feature can be enabled in the player and takes effect globally. To enable it, you need to configure two parameters: local cache directory and cache size.

```
File sdcardDir = getApplicationContext().getExternalFilesDir(null);
if (sdcardDir != null) {
    // Set the global cache directory of the playback engine
   TXPlayerGlobalSetting.setCacheFolderPath(sdcardDir.getPath() + "/txcache");
    // Set the global cache directory and cache size in MB of the playback engine
    TXPlayerGlobalSetting.setMaxCacheSize(200);
}

// Use the player
```

Note:

The `TXVodPlayConfig#setMaxCacheItems` API used for configuration on earlier versions has been deprecated and is not recommended.

18. DRM-encrypted video playback

Note：

This feature requires the premium version of the player to be supported.

DRM playback must be done by importing the Surface of the system `android.view.SurfaceView` through `TXVodPlayer#setSurface` before playing, and TXCloudVideoView cannot be used.

The premium version of Player SDK supports playback of commercial DRM-encrypted videos, currently supporting two DRM schemes: WideVine and Fairplay. For more information on commercial DRM, please refer to the product introduction.

You can play it as follows :

Playback via FileId

Customized playback configuration

```
// DRM playback must be done by importing the Surface of the system android.view.Su
mVodPlayer.setSurface(surfaceView.getHolder().getSurface());

// psign is the signature of the player. For information on generating and using pl
TXPlayInfoParams playInfoParam = new TXPlayInfoParams(${appId}, // Tencent Cloud ac
    ${fieId}, // The fileId of the DRM-encrypted video
    ${psgin}); // The player signature of the encrypted video
mVodPlayer.startVodPlay(playInfoParam);
```

Playback via FileId is suitable for integration with the VOD backend. This method is no different from playing regular FileId files, but DRM resources need to be configured in the VOD backend first, and the SDK will recognize and process them internally.

```
// DRM playback must be done by importing the Surface of the system android.view.Su
mVodPlayer.setSurface(surfaceView.getHolder().getSurface());

// Step 1: Set the DRM certificate provider environment. This step is not required
// The Google DRM certificate provider environment defaults to the googleapis.com d
TXPlayerGlobalSetting.setDrmProvisionEnv(TXPlayerGlobalSetting.DrmProvisionEnv.DRM_

// Step 2: Play using the TXVodPlayer#startPlayDrm interface.
TXPlayerDrmBuilder builder = new TXPlayerDrmBuilder();
builder.setPlayUrl(${url});  // Set the URL of the video to be played.
builder.setKeyLicenseUrl(${keyLicneseUrl});   // Set decryption key URL
mVodPlayer.startPlayDrm(builder);
```

19. External Subtitles

Note：

This feature requires the premium version of the player to be supported.

The premium version of the player SDK supports adding and switching external subtitles, and now supports two subtitle formats: SRT and VTT.

Best practice: It is recommended to add subtitles and configure subtitle styles before calling startVodPlay. After receiving the VOD_PLAY_EVT_VOD_PLAY_PREPARED event, call selectTrack to choose the subtitle. Adding subtitles does not automatically load them. After calling selectTrack, the subtitles will be loaded. The successful selection of subtitles will trigger the VOD_PLAY_EVT_SELECT_TRACK_COMPLETE event callback.

The usage is as follows:

Step 1: Set the subtitle rendering target object View.

```
// Add TXSubtitleView to the layout XML where the player is located.
<com.tencent.rtmp.ui.TXSubtitleView
            android:id="@+id/subtitle_view"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>

mSubtitleView = findViewById(R.id.subtitle_view);
// Set the subtitle rendering target object.
mVodPlayer.setSubtitleView(mSubtitleView);
```

Step 2: Add external subtitles.

```
// Pass in the subtitle URL, subtitle name, and subtitle type. It is recommended to
```

```
mVodPlayer.addSubtitleSource("https://mediacloud-76607.gzc.vod.tencent-cloud.com/De
```

Step 3: Switch subtitles after playback starts.

```
// After starting to play the video, select the added external subtitle. Please cal
@Override
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
    if (event == TXVodConstants.VOD_PLAY_EVT_VOD_PLAY_PREPARED) {
        List<TXTrackInfo> subtitleTrackInfoList = mVodPlayer.getSubtitleTrackInfo()
        for (TXTrackInfo track : subtitleTrackInfoList) {
            Log.d(TAG, "TrackIndex= " + track.getTrackIndex() + " ,name= " + track.
            if (TextUtils.equals(track.getName(), "subtitleName")) {
                // Selected Subtitles
                mVodPlayer.selectTrack(track.trackIndex);
            } else {
                // If other subtitles are not required, proceed with deselectTrack.
                mVodPlayer.deselectTrack(track.trackIndex);
            }
        }
    }
}


// If needed, you can listen for track switching messages.
mVodPlayer.setVodListener(new ITXVodPlayListener() {
    @Override
    public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
        if (event == TXVodConstants.VOD_PLAY_EVT_SELECT_TRACK_COMPLETE) {
            int trackIndex = param.getInt(TXVodConstants.EVT_KEY_SELECT_TRACK_INDEX
            int errorCode = param.getInt(TXVodConstants.EVT_KEY_SELECT_TRACK_ERROR_
            Log.d(TAG, "receive VOD_PLAY_EVT_SELECT_TRACK_COMPLETE, trackIndex=" +
        }
    }

    @Override
    public void onNetStatus(TXVodPlayer player, Bundle status) {

    }
});
```

Step 4: Configure subtitle style.

Subtitle style can be configured before or during playback.

```
// For detailed parameter configuration, please refer to the API documentation.
TXSubtitleRenderModel model = new TXSubtitleRenderModel();
model.canvasWidth = 1920;  // Width of the subtitle rendering canvas
model.canvasHeight = 1080;  // Height of the subtitle rendering canvas model
```

```
model.fontColor = 0xFFFFFFFF; // Set the font color of the subtitle, default is whi
model.isBondFontStyle = false;  // Set whether the subtitle font is bold
mVodPlayer.setSubtitleStyle(model);
```

20. Subtitle text callback

Note：

This feature is supported starting from Player Premium 12.3.

The default configuration of the player premium SDK is to render and display subtitles through the built-in engine. You can modify the configuration to support callback text. The business can render and display the subtitle text by itself after obtaining it. Subtitles in SRT and VTT formats are currently supported.

The detailed usage is as follows:

Step 1: Set up the subtitle text callback

```
TXVodPlayConfig config = new TXVodPlayConfig();
Map<String, Object> extInfoMap = new HashMap<>();
extInfoMap.put("450", new Integer(0));
config.setExtInfo(extInfoMap);
mVodPlayer.setConfig(config);
```

Step 2: Add and select subtitles

To add and select subtitle files, please refer to the External Subtitles section.

Step 3: Register to listen for subtitle text callback

After selecting the subtitle, you can register the following interface to listen to the subtitle text content. The meaning of the relevant fields: `TXVodSubtitleData#trackIndex,` the track index of the current subtitle; `TXVodSubtitleData#subtitleData,` the actual subtitle text content. When the callback subtitleData is empty, it means that the subtitle is empty. The business can be encapsulated and displayed; the other fields of the `TXVodSubtitleData` class have no practical meaning for the time being, so don't pay attention to them.

```
mVodPlayer.setVodSubtitleDataListener(new ITXVodSubtitleDataListener() {
    @Override
    public void onSubtitleData(TXVodDef.TXVodSubtitleData subtitleData) {
       long trackIndex = subtitleData.trackIndex; // The track index of the current
       String data = subtitleData.subtitleData; // The actual subtitle text content
       // Display the data subtitle text content as needed
    }
});
```

21. Switching between multiple audio tracks

Note：

This feature requires the premium version of the player to be supported.

The advanced version of the player SDK supports switching between multiple audio tracks embedded in the video. Usage is as follows:

```
// Returns a list of audio track information
List<TXTrackInfo> soundTrackInfoList = mVodPlayer.getAudioTrackInfo();
for (TXTrackInfo trackInfo : soundTrackInfoList) {
    if (trackInfo.trackIndex ==  0) {
        // Switch to the required audio track by checking the trackIndex or name
        mVodPlayer.selectTrack(trackInfo.trackIndex);
    } else {
        // Deselect the unwanted audio track
        mVodPlayer.deselectTrack(trackInfo.trackIndex);
    }
}
```

Using Advanced Features

1. Video preloading

Step 1. Use video preloading

In UGSV playback scenarios, the preloading feature contributes to a smooth viewing experience: When a video is being played, you can load the next video to be played back on the backend. When a user switches to the next video, it will already be loaded and can be played back immediately.

Video preloading can deliver an instant playback effect but has certain performance overheads. It will occupy download bandwidth and thread resources. It is recommended that the number of concurrent video pre-playbacks be controlled within 3. If your business needs to preload many videos, we recommend you use this feature together with video predownloading.

This is how seamless switch works in video playback. You can use `setAutoPlay` in `TXVodPlayer` to implement the feature as follows:

```
// Play back video A: If `autoPlay` is set to `true`, the video will be immediately
String urlA = "http://1252463788.vod2.myqcloud.com/xxxxx/v.f10.mp4";
playerA.setAutoPlay(true);
playerA.startVodPlay(urlA);

// To preload video B when playing back video A, set `setAutoPlay` to `false`
String urlB = @"http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
playerB.setAutoPlay(false);
playerB.startVodPlay(urlB); // The video won't be played back immediately but will
```

After video A ends and video B is automatically or manually switched to, you can call the `resume` function to immediately play back video B.

Note:

After `autoPlay` is set to `false`, make sure that video B has been prepared before calling `resume`, that is, you should call it only after the `PLAY_EVT_VOD_PLAY_PREPARED` event of video B (2013: the player has been prepared, and the video can be played back) is detected.

```
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
    // When video A ends, directly start playing back video B for seamless switch
    if (event == PLAY_EVT_PLAY_END) {
            playerA.stop();
            playerB.setPlayerView(mPlayerView);
            playerB.resume();
        }
}
```

Step 2. Configure the video preloading buffer

You can set a large buffer to play back videos more smoothly under unstable network conditions.
You can set a smaller buffer to reduce the traffic consumption.

Preloading buffer size

This API is used to control the maximum buffer size before the playback starts in preloading scenarios (that is, `AutoPlay` of the player is set to `false` before video playback starts).

```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setMaxPreloadSize(2);  // Maximum preloading buffer size in MB. Set it based
mVodPlayer.setConfig(config);  // Pass in `config` to `mVodPlayer`
```

Playback buffer size

During normal video playback, you can control the maximum size of the data buffered from the network in advance. If the maximum buffer size is not configured, the player will use the default buffer policy to guarantee a smooth playback experience.

```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setMaxBufferSize(10);  // Maximum buffer size during playback in MB
mVodPlayer.setConfig(config);  // Pass in `config` to `mVodPlayer`
```

2. Video predownloading

You can download part of the video content in advance without creating a player instance, so as to start playing back the video faster when using the player. This helps deliver a better playback experience.

Before using the playback service, make sure that video cache has been set.

Note:

1. Video pre-downloading will occupy download bandwidth and thread resources. It is recommended to control the queue and limit the number of concurrent downloads to less than 3.
2. `TXPlayerGlobalSetting` is the global cache setting API, and the original `TXVodConfig` API has been deprecated.
3. The global cache directory and size settings have a higher priority than those configured in `TXVodConfig` of the player.

Download via media URL

Download via media FileId

The code example for pre-downloading video via media URL is as follows:

```
// Set the global cache directory and cache size of the playback engine
File sdcardDir = getApplicationContext().getExternalFilesDir(null);
// Set the global cache directory and cache size of the playback engine
if (sdcardDir != null) {
    TXPlayerGlobalSetting.setCacheFolderPath(sdcardDir.getPath() + "/PlayerCache");
    TXPlayerGlobalSetting.setMaxCacheSize(200); // Unit: MB
}

String palyrl = "http://****";
// Start predownloading
final TXVodPreloadManager downloadManager = TXVodPreloadManager.getInstance(getAppl
final int taskID = downloadManager.startPreload(playUrl, 3, 1920*1080, new ITXVodPr
    @Override
    public void onComplete(int taskID, String url) {
        Log.d(TAG, "preload: onComplete: url: " + url);
    }

    @Override
    public void onError(int taskID, String url, int code, String msg) {
        Log.d(TAG, "preload: onError: url: " + url + ", code: " + code + ", msg: "
    }

});

// Cancel predownloading
downloadManager.stopPreload(taskID);
```

Noe:

Pre-downloading via fileId is supported from version 11.3 onwards.

Pre-downloading via fileId is a time-consuming operation. Please do not call it on the main thread, otherwise an illegal call exception will be thrown. The preferredResolution parameter passed in when calling startPreload should be

consistent with the preferred resolution set when starting playback, otherwise the expected effect will not be achieved. Here is an example of how to use it:

```
// Set the global cache directory and cache size for the playback engine File
File sdcardDir = getApplicationContext().getExternalFilesDir(null);
// Set the global cache directory and cache size for the playback engine
if (sdcardDir != null) {
    TXPlayerGlobalSetting.setCacheFolderPath(sdcardDir.getPath() + "/PlayerCache");
    TXPlayerGlobalSetting.setMaxCacheSize(200); // Unit: MB
}

// Start preloading
Runnable task = new Runnable() {
    @Override
    public void run() {
        TXPlayInfoParams playInfoParams = new TXPlayInfoParams(${appId}, "${fileId}
                "${psign}");
        //  Note: This is a time-consuming operation, do not call it on the main th
        mPreLoadManager.startPreload(playInfoParams, 3, 1920 * 1080, new ITXVodFile

            @Override
            public void onStart(int taskID, String fileId, String url, Bundle bundl
                // The onStart method will be called when the file URL is successfu
                Log.d(TAG, "preload: onStart: taskID: " + taskID + ", fileId: " + f
            }

            @Override
            public void onComplete(int taskID, String url) {
                Log.d(TAG, "preload: onComplete: url: " + url);
            }

            @Override
            public void onError(int taskID, String url, int code, String msg) {
                Log.d(TAG, "preload: onError: url: " + url + ", code: " + code + ",
            }
        });
    }
};
new Thread(task).start();


// Cancel preloading
downloadManager.stopPreload(taskID);
```

3. Video download

Video download allows users to download online videos and watch them offline. If the video is encrypted, the downloaded video through the player SDK will be kept in an encrypted state locally and can only be decrypted and played through Tencent Cloud Player SDK. This can effectively prevent illegal dissemination of downloaded videos and protect video security.
As HLS streaming media cannot be directly saved locally, you cannot download them and play back them as local files. You can use the video download scheme based on `TXVodDownloadManager` to implement offline HLS playback.

Note:

Video download supports downloading MP4 and HLS videos. For nested HLS videos, you need to specify the preferred resolution.

Step 1. Make preparations

When initializing the SDK, set the global storage path for video downloading, preloading, caching, and other functions. Usage is as follows:

```
File sdcardDir = context.getExternalFilesDir(null);
TXPlayerGlobalSetting.setCacheFolderPath(sdcardDir.getPath() + "/txcache");
```

`TXVodDownloadManager` is designed as a singleton; therefore, you cannot create multiple download objects. It is used as follows:

```
TXVodDownloadManager downloader = TXVodDownloadManager.getInstance();
```

Set the httpHeader used for downloading

Configure according to business needs. When the player starts downloading, it will be sent to the server. Player version 12.2 starts to support it.

```
Map<String, String> httpHeaders = new HashMap<>();
downloader.setHeaders(httpHeaders);   // Set the download httpHeader
```

Step 2. Start the download

You can start the download through Fileid or URL as follows:
Through Fileid
Through URL
For download through `Fileid`, you need to pass in `AppID`, `Fileid`, and `qualityId` at least. For signed videos, you also need to pass in `pSign`. If no specific value is passed in to `userName`, it will be `default` by default.

Note: You can download encrypted videos only through `Fileid` and must enter the `psign` parameter.

```
// QUALITY_240P  240p
```

```
// QUALITY_360P  360P
// QUALITY_480P  480p
// QUALITY_540P  540p
// QUALITY_720P  720p
// QUALITY_1080P 1080p
// QUALITY_2K    2k
// QUALITY_4K    4k
// The quality parameter can be customized to take the minimum value of the resolut
// The player SDK will select a stream that is less than or equal to the passed-in
TXVodDownloadDataSource source = new TXVodDownloadDataSource(1252463788, "456497281
downloader.startDownload(source)
```

You need to pass in the download URL at least. Only the non-nested HLS, i.e., single-bitstream HLS, is supported. If no specific value is passed in to `userName`, it will be `default` by default.

```
downloader.startDownloadUrl("http://1500005830.vod2.myqcloud.com/43843ec0vodtranscq
```

Step 3. Receive the task information

Before receiving the task information, you need to set the callback listener first.

```
downloader.setListener(this);
```

You may receive the following task callbacks:

Callback Message	Description
void onDownloadStart(TXVodDownloadMediaInfo mediaInfo)	The task started, that is, the SDK started the download.
void onDownloadProgress(TXVodDownloadMediaInfo mediaInfo)	Task progress. During download, the SDK will frequently call back this API. You can use `mediaInfo.getProgress()` to get the current progress.
void onDownloadStop(TXVodDownloadMediaInfo mediaInfo)	The task stopped. When you call `stopDownload` to stop the download, if this message is received, the download is stopped successfully.
void onDownloadFinish(TXVodDownloadMediaInfo mediaInfo)	Download was completed. If this callback is received, the entire file has been downloaded, and the downloaded file can be played back by `TXVodPlayer`.
void onDownloadError(TXVodDownloadMediaInfo mediaInfo, int error, String reason)	A download error occurred. If the network is disconnected during download, this API will be called back and the

	download task will stop. The error code is in `TXVodDownloadManager`.

The download error codes are as follows:

Error Code	Value	Description
DOWNLOAD_SUCCESS	0	Download succeeded.
DOWNLOAD_AUTH_FAILED	-5001	Request for video information from the VOD console failed. Check whether the fileId and psign parameters are correct.
DOWNLOAD_NO_FILE	-5003	The file of the specified definition does not exist.
DOWNLOAD_FORMAT_ERROR	-5004	The downloaded file format is not supported.
DOWNLOAD_DISCONNECT	-5005	Network disconnected. Check whether the network is normal.
DOWNLOAD_HLS_KEY_ERROR	-5006	Failed to obtain the HLS decryption key.
DOWNLOAD_PATH_ERROR	-5007	Failed to access the download directory. Check whether you have the permission to access the download directory.
DOWNLOAD_403FORBIDDEN	-5008	Authentication information failed to pass when downloading. Check whether the signature (psign) has expired.

As the downloader can download multiple files at a time, the callback API carries the `TXVodDownloadMediaInfo` object. You can access the URL or `dataSource` to determine the download source and get other information such as download progress and file size.

Step 4. Stop the download

You can call the `downloader.stopDownload()` method to stop the download. The parameter is the object returned by `downloader.startDownload()`. **The SDK supports checkpoint restart.** If the download directory is not changed, when you resume downloading a file, the download will start from the point where it stopped.

Step 5. Manage downloads

You can get the download lists of all accounts or the specified account.

```
// Get the download lists of all users
```

```
// You can distinguish between the download lists of different users by `userName`
// getDownloadMediaInfoList is a time-consuming function. Please do not call it in
List<TXVodDownloadMediaInfo> downloadInfoList = downloader.getDownloadMediaInfoList
if (downloadInfoList == null || downloadInfoList.size() <= 0) return;
// Get the download list of the `default` user
List<TXVodDownloadMediaInfo> defaultUserDownloadList = new ArrayList<>();
for(TXVodDownloadMediaInfo downloadMediaInfo : downloadInfoList) {
  if ("default".equals(downloadMediaInfo.getUserName())) {
      defaultUserDownloadList.add(downloadMediaInfo);
  }
}
```

To get the download information of a `Fileid`, such as the current download status and progress, you need to pass in `AppID`, `Fileid`, and `qualityId`.

```
// Get the download information of a `fileId`
TXVodDownloadMediaInfo downloadInfo = downloader.getDownloadMediaInfo(1252463788, "
// Get the total size of the file being downloaded in bytes. This API takes effect
// Note: The total size refers to the size of the original file uploaded to the VOD
int size = downloadInfo.getSize();  // Get the total size of the file being downloa
int duration = downloadInfo.getDuration();      // Get the total duration
int playableDuration = downloadInfo.getPlayableDuration(); // Get the playable dura
float progress = downloadInfo.getProgress();    // Get the download progress
String playPath = downloadInfo.getPlayPath(); // Get the offline playback path, whi
int downloadState = downloadInfo.getDownloadState(); // Get the download status. Fo
boolean isDownloadFinished = downloadInfo.isDownloadFinished(); // If `true` is ret
```

To get the download information of a URL, you need to pass in the URL information.

```
// Get the download information of a URL
TXVodDownloadMediaInfo downloadInfo = downloader.getDownloadMediaInfo("http://12531
```

To delete the download information and relevant file, you need to pass in the `TXVodDownloadMediaInfo` parameter.

```
// Delete the download information
boolean deleteRst = downloader.deleteDownloadMediaInfo(downloadInfo);
```

Step 6: Offline Playback after Download

The downloaded video supports playback without network connection. After downloading, you can obtain the download URL through `TXVodDownloadMediaInfo#getPlayPath` for playback.

```
// `getDownloadMediaInfoList` is a time-consuming function. Please do not call
it in the main thread.
```

```
List<TXVodDownloadMediaInfo> mediaInfoList =
TXVodDownloadManager.getInstance().getDownloadMediaInfoList();
// Business logic finds the media object that needs to be played according to
actual needs
for (TXVodDownloadMediaInfo mediaInfo : mediaInfoList) {
    if (mediaInfo.getDownloadState() == TXVodDownloadMediaInfo.STATE_FINISH) {
// Check if the download is complete
        mVodPlayer.startVodPlay(mediaInfo.getPlayPath());  // Play the
downloaded video
    }
}
```

4. Encrypted playback

The video encryption solution is used in scenarios where the video copyright needs to be protected, such as online education. To encrypt your video resources, you need to alter the player and encrypt and transcode video sources. For more information, see Media Encryption and Copyright Protection Overview.

After you get the `appId` as well as the encrypted video's `fileId` and `psign` in the Tencent Cloud console, you can play back the video as follows:

```
// `psign` is a player signature. For more information on the signature and how to
TXPlayInfoParams playInfoParam = new TXPlayInfoParams(1252463788, // `appId` of the
    "4564972819220421305", // `fileId` of the video
    "psignxxxxxx"); // The player signature
mVodPlayer.startVodPlay(playInfoParam);
```

5. Player configuration

Before calling `statPlay`, you can call `setConfig` to configure the player parameters, such as player connection timeout period, progress callback interval, and maximum number of cached files. `TXVodPlayConfig` allows you to configure detailed parameters. For more information, see Basic Configuration API. Below is the configuration sample code:

```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setEnableAccurateSeek(true);  // Set whether to enable accurate seek. Defaul
config.setMaxCacheItems(5);  // Set the maximum number of cached files to 5
config.setProgressInterval(200);  // Set the progress callback interval in ms
config.setMaxBufferSize(50);  // Set the maximum preloading buffer size in MB
mVodPlayer.setConfig(config);  // Pass in `config` to `mVodPlayer`
```

Specifying resolution when playback starts

When playing back an HLS multi-bitrate video source, if you know the video stream resolution information in advance, you can specify the preferred resolution before playback starts, and the player will select and play back the stream at

or below the preferred resolution. In this way, after playback starts, you don't need to call `setBitrateIndex` to switch to the required bitstream.

```
TXVodPlayConfig config = new TXVodPlayConfig();
// The parameter passed in is the product of the video width and height. You can pa
config.setPreferredResolution(TXLiveConstants.VIDEO_RESOLUTION_720X1280);
mVodPlayer.setConfig(config);  // Pass in `config` to `mVodPlayer`
```

Specify media type before broadcasting

When the type of media asset to be played is known in advance, the playback speed can be enhanced by configuring `TXVodPlayConfig#setMediaType` to reduce the internal playback type detection of the player SDK.

Note:

TXVodPlayConfig#setMediaType is supported starting from version 11.2.

```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setMediaType(TXVodConstants.MEDIA_TYPE_FILE_VOD);  // Used to improve MP4 pl
// config.setMediaType(TXVodConstants.MEDIA_TYPE_HLS_VOD);   // Used to improve HLS
mVodPlayer.setConfig(config);
```

Setting playback progress callback interval

```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setProgressInterval(200);  // Set the progress callback interval in ms
mVodPlayer.setConfig(config);  // Pass in `config` to `mVodPlayer`
```

Specify the priority audio track before starting playback

Note:

This feature is supported starting from the Premium Player 12.3 version.

When the name of the audio track to be played is known in advance, you can specify the priority audio track before starting playback by configuring TXVodPlayConfig#setPreferredAudioTrack.

```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setPreferredAudioTrack("audioTrackName"); // audioTrackName is the actual tr
mVodPlayer.setConfig(config); // Pass config to mVodPlayer
```

6. HttpDNS resolution service

Mobile analysis (HTTPDNS) sends domain name resolution requests to DNS servers based on the HTTP protocol, replacing the traditional method of sending resolution requests to the operator's local DNS based on the DNS protocol. This can avoid domain name hijacking and cross-network access issues caused by local DNS, and solve the problem of video playback failure caused by abnormal domain name resolution in mobile Internet services.

Note:

HttpDNS resolution service is supported starting from version 10.9.

1. Enabling HTTPDNS resolution service
You can choose Tencent Cloud or other cloud providers to enable HTTPDNS resolution service. Make sure to integrate it into the player SDK after successful activation.
2. Accessing HTTPDNS resolution service in the player SD
Taking Tencent Cloud HTTPDNS as an example, the following shows how to access it in the player SDK: :
It is not recommended for the business layer to cache domain name resolution results. For example, Tencent HTTPDNS has a built-in intelligent cache mechanism. Additional caching on the business side may result in untimely updates.
Disable the LocaIDNS fallback domain name resolution policy. Configure setUseExpiredIpEnable(false) and setCachedIpEnable(false) through the DnsConfig of the HTTPDNS SDK to avoid obtaining non-optimal or unavailable IPs in the scenario where LocaIDNS has an exception (such as hijacking, pollution, or recursive resolution failure).

```
// Step 1: Enable HttpDNS parsing switch
TXLiveBase.enableCustomHttpDNS(true);
// Step 2: Set HttpDNS resolution callback, TXLiveBaseListener#onCustomHttpDNS
TXLiveBase.setListener(new TXLiveBaseListener() {
    @Override
    public void onCustomHttpDNS(String hostName, List<String> ipList) {
        // The player SDK will callback the hostName to the business. The business
        // If an empty ip is returned, the SDK will not use httpdns for this networ
        // After resolving the hostName to an ip address, save it to iPList and ret
        // Usage example: MSDKDnsResolver is the HTTPDNS SDK resolution API provide
        // String ips = MSDKDnsResolver.getInstance().getAddrByName(hostname);
        String[] ipArr = ips.split(";");
        if (0 != ipArr.length) {
            for (String ip : ipArr) {
                if ("0".equals(ip)) {
                    continue;
                }
                ipList.add(ip);
            }
        }
    }
});
```

7. HEVC Adaptive Downgrade Play

The player supports playing links that contain both HEVC and other video encoding formats, such as H.264. When the player device does not support the HEVC format, it will automatically downgrade to playing videos in the configured alternative encoding format (such as H.264).
Note: Supported from player version 11.7 of the premium version.

```
// Set backup playback link
String backupPlayUrl = "${backupPlayUrl}";  // Alternative playback link
```

```
mVodPlayer.setStringOption(TXVodConstants.VOD_KEY_MIMETYPE, TXVodConstants.VOD_PLAY
mVodPlayer.setStringOption(TXVodConstants.VOD_KEY_BACKUP_URL, backupPlayUrl); // Se

// Set the original HEVC playback link
String hevcPlayUrl = "${hevcPlayUrl}";
mVodPlayer.startVodPlay(hevcPlayUrl);
```

8. Volume Normalization

The player supports automatically adjusting the volume when playing audio to ensure that the volume of all audio is consistent. This can avoid problems with some audio being too loud or too quiet, providing a better auditory experience. Use `TXVodPlayer#setAudioNormalization` to set the volume normalization, with a loudness range of -70 to 0 (LUFS), and custom values are also supported.

Note: Supported from player version 11.7 of the premium version.

```
/**
 Can be set to preset values (related classes or files: Android: TXVodConstants; iO
 Off: AUDIO_NORMALIZATION_OFF
 On: AUDIO_NORMALIZATION_STANDARD (standard)
     AUDIO_NORMALIZATION_LOW (low)
     AUDIO_NORMALIZATION_HIGH (high)
 Custom values can be set: from low to high, range -70 to 0 LUFS
*/
mVodPlayer.setAudioNormalization(TXVodConstants.AUDIO_NORMALIZATION_STANDARD);  //

mVodPlayer.setAudioNormalization(TXVodConstants.AUDIO_NORMALIZATION_OFF);       //
```

9. MP4 video local encryption

After MP4 local encryption is turned on, the player will encrypt and store the data when caching files. Encrypted video files can only be decrypted and played by the player, and cannot be played by third-party players.

Note:

After MP4 local encryption is turned on and playback begins, the encryption settings cannot be changed unless the file is cleared and re-cached.

This feature is supported starting with the Advanced Player 12.2 version.

```
//Precondition: Set the global cache directory of the player. This configuration on
File sdcardDir = getApplicationContext().getExternalFilesDir(null);
if (sdcardDir != null) {
    TXPlayerGlobalSetting.setCacheFolderPath(sdcardDir.getPath() + "/PlayerCache");
}

TXVodPlayConfig config = new TXVodPlayConfig();
config.setEncryptedMp4Level(TXVodConstants.MP4_ENCRYPTION_LEVEL_L2); // Set to use
```

```
mVodPlayer.setConfig(config);
```

Player Event Listening

You can bind a `TXVodPlayListener` listener to the `TXVodPlayer` object to use `onPlayEvent` (event notification) and `onNetStatus` (status feedback) to sync information to your application.

Playback event notifications (onPlayEvent)

Event ID	Code	Description
PLAY_EVT_PLAY_BEGIN	2004	Video playback started.
PLAY_EVT_PLAY_PROGRESS	2005	The video playback progress (including the current playback progress, loading progress, and total video duration).
PLAY_EVT_PLAY_LOADING	2007	The video is being loaded. The `LOADING_END` event will be reported if video playback resumes.
PLAY_EVT_VOD_LOADING_END	2014	Video loading ended, and video playback resumed.
TXVodConstants.VOD_PLAY_EVT_SEEK_COMPLETE	2019	Seeking was completed. The seeking feature is supported by v10.3 or later.
VOD_PLAY_EVT_LOOP_ONCE_COMPLETE	6001	A round of loop was completed. The loop feature is supported by v10.8 or later.
TXVodConstants.VOD_PLAY_EVT_HIT_CACHE	2002	Cache hit event during playback (supported by v11.2 or later).
TXVodConstants.VOD_PLAY_EVT_VIDEO_SEI	2030	Received SEI frame event (supported from player version 11.6 of the premium version).
VOD_PLAY_EVT_HEVC_DOWNGRADE_PLAYBACK	2031	HEVC downgrade playback occurs (supported by the player's advanced version 12.0).
VOD_PLAY_EVT_FIRST_VIDEO_PACKET	2017	The player receives the first frame data packet event (supported by version 12.0).

SEI frame

SEI (Supplemental Enhancement Information) frames are a type of frame used to transmit additional information. The premium version of the player will parse the SEI frames in the video stream and provide callbacks through the `VOD_PLAY_EVT_VIDEO_SEI` event. Note: Supported from player version 11.6 of the premium version.

```
@Override
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
    if (event == TXVodConstants.VOD_PLAY_EVT_VIDEO_SEI) {
        int seiType = param.getInt(TXVodConstants.EVT_KEY_SEI_TYPE);  //  the type
        int seiSize = param.getInt(TXVodConstants.EVT_KEY_SEI_SIZE);  // the data s
        byte[] seiData = param.getByteArray(TXVodConstants.EVT_KEY_SEI_DATA);  // t
    }
}
```

Stop events

Event ID	Code	Description
PLAY_EVT_PLAY_END	2006	Video playback ended.
PLAY_ERR_NET_DISCONNECT	-2301	The network was disconnected and could not be reconnected after multiple retries. You can restart the player to perform more connection retries.
PLAY_ERR_HLS_KEY	-2305	Failed to get the HLS decryption key.

Warning events

You can ignore the following events, which are only used to notify you of some internal events of the SDK.

Event ID	Code	Description
PLAY_WARNING_VIDEO_DECODE_FAIL	2101	Failed to decode the current video frame.
PLAY_WARNING_AUDIO_DECODE_FAIL	2102	Failed to decode the current audio frame.
PLAY_WARNING_RECONNECT	2103	The network was disconnected, and automatic reconnection was performed (the `PLAY_ERR_NET_DISCONNECT` event will be thrown after three failed attempts).
PLAY_WARNING_HW_ACCELERATION_FAIL	2106	Failed to start the hardware decoder, and the software decoder was used instead.

Connection events

The following server connection events are mainly used to measure and collect the server connection time:

Event ID	Code	Description
PLAY_EVT_VOD_PLAY_PREPARED	2013	The player has been prepared and can start playback. If `autoPlay` is set to `false`, you need to call `resume` after receiving this event to start playback.
PLAY_EVT_RCV_FIRST_I_FRAME	2003	The network received the first renderable video data packet (IDR).

Image quality events

The following events are used to get image change information:

Event ID	Code	Description
PLAY_EVT_CHANGE_RESOLUTION	2009	The video resolution changed.
PLAY_EVT_CHANGE_ROTATION	2011	The MP4 video was rotated.

Video information events

Event ID	Code	Description
TXLiveConstants.PLAY_EVT_GET_PLAYINFO_SUCC	2010	Obtained the information of the file played back successfully.

If you play back a video through `fileId` and the playback request succeeds (called API: `startVodPlay(TXPlayInfoParams playInfoParams)`), the SDK will notify the upper layer of some request information, and you can parse `param` to get the video information after receiving the `TXLiveConstants.PLAY_EVT_GET_PLAYINFO_SUCC` event.

Video Information	Description
EVT_PLAY_COVER_URL	Video thumbnail URL
EVT_PLAY_URL	Video playback address
EVT_PLAY_DURATION	Video duration
EVT_DESCRIPTION	Event description
EVT_PLAY_NAME	Video name
TXVodConstants.EVT_IMAGESPRIT_WEBVTTURL	Download URL of the image sprite WebVTT file, which is supported by v10.2 or later

TXVodConstants.EVT_IMAGESPRIT_IMAGEURL_LIST	The download URL of the image sprite image, which is supported by v10.2 or later.
TXVodConstants.EVT_DRM_TYPE	The encryption type, which is supported by v10.2 or later.
TXVodConstants.EVT_KEY_WATER_MARK_TEXT	Ghost watermark text content (supported from version 11.6).

Below is the sample code of using `onPlayEvent` to get the video playback information:

```
mVodPlayer.setVodListener(new ITXVodPlayListener() {
    @Override
    public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
        if (event == TXLiveConstants.PLAY_EVT_VOD_PLAY_PREPARED) {
            // The player preparation completion event is received, and you can cal
        } else if (event == TXLiveConstants.PLAY_EVT_PLAY_BEGIN) {
            // The playback start event is received
        } else if (event == TXLiveConstants.PLAY_EVT_PLAY_END) {
            // The playback end event is received
        }
    }

    @Override
    public void onNetStatus(TXVodPlayer player, Bundle bundle) {
    }
});
```

Ghost watermark

The content of the ghost watermark is filled in the player signature and is ultimately displayed on the playback end through collaboration between the cloud and the player, ensuring the security of the watermark throughout the transmission process. Follow the tutorial to configure the ghost watermark in the player signature. The content of the ghost watermark can be obtained through `param.getString(TXVodConstants.EVT_KEY_WATER_MARK_TEXT)` after receiving the `TXVodConstants#VOD_PLAY_EVT_GET_PLAYINFO_SUCC` event from the player. For detailed usage tutorial, please refer to SuperPlayer Component > Ghost Watermark.

Note: Supported from player version 11.6.

Playback error event

Note:

Error events [-6004, -6010] are supported starting from version 11.0.

Event ID	Value	Description
PLAY_ERR_NET_DISCONNECT	-2301	Video data error that cannot be recovered by

		retrying. For example, network anomaly or data download error that causes demultiplexing timeout or failure.
PLAY_ERR_HLS_KEY	-2305	Failed to obtain HLS decryption key.
VOD_PLAY_ERR_SYSTEM_PLAY_FAIL	-6004	System player playback error.
VOD_PLAY_ERR_DECODE_VIDEO_FAIL	-6006	Video decoding error, video format not supported.
VOD_PLAY_ERR_DECODE_AUDIO_FAIL	-6007	Audio decoding error, audio format not supported.
VOD_PLAY_ERR_DECODE_SUBTITLE_FAIL	-6008	Subtitle decoding error.
VOD_PLAY_ERR_RENDER_FAIL	-6009	Video rendering error.
VOD_PLAY_ERR_PROCESS_VIDEO_FAIL	-6010	Video post-processing error.
VOD_PLAY_ERR_GET_PLAYINFO_FAIL	-2306	Failed to obtain the on-demand file information. It is recommended to check whether the AppId, FileId or Psign is filled in correctly.

Playback status feedback (onNetStatus)

The status feedback is triggered once every 0.5 seconds to provide real-time feedback on the current status of the pusher. It can act as a dashboard to inform you of what is happening inside the SDK so you can better understand the current video playback status.

Parameter	Description
NET_STATUS_CPU_USAGE	Current instantaneous CPU utilization
NET_STATUS_VIDEO_WIDTH	Video resolution - width
NET_STATUS_VIDEO_HEIGHT	Video resolution - height
NET_STATUS_NET_SPEED	Current network data reception speed in KBps
NET_STATUS_VIDEO_FPS	Current video frame rate of streaming media
NET_STATUS_VIDEO_BITRATE	Current video bitrate in bps of streaming media
NET_STATUS_AUDIO_BITRATE	Current audio bitrate in bps of streaming media
NET_STATUS_VIDEO_CACHE	Buffer (`jitterbuffer`) size. If the current buffer length is 0, lag will occur

	soon.
NET_STATUS_SERVER_IP	Connected server IP

Below is the sample code of using `onNetStatus` to get the video playback information:

```
mVodPlayer.setVodListener(new ITXVodPlayListener() {
    @Override
    public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
    }

    @Override
    public void onNetStatus(TXVodPlayer player, Bundle bundle) {
        // Get the current CPU utilization
        CharSequence cpuUsage = bundle.getCharSequence(TXLiveConstants.NET_STATUS_C
        // Get the video width
        int videoWidth = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_WIDTH);
        // Get the video height
        int videoHeight = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_HEIGHT);
        // Get the real-time rate in Kbps
        int speed = bundle.getInt(TXLiveConstants.NET_STATUS_NET_SPEED);
        // Get the current video frame rate of streaming media
        int fps = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_FPS);
        // Get the current video bitrate in bps of streaming media
        int videoBitRate = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_BITRATE);
        // Get the current audio bitrate in bps of streaming media
        int audioBitRate = bundle.getInt(TXLiveConstants.NET_STATUS_AUDIO_BITRATE);
        // Get the buffer (`jitterbuffer`) size. If the current buffer length is 0,
        int jitterbuffer = bundle.getInt(TXLiveConstants.NET_STATUS_VIDEO_CACHE);
        // Get the connected server IP
        String ip = bundle.getString(TXLiveConstants.NET_STATUS_SERVER_IP);
    }
});
```

Other functions

HLS live video source playback

The premium version of the player supports playing HLS live video sources. Starting from version 11.8, it supports live video sources with HLS EVENT. Usage is as follows:

```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setMediaType(TXVodConstants.MEDIA_TYPE_HLS_LIVE); // Specify the HLS live me
mVodPlayer.setConfig(config);
```

```
mVodPlayer.startVodPlay(${YOUR_HSL_LIVE_URL});
```

Scenario-Specific Features

1. SDK-based demo component

Based on the Player SDK, Tencent Cloud has developed a player component. 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 popular video apps.

2. Open-source GitHub projects

Based on the Player SDK, Tencent Cloud has developed immersive video player, video feed stream, and multi-layer reuse components and will provide more user scenario-based components on future versions. You can download Player for Android to try them out.

Flutter Integration
Integration Guide

Last updated：2025-04-01 15:11:25

Environment Requirements

Flutter 3.0 or later
Developing for Android:
Android Studio 3.5 or later.
Devices with Android 4.1 or later.
Developing for iOS:
Xcode 11.0 or later.
OS X 10.11 or later.
A valid developer signature for your project.

SDK Download

You can download the Player SDK for Flutter here.

Note：

To run this demo, you need to set your own player license in the demo_config, and modify the package name and bundleId to your signed package name and bundleId in the Android and iOS configurations.

Quick Integration

Adding the following dependencies to the `pubspec.yaml` of your project

You can integrate `LiteAVSDK_Player` or `LiteAVSDK_Professional` as needed.
1. To integrate the latest version of LiteAVSDK_Player_Premium (player premium version), add configuration in pubspec.yaml：

```
super_player:
  git:
    url: https://github.com/LiteAVSDK/Player_Flutter
    path: Flutter
    ref: Player_Premium
```

Integrate the latest version of `LiteAVSDK_Player` (which is integrated by default) and add the following configuration to `pubspec.yaml` :

```
super_player:
  git:
    url: https://github.com/LiteAVSDK/Player_Flutter
    path: Flutter
```

To integrate the latest version of `LiteAVSDK_Professional` , change the configuration in `pubspec.yaml` as follows:

```
super_player:
  git:
    url: https://github.com/LiteAVSDK/Player_Flutter
    path: Flutter
    ref: Professional
```

To integrate a specified version of the player SDK, specify the corresponding version through the `tag` that `ref` depends on as follows:

```
super_player:
  git:
    url: https://github.com/LiteAVSDK/Player_Flutter
    path: Flutter
    ref: release_pro_v12.0.0

# release_pro_v12.0.0 indicates the integration of Android version
# TXLiteAVSDK_Professional_12.0.0.14681 and iOS version TXLiteAVSDK_Professional_12
```

For more archived tags, see Release List.

Pub integration method

```
# The pub integration defaults to the professional version.
# If there are requirements for other versions, please use the branch integration m
super_player: ^12.3.0
```

2. After the integration, you can obtain the Flutter dependencies through the UI that comes with the code editor or by directly running the following command:

```
flutter pub get
```

3. During use, you can run the following command to update existing Flutter dependencies:

```
flutter pub upgrade
```

Adding native configuration

Android configuration

1. Add the following configuration to `AndroidManifest.xml` .

```
<!--network permission-->
<uses-permission android:name="android.permission.INTERNET" />
<!--storage-->
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Network security configuration allows the app to send HTTP requests

For security reasons, starting from Android P, Google requires that all requests made by the app use encrypted connections. The player SDK will start a local server to proxy HTTP requests. **If your app's targetSdkVersion is greater than or equal to 28**, you can use network security configuration to allow HTTP requests to be sent to 127.0.0.1. Otherwise, you will encounter the error "java.io.IOException: Cleartext HTTP traffic to 127.0.0.1 not permitted" when playing videos, which will prevent the video from playing. The configuration steps are as follows:

1.1 Create a new file network_security_config.xml under res/xml in your project and configure the network security settings.

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">127.0.0.1</domain>
    </domain-config>
</network-security-config>
```

1.2 Add the following attributes to the application tag in the AndroidManifest.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:networkSecurityConfig="@xml/network_security_config"
                    ... >
        ...
    </application>
</manifest>
```

2. Make sure that the `build.gradle` in the Android directory uses `mavenCenter` and can successfully download dependencies.

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

3. Configure the minimum SDK version for Android. Due to the default configuration of Flutter, the minimum version of Android is too low. You need to manually change it to at least 19. If you want to use the picture-in-picture feature,

the compileSdkVersion and targetSdkVersion need to be changed to at least 31.

```
compileSdkVersion 31
defaultConfig {
    applicationId "com.tencent.liteav.demo"
    minSdkVersion 19
    targetSdkVersion 31
    versionCode flutterVersionCode.toInteger()
    versionName flutterVersionName
}
```

4. Add the following configuration `xmlns:tools="http://schemas.android.com/tools"` to the root manifest tag in the AndroidManifest.xml file. Here is an example:

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.player">
 <!-- your config..... -->
</manifest>
```

Add `tools:replace="android:label"` to the application node. Here is an example:

```
<application
    android:label="super_player_example"
    android:icon="@mipmap/ic_launcher"
    android:requestLegacyExternalStorage="true"
    tools:replace="android:label">
<!-- your config..... -->
</application>
```

5. To update the dependency version of the native SDK, manually delete the `build` folder in the Android directory or run the following command to force a refresh.

```
./gradlew build
```

iOS configuration

Note: **iOS currently does not support project running and debugging in simulators; therefore, we recommend you develop and debug your project on a real device**.

1. Add the following configuration to the `Info.plist` file of iOS:

```
<key>NSAppTransportSecurity</key>
<dict>
 <key>NSAllowsArbitraryLoads</key>
 <true/>
```

```
</dict>
```

2. On the iOS native business side, you can edit the podfile file yourself to specify the version of your player SDK. By default, the Player version SDK is integrated.

```
#pod 'TXLiteAVSDK_Player_Premium'         //Player_Premium Edition
pod 'TXLiteAVSDK_Player'                 // Player Edition
```

Integrate `LiteAVSDK_Professional` :

```
pod 'TXLiteAVSDK_Professional'  // Professional Edition
```

If you do not specify the edition, the latest version of `TXLiteAVSDK_Player` will be installed.

3. In some cases such as new version release, you need to forcibly update the iOS player dependencies by running the following command in the iOS directory:

```
rm -rf Pods
rm -rf Podfile.lock
pod update
```

4. Landscape configuration

If your app needs to support landscape mode, you need to set the supported orientations for portrait and landscape in the `Deployment Info` tab on the `General` page of the IOS project configuration. You can check all options as shown in the figure below:

5. Picture-in-Picture configuration

If your project needs to support picture-in-picture, you need to check the "Audio, AirPlay, and Picture in Picture" option under the `Background Modes` tab on the `Signing & Capabilities` page of the IOS project configuration to enable the picture-in-picture capability for your project, as shown in the figure below:

Integrating the Video Playback License

If you have obtained a license, you can view the license URL and key in the RT-Cube console.

If you don't have the required license yet, you can get it as instructed in Adding and Renewing a License.

Before you integrate the player, you need to sign up for a Tencent Cloud account, apply for the video playback license, and configure the license as follows (we recommend you do this when the application is launched):

If you don’t configure a license, errors may occur during playback.

```
String licenceURL = ""; // The license URL obtained
String licenceKey = ""; // The license key obtained
SuperPlayerPlugin.setGlobalLicense(licenceURL, licenceKey);
```

Custom Development

The Flutter plugin of the Player SDK is based on the system's native player capabilities. You can use either of the following methods for custom development:

Use the VOD playback API class `TXVodPlayerController` or live playback API class `TXLivePlayerController` for custom development. You can refer to the demos we provide (`DemoTXVodPlayer` and `DemoTXLivePlayer` in `example`).

The player component `SuperPlayerController` encapsulates VOD playback and live playback capabilities and includes basic UI elements.

You can copy the code of the player component (in `example/lib/superplayer`) to your project for custom development.

FAQs

1. Errors about missing APIs such as `No visible @interface for 'TXLivePlayer' declares the selector 'startLivePlay:type:'` occur on iOS.

Run the following command to update the iOS SDK:

```
rm -rf Pods
rm -rf Podfile.lock
pod update
```

2. SDK or symbol conflicts occur when `tencent_trtc_cloud` and Flutter player are integrated at the same time.

Common exception log: `java. lang.RuntimeException: Duplicate class com.tencent.liteav.TXLiteAVCode found in modules classes.jar`

In this case, you need to integrate the Professional version of the Flutter player, so that `tencent_trtc_cloud` and the Flutter player both depend on the same version of `LiteAVSDK_Professional` .

For example, to depend on `TXLiteAVSDK_Professional_10.3.0.11196` for Android or `TXLiteAVSDK_Professional to 10.3.12231` for iOS, the dependency declaration is as follows:

```
tencent_trtc_cloud: 2.3.8

super_player:
```

```
  git:
    url: https://github.com/LiteAVSDK/Player_Flutter
    path: Flutter
    ref: release_pro_v1.0.3.11196_12231
```

3. When multiple player instances need to be used at the same time, the video image gets blurry when videos are frequently switched.

When each player component container is terminated, call the `dispose` method of the player to release the player.

4. Other common issues with Flutter dependencies:

Run the `flutter doctor` command to check the runtime environment until "No issues found!" is displayed.

Run `flutter pub get` to ensure that all dependent components have been updated successfully.

5. After integrating SuperPlayer, you may encounter the following manifest error:

```
Attribute application@label value=(super_player_example) from AndroidManifest.xml:9
is also present at [com.tencent.liteav:LiteAVSDK_Player:10.8.0.13065] AndroidManife
Suggestion: add 'tools:replace="android:label"' to <application> element at Android
```

Solution: The SuperPlayer Android SDK has already defined the label attribute in its AndroidManifest, and when you create a new Flutter project, the AndroidManifest in the Android directory also defines the label attribute. Therefore, it is recommended that you follow the error message and add `xmlns:tools="http://schemas.android.com/tools"` to the root `manifest` tag in your Android project directory's AndroidManifest.xml. Then, add `tools:replace="android:label"` to the application node. This will override the label attribute and resolve the conflict.

6. After integrating SuperPlayer, you may encounter version conflicts with other dependencies. The error message may look like this:

```
uses-sdk:minSdkVersion 16 cannot be smaller than version 19 declared in library [:s
```

Solution: Currently, the minimum supported version for the SuperPlayer Android SDK is Android 19, while some versions of Flutter default to Android 16 as the minimum supported version. It is recommended that you raise the minimum supported version to Android 19. To do this, go to the main module of your Android project, which is usually the `app` directory, and modify the `minSdkVersion` in the `build.gradle` file to 19.

7. How to extract the runtime logs of the player SDK

Solution: The player SDK outputs the runtime logs to a local file by default. Tencent Cloud technical support may need these logs to diagnose issues. On the Android platform, the logs are saved in the directory `/sdcard/Android/data/packagename/files/log/tencent/liteav`, while on the iOS platform, the logs are saved in the `sandbox/Documents/log` directory.

8. How to reduce console log output

Solution: You can set the log output level using the following interface: SuperPlayerPlugin.setLogLevel(TXLogLevel.LOG_LEVEL_NULL). The following log levels are supported:

```
class TXLogLevel {
  static const LOG_LEVEL_VERBOSE = 0; // Output logs of all levels.
  static const LOG_LEVEL_DEBUG = 1; // Output logs of DEBUG, INFO, WARNING, ERROR,
  static const LOG_LEVEL_INFO = 2; // Output logs of INFO, WARNING, ERROR, and FATA
  static const LOG_LEVEL_WARN = 3; // Output logs of WARNING, ERROR, and FATAL leve
  static const LOG_LEVEL_ERROR = 4; // Output logs of ERROR and FATAL levels.
  static const LOG_LEVEL_FATAL = 5; // Only output logs of FATAL level.
  static const LOG_LEVEL_NULL = 6; // Do not output any SDK logs.
}
```

9. During the use of the project, native-related errors may occur, such as:

`Errors :'incompatible types'` , `error: initializing 'BOOL' (aka 'bool') with an expression of incompatible type 'void'` , etc., are caused by SDK updates, which make the SDK incompatible with the native code in the Flutter end. In this case, you only need to update the SDK version.

Solution: In the project directory, open the terminal and enter the following commands in sequence:

```
flutter pub cache clean
flutter clean
flutter pub upgrade
flutter pub get
```

Make sure the commands execute successfully to update the local Flutter dependencies.

Then, in the ios directory, open the terminal and enter the following command to update the iOS dependencies:

```
rm -rf Pods
rm -rf Podfile.lock
pod update
```

If the problem still persists, you can try deleting the project's build folder and manually deleting the Flutter dependency cache folder `.pub-cache` on your computer. Then refresh the Flutter pub dependencies and compile and run the project again.

10. When playing a video on the Android on-demand player, the edges of the player appear tiled and stretched.

This issue is caused by the texture rendering problem of the flutter end sdk. You can upgrade the flutter version to flutter 3.7.0 or above.

11. The flutter debugging and test package runs without any problems, but the official package crashes when it is installed.

Flutter uses obfuscation by default when building a formal package. The player SDK needs to configure the following obfuscation rules:

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

12. Unable to play local videos.

The flutter player supports local video playback. You need to pass the correct local video address to the video playback interface. If you encounter playback issues, first check if the local video address is available and if the file is damaged. If the local video is fine, check if the app has storage or image/video reading permissions.

13. When running the iOS project, errors such as `CocoaPods could not find compatible versions for pod "Flutter"` or similar are reported.

This issue occurs because the high flutter development environment no longer supports lower iOS versions. You can check if the iOS version configured for Minimum Deployments in the project is too low or if it inherits dependencies that only support lower iOS versions.

14. When the player switches the bound texture, the picture often disappears.

Since the Flutter interface often globally refreshes the UI, it causes TXPlayerVideo and the controller to be passively rebound repeatedly, resulting in a texture binding contention issue. It is recommended to use the onRenderViewCreatedListener callback of TXPlayerVideo. After obtaining the viewId, use the setPlayView method of the controller for binding.

More

You can try out all the features by running the example in the project.

The player SDK website provides demos for iOS, Android, and web. Click here to use them.

VOD Scenario

Last updated：2025-05-12 17:34:58

Intended Audience

This document describes some of the proprietary capabilities of Tencent Cloud. Make sure that you have activated the relevant Tencent Cloud services before reading this document. If you don't have an account yet, sign up for free trial first.

This Document Describes

How to integrate the Tencent Cloud RT-Cube Player SDK for Flutter.
How to use the Player SDK for VOD playback.
How to use the underlying capabilities of the Player SDK to implement more features.

Basics

This document describes the VOD playback feature of the Player SDK. You can start by understanding the following basics:
Live streaming and video on demand
In live streaming, the video source is pushed by the host in real time. When the host stops pushing the stream, the player will also stop playing the video. Because the live stream is played back in real time, no progress bar will be displayed in the player during the playback.
In video on demand (VOD), the video source is a video file in the cloud, which can be played back at any time as long as it is not deleted from the cloud. A progress bar is displayed for controlling the playback progress. Typical VOD scenarios including viewing videos on video websites such as Tencent Video and Youku Tudou.
Supported protocols
Common VOD protocols are as listed below. Currently, VOD URLs in HLS format (starting with `http` and ending with `.m3u8`) are popular.

Notes

The Player SDK **does not impose any limits on the sources of playback URLs**, which means that you can use it to play back videos from both Tencent Cloud and non-Tencent Cloud URLs. However, players in the SDK support

only live streaming URLs in FLV, RTMP, and HLS (M3U8) formats, as well as VOD URLs in MP4, HLS (M3U8), and FLV formats.

SDK Integration

Step 1. Integrate the SDK ZIP file

Download and integrate the SDK ZIP file as instructed in Integration Guide.

Step 2. Create a controller

```
TXVodPlayerController _controller = TXVodPlayerController();
```

Step 3. Configure event listening

```
// Listen for the video width and height change and set an appropriate aspect ratio
playEventSubscription = _controller.onPlayerEventBroadcast.listen((event) async {
  // Subscribe to event distribution
  final int code = event["event"];
  if (code == TXVodPlayEvent.PLAY_EVT_CHANGE_RESOLUTION) {
    int? videoWidth = event[TXVodPlayEvent.EVT_PARAM1];
    int? videoHeight = event[TXVodPlayEvent.EVT_PARAM2];
  }
});
```

Step 4. Add a layout

```
@override
Widget build(BuildContext context) {
return Container(
  decoration: BoxDecoration(
      image: DecorationImage(
        image: AssetImage("images/ic_new_vod_bg.png"),
        fit: BoxFit.cover,
      )),
  child: Scaffold(
      backgroundColor: Colors.transparent,
      appBar: AppBar(
        backgroundColor: Colors.transparent,
        title: const Text('VOD'),
      ),
      body: SafeArea(
          child: Container(
```

```
            height: 150,
            color: Colors.black,
            child: Center(
              child: _aspectRatio > 0
                  ? AspectRatio(
                aspectRatio: _aspectRatio,
                child: TXPlayerVideo(
                  androidRenderType: _renderType,
                  onRenderViewCreatedListener: (viewId) {
                    _controller.setPlayerView(viewId);
                  },
                ),
              ) : Container(),
            ),
          ))));
}
```

Step 5. Initialize the player (No longer required to call in version 12.3.1 and later)

```
// Initialize the player and assign the shared texture
await _controller.initialize();
```

Step 6. Start the playback

Through the URL

Through fileId

`TXVodPlayerController` will internally recognize the playback protocol automatically. You only need to pass in your playback URL to the `startVodPlay` function.

```
// Play back the video resource
String _url =
    "http://1400329073.vod2.myqcloud.com/d62d88a7vodtranscq1400329073/59c68fe752858
await _controller.startVodPlay(_url);
```

```
// `psign` is a player signature. For more information on the signature and how to
TXPlayInfoParams params = TXPlayInfoParams(appId: 1252463788,
        fileId: "4564972819220421305", psign: "psignxxxxxx");
await _controller.startVodPlayWithParams(params);
```

Find the target video file in Media Assets, and you can view the `FileId` below the filename.

Play back the video through the `FileId`, and the player will request the backend for the real playback URL. If the network is abnormal or the `FileId` doesn't exist, the `TXLiveConstants.PLAY_ERR_GET_PLAYINFO_FAIL`

event will be received; otherwise, `TXLiveConstants.PLAY_EVT_GET_PLAYINFO_SUCC` will be received, indicating that the request succeeded.

Step 7. Stop the playback

Remember to call the controller termination method when stopping the playback, especially before the next call of `startVodPlay`. This can prevent memory leak and screen flashing issues.

```
@override
void dispose() {
  _controller.dispose();
  super.dispose();
}
```

Basic Feature Usage

1. Playback control

Starting playback

```
// Start playback
_controller.startVodPlay(url)
```

Pausing playback

```
// Pause the video
_controller.pause();
```

Resuming playback

```
// Resume the video
_controller.resume();
```

Stopping playback

```
// Stop the video
_controller.stopPlay(true);
```

Stopping the player

```
// Release the controller
```

```
_controller.dispose();
```

Adjusting playback progress (seek)

When the user drags the progress bar, `seek` can be called to start playback at the specified position. The Player SDK supports accurate seek.

```
double time = 600; // The value is of `double` type and is in seconds.
// Adjust the playback progress
_controller.seek(time);
```

Seek to the specified Program Date Time (PDT) point in the video stream

To seek to the specified Program Date Time (PDT) point in the video stream, which enables functions such as fast-forward, rewind, and progress bar jumping, currently only HLS video format is supported.
Note: Starting from version 11.6 of the player's premium edition, this function is supported.

```
int pdtTimeMs = 600; // Unit is milliseconds
_controller.seekToPdtTime(time);
```

Specifying playback start time

You can specify the playback start time before calling `startVodPlay` for the first time.

```
double startTimeInSecond = 60; // Unit: Second.
_controller.setStartTime(startTimeInSecond);   // Set the playback start time
_controller.startVodPlay(url);
```

2. Adjustable-Speed playback

The VOD player supports adjustable-speed playback. You can use the `setRate` API to set the VOD playback speed, such as 0.5x, 1.0x, 1.2x, and 2x speed.

```
// Set playback at 1.2X rate
_controller.setRate(1.2);
```

3. Playback loop

```
// Set playback loop
_controller.setLoop(true);
// Get the current playback loop status
_controller.isLoop();
```

4. Muting/Unmuting

```
// Mute or unmute the player. true: Mute; false: Unmute
_controller.setMute(true);
```

5. Roll image ad

The Player SDK allows you to add roll images on the UI for advertising as follows:
If `autoPlay` is set to `NO`, the player will load the video normally but will not immediately start playing it back. Users can see the roll image ad on the player UI after the player is loaded and before the video playback starts. When the ad display stop conditions are met, the `resume` API will be called to start video playback.

```
_controller.setAutoPlay(false);  // Set manual playback
_controller.startVodPlay(url);    // The video will be loaded after `startVodPlay`
// ......
// Display the ad on the player UI
// ......
_controller.resume();  // Call `resume` to start playing back the video after the a
```

6. HTTP-REF

`headers` in `TXVodPlayConfig` can be used to set HTTP request headers, such as the `Referer` field commonly used to prevent the URL from being copied arbitrarily (Tencent Cloud provides a more secure signature-based hotlink protection solution) and the `Cookie` field for client authentication.

```
FTXVodPlayConfig playConfig = FTXVodPlayConfig();
Map<String, String> httpHeaders = {'Referer': 'Referer Content'};
playConfig.headers = httpHeaders;
_controller.setConfig(playConfig);
```

7. Hardware acceleration

It is extremely difficult to play back videos of the Blu-ray (1080p) or higher image quality smoothly if only software decoding is used. Therefore, if your main scenario is game live streaming, we recommend you use hardware acceleration.
Before switching between software and hardware decoding, you need to call **stopPlay** first. After the switch, you need to call **startVodPlay**; otherwise, severe blurs will occur.

```
_controller.stopPlay(true);
_controller.enableHardwareDecode(true);
_controller.startVodPlay(url);
```

8. Definition settings

The SDK supports the multi-bitrate format of HLS, so users can switch between streams at different bitrates to switch the video definition. You can set the definition as follows:

```
List _supportedBitrates = (await _controller.getSupportedBitrates())!;; // Get the
int index = _supportedBitrates[i];  // Specify the subscript of the bitrate of the
_controller.setBitrateIndex(index);  // Switch to the stream at the target bitrate
```

During playback, you can call `_controller.setBitrateIndex(int)` at any time to switch the bitrate. During switch, the data of another stream will be pulled. The SDK is optimized for Tencent Cloud multi-bitrate files to implement smooth switch.

If you know the resolution information of the video stream in advance, you can specify the video resolution to be played before starting playback to avoid switching streams after playback. For detailed methods, refer to Player Configuration#Specifying Resolution Before Playback.

9. Adaptive bitrate streaming

The SDK supports adaptive bitrate streaming of HLS. After this capability is enabled, the player can dynamically select the most appropriate bitrate for playback based on the current bandwidth. You can enable adaptive bitrate streaming as follows:

```
_controller.setBitrateIndex(-1); // Pass in `-1` for the `index` parameter
```

During playback, you can call `_controller.setBitrateIndex(int)` at any time to switch to another bitrate. After the switch, adaptive bitrate streaming will be disabled.

10. Enabling smooth bitrate switching

Before starting playback, you can enable smooth bitrate switching to seamlessly switch between different definitions (bitrates) during playback. If smooth bitrate switch is enabled, the transition between different bitrates will be smoother but will be more time-consuming. Therefore, this feature can be configured as needed.

```
FTXVodPlayConfig playConfig = FTXVodPlayConfig();
/// If it is set to `true`, the bitrate can be switched smoothly. If it is set to `
playConfig.smoothSwitchBitrate = true;
_controller.setConfig(playConfig);
```

11. Playback progress listening

There are two metrics for the VOD progress: **loading progress** and **playback progress**. Currently, the SDK notifies the two progress metrics in real time through event notifications.

You can use the `onPlayerEventBroadcast` API to listen on player events, and the progress notification will be called back to your application through the **PLAY_EVT_PLAY_PROGRESS** event.

```
_controller.onPlayerEventBroadcast.listen((event) async {
```

```
  if(event["event"] == TXVodPlayEvent.PLAY_EVT_PLAY_PROGRESS) {// For more informat
    // Playable duration, i.e., loading progress, in milliseconds
    double playableDuration = event[TXVodPlayEvent.EVT_PLAYABLE_DURATION_MS].toDoub
    // Playback progress in seconds
    int progress = event[TXVodPlayEvent.EVT_PLAY_PROGRESS].toInt();
    // Total video duration in seconds
    int duration = event[TXVodPlayEvent.EVT_PLAY_DURATION].toInt();
  }
});
```

12. Playback network speed listening

You can use the `onPlayerNetStatusBroadcast` API to listen on the player network status such as `NET_STATUS_NET_SPEED`.

```
_controller.onPlayerNetStatusBroadcast.listen((event) {
    (event[TXVodNetEvent.NET_STATUS_NET_SPEED]).toDouble();
});
```

13. Video resolution acquisition

The Player SDK plays back a video through a URL string. The URL doesn't contain the video information, and you need to access the cloud server to load such information. Therefore, the SDK can only send the video information to your application as event notifications.

You can get the resolution information in the following two methods:

Method 1: Use the `NET_STATUS_VIDEO_WIDTH` and `NET_STATUS_VIDEO_HEIGHT` of `onPlayerNetStatusBroadcast` to get the video width and height.

Method 2: Directly call `getWidth()` and `getHeight()` to get the current video width and height after receiving the `PLAY_EVT_VOD_PLAY_PREPARED` event callback from the player.

```
 _controller.onPlayerNetStatusBroadcast.listen((event) {
  double w = (event[TXVodNetEvent.NET_STATUS_VIDEO_WIDTH]).toDouble();
  double h = (event[TXVodNetEvent.NET_STATUS_VIDEO_HEIGHT]).toDouble();
});

// Get the video width and height. The values can be returned only after the `PLAY_
_controller.getWidth();
_controller.getHeight();
```

14. Player buffer size

During normal video playback, you can control the maximum size of the data buffered from the network in advance. If the maximum buffer size is not configured, the player will use the default buffer policy to guarantee a smooth playback

experience.

```
FTXVodPlayConfig playConfig = FTXVodPlayConfig();
playConfig.maxBufferSize = 10;  ///  The maximum buffer size during playback in MB
_controller.setConfig(playConfig);
```

15. Local video cache

In short video playback scenarios, the local video file cache is required, so that general users don't need to consume traffic again to reload an already watched video.

Supported format: The SDK supports caching videos in two common VOD formats: HLS (M3U8) and MP4.

Enablement time: The SDK doesn't enable the caching feature by default. We recommend you do not enable it for scenarios in which most videos are watched only once.

Enablement method: This feature can be enabled in the player and takes effect globally. To enable it, you need to configure two parameters: local cache directory and cache size.

```
// Set the global cache directory and cache size in MB of the playback engine
SuperPlayerPlugin.setGlobalMaxCacheSize(200);
// Set the global cache directory of the playback engine
SuperPlayerPlugin.setGlobalCacheFolderPath("postfixPath");
```

16. External Subtitles

Note: This feature requires Player Premium version 11.7 to be supported.

The premium version of the player SDK supports adding and switching external subtitles, and now supports two subtitle formats: SRT and VTT.

Best practice: It is recommended to add subtitles and configure subtitle styles before calling startVodPlay. After receiving the VOD_PLAY_EVT_VOD_PLAY_PREPARED event, call selectTrack to choose the subtitle. Adding subtitles does not automatically load them. After calling selectTrack, the subtitles will be loaded. The successful selection of subtitles will trigger the VOD_PLAY_EVT_SELECT_TRACK_COMPLETE event callback.When a subtitle is selected, the subtitle text content is passed through TXVodPlayEvent.EVENT_SUBTITLE_DATA event callback, the display of subtitles needs to be handled by the business side.

Step 1: Add external subtitles.

```
// Add external subtitles, pass in the subtitle url, subtitle name, subtitle type,
controller.addSubtitleSource("https://mediacloud-76607.gzc.vod.tencent-cloud.com/De


// After starting to play the video, monitor the subtitle text content callback
_controller.onPlayerEventBroadcast.listen((event) async {
if(event["event"] == TXVodPlayEvent.EVENT_SUBTITLE_DATA) {
   // Subtitle text content, can be used for display
```

```
    String subtitleDataStr = event[TXVodPlayEvent.EXTRA_SUBTITLE_DATA] ?? "";
}
});
```

Step 2: Switch subtitles after playback starts.

```
// After starting to play the video, select the added external subtitles and call i
_controller.onPlayerEventBroadcast.listen((event) async {
if(event["event"] == TXVodPlayEvent.PLAY_EVT_VOD_PLAY_PREPARED) {
     List<TXTrackInfo> trackInfoList = await _vodPlayerController.getSubtitleTrackI
       for (TXTrackInfo tempInfo in trackInfoList) {
         if(tempInfo.name == "subtitleName") {
           //Select subtitles
           _vodPlayerController.selectTrack(tempInfo.trackIndex);
         } else {
           // If other subtitles are not needed, perform deselectTrack
           _vodPlayerController.deselectTrack(tempInfo.trackIndex);
         }
       }
     }
});


// If necessary, you can listen for track switching messages
_controller.onPlayerEventBroadcast.listen((event) async {
   if(event["event"] == TXVodPlayEvent.VOD_PLAY_EVT_SELECT_TRACK_COMPLETE) {
     int trackIndex = event[TXVodPlayEvent.EVT_KEY_SELECT_TRACK_INDEX];
     int errorCode = event[TXVodPlayEvent.EVT_KEY_SELECT_TRACK_ERROR_CODE];
   }
});
```

Step 3: Monitor subtitle text content

```
// After starting to play the video, monitor the subtitle text content callback
_controller.onPlayerEventBroadcast.listen((event) async {
if(event["event"] == TXVodPlayEvent.EVENT_SUBTITLE_DATA) {
   // Subtitle text content, can be used for display
   String subtitleDataStr = event[TXVodPlayEvent.EXTRA_SUBTITLE_DATA] ?? "";
}
});
```

17. Switching between multiple audio tracks

Note: This feature requires Player Premium version 11.7 to be supported.

The advanced version of the player SDK supports switching between multiple audio tracks embedded in the video. Usage is as follows:

```
// Return the audio track information list
List<TXTrackInfo> trackInfoList = await _vodPlayerController.getAudioTrackInfo();
for (TXTrackInfo tempInfo in trackInfoList) {
   if(tempInfo.trackIndex == 0) {
     //Switch to the required audio track by judging trackIndex or name
     _vodPlayerController.selectTrack(tempInfo.trackIndex);
   } else {
     // Unnecessary audio tracks are deselectTrack
     _vodPlayerController.deselectTrack(tempInfo.trackIndex);
   }
}
```

18. Enter Picture-in-Picture

Currently, both platforms support the picture-in-picture feature. On the Android side, you can pass a custom window image with a size limit of 1 MB. If not passed, the default icon will be used. Passing an empty string will hide the icon. On iOS, only the system default icon can be used. The usage is as follows:

```
_playerController.enterPictureInPictureMode(
    backIconForAndroid: backIconForAndroid,
    playIconForAndroid: playIconForAndroid,
    pauseIconForAndroid: pauseIconForAndroid,
    forwardIconForAndroid: forwardIconForAndroid);
```

If you want to enhance the Picture-in-Picture (PiP) experience on iOS or require Flutter's iOS live streaming PiP capability, you can refer to iOS's advanced Picture-in-Picture features. This requires the advanced edition support and the addition of resource files.

19. Bind Texture

After laying out TXPlayerVideo in the build, it needs to be bound to the player to display the picture in TXPlayerVideo. An example is as follows:

```
  controller.setPlayerView(viewId);
```

20. Set Tiling Mode

The controller provides two tiling modes. One is `FTXPlayerRenderMode.ADJUST_RESOLUTION`, which prioritizes displaying the complete video frame while maintaining the aspect ratio. The other is `FTXPlayerRenderMode.FULL_FILL_CONTAINER`, which fills the container while maintaining the aspect ratio. An example is as follows:

```
_controller.setRenderMode(FTXPlayerRenderMode.ADJUST_RESOLUTION);
```

Using Advanced Features

1. Video preloading

Step 1. Use video preloading

In UGSV playback scenarios, the preloading feature contributes to a smooth viewing experience: When a video is being played, you can load the next video to be played back on the backend. When a user switches to the next video, it will already be loaded and can be played back immediately.
Video preloading can deliver an instant playback effect but has certain performance overheads. It will occupy download bandwidth and thread resources. It is recommended that the number of concurrent video pre-playbacks be controlled within 3. If your business needs to preload many videos, we recommend you use this feature together with video predownloading.
This is how seamless switching works in video playback. You can use `setAutoPlay` in `TXVodPlayerController` to implement the feature as follows:

```
// Play back video A: If `autoPlay` is set to `true`, the video will be immediately
String urlA = "http://1252463788.vod2.myqcloud.com/xxxxx/v.f10.mp4";
controller.setAutoPlay(isAutoPlay: true);
controller.startVodPlay(urlA);

// To preload video B when playing back video A, set `setAutoPlay` to `false`
String urlB = "http://1252463788.vod2.myqcloud.com/xxxxx/v.f20.mp4";
controller.setAutoPlay(isAutoPlay: false);
controller.startVodPlay(urlB); // The video won't be played back immediately but wi
```

After video A ends and video B is automatically or manually switched to, you can call the `resume` function to immediately play back video B.

Note:

After `autoPlay` is set to `false`, make sure that video B has been prepared before calling `resume`, that is, you should call it only after the `PLAY_EVT_VOD_PLAY_PREPARED` event of video B (2013: the player has been prepared, and the video can be played back) is detected.

```
controller.onPlayerEventBroadcast.listen((event) async {// Subscribe to status chan
  if(event["event"] == TXVodPlayEvent.PLAY_EVT_PLAY_END) {
    await _controller_A.stop();
```

```
    await _controller_B.resume();
  }
});
```

Step 2. Configure the video preloading buffer

You can set a large buffer to play back videos more smoothly under unstable network conditions.
You can set a smaller buffer to reduce the traffic consumption.

Preloading buffer size

This API is used to control the maximum buffer size before the playback starts in preloading scenarios (that is, `AutoPlay` of the player is set to `false` before video playback starts).

```
TXVodPlayConfig config = new TXVodPlayConfig();
config.setMaxPreloadSize(2);  // Maximum preloading buffer size in MB. Set it based
mVodPlayer.setConfig(config);  // Pass in `config` to `mVodPlayer`
```

Playback buffer size

During normal video playback, you can control the maximum size of the data buffered from the network in advance. If the maximum buffer size is not configured, the player will use the default buffer policy to guarantee a smooth playback experience.

```
FTXVodPlayConfig config = FTXVodPlayConfig();
config.maxBufferSize = 10; // The maximum buffer size during playback in MB
_controller.setPlayConfig(config); // Pass in `config` to `controller`
```

2. Video predownloading

You can download part of the video content in advance without creating a player instance, so as to start playing back the video faster when using the player. This helps deliver a better playback experience.
Before using the playback service, make sure that video cache has been set.
Sample:
Note:
1. Video pre-downloading will occupy download bandwidth and thread resources. It is recommended to control the queue and limit the number of concurrent downloads to less than 3.
2. `TXPlayerGlobalSetting` is the global cache setting API, and the original `TXVodConfig` API has been deprecated.
3. The global cache directory and size settings have a higher priority than those configured in `TXVodConfig` of the player.
Download via media URL
Download via media FileId

```
// Set the global cache directory and cache size of the playback engine
SuperPlayerPlugin.setGlobalMaxCacheSize(200);
// The cache path is set to the application's sandbox directory by default. You onl
// On Android, videos will be cached to the `Android/data/your-pkg-name/files/testC
// On iOS, videos will be cached to the `Documents/testCache` directory in the sand
SuperPlayerPlugin.setGlobalCacheFolderPath("postfixPath");

String palyrl = "http://****";
// Start predownloading
int taskId = await TXVodDownloadController.instance.startPreLoad(palyrl, 3, 1920*10
  onCompleteListener:(int taskId,String url) {
    print('taskID=${taskId} ,url=${url}');
  }, onErrorListener: (int taskId, String url, int code, String msg) {
    print('taskID=${taskId} ,url=${url}, code=${code} , msg=${msg}');
  }
);

// Cancel predownloading
TXVodDownloadController.instance.stopPreLoad(taskId);
```

```
// Set the global cache directory and cache size of the playback engine
SuperPlayerPlugin.setGlobalMaxCacheSize(200);
// The cache path is set to the application's sandbox directory by default. You onl
// On Android, videos will be cached to the `Android/data/your-pkg-name/files/testC
// On iOS, videos will be cached to the `Documents/testCache` directory in the sand
SuperPlayerPlugin.setGlobalCacheFolderPath("postfixPath");

int retTaskId = -1;
TXVodDownloadController.instance.startPreload(TXPlayInfoParams(appId: 0, fileId: "y
    onStartListener: (taskId, fileId, url, params) {
      // TXVodDownloadController will call this block for callback taskId and video
      retTaskId = taskId;
    },
    onCompleteListener: (taskId, url) {
      // preDownload complete
    },
    onErrorListener: (taskId, url, code, msg) {
      // preDownload error
    });

// Cancel predownloading
TXVodDownloadController.instance.stopPreLoad(retTaskId);
```

3. Video download

Video download allows users to download online videos and watch them offline. In addition, the Player SDK provides the local encryption feature, so that downloaded local files are still encrypted and can be decrypted and played back only in the specified player. This feature can effectively prevent downloaded videos from being distributed without authorization and protect the video security.

As HLS streaming media cannot be directly saved locally, you cannot download them and play back them as local files. You can use the video download scheme based on `TXVodDownloadController` to implement offline HLS playback.

Note:

Video download supports downloading MP4 and HLS videos. For nested HLS videos, you need to specify the preferred resolution.

Step 1. Make preparations

`TXVodDownloadController` is designed as a singleton; therefore, you cannot create multiple download objects. It is used as follows:

```
// The cache path is set to the application's sandbox directory by default. You onl
// On Android, videos will be cached to the `Android/data/your-pkg-name/files/testC
// On iOS, videos will be cached to the `Documents/testCache` directory in the sand
SuperPlayerPlugin.setGlobalCacheFolderPath("postfixPath");
```

Step 2. Start the download

You can start the download through Fileid or URL as follows:

Through Fileid

Through URL

For download through `Fileid`, you need to pass in `AppID`, `Fileid`, and `qualityId` at least. For signed videos, you also need to pass in `pSign`. If no specific value is passed in to `userName`, it will be `default` by default.

Note:

You can download encrypted videos only through `Fileid` and must enter the `psign` parameter.

```
// QUALITY_240P  240p
// QUALITY_360P  360P
// QUALITY_480P  480p
// QUALITY_540P  540p
// QUALITY_720P  720p
// QUALITY_1080P 1080p
// QUALITY_2K    2k
// QUALITY_4K    4k
// The quality parameter can be customized to take the minimum value of the resolut
// (for example, for a resolution of 1280*720, if you want to download a stream of
// you can pass in QUALITY_720P for the quality parameter). The player SDK will sel
```

```
// a resolution less than or equal to the passed-in resolution for downloading.
// Using quality ID to download
DownloadHelper.instance.startDownload(videoModel, qualityId);
// Using mediaInfo to download
DownloadHelper.instance.startDownloadOrg(mediaInfo);
```

You need to pass in the download URL at least. Only the non-nested HLS, i.e., single-bitstream HLS, is supported. If no specific value is passed in to `userName`, it will be `default` by default.

```
TXVodDownloadMedialnfo medialnfo = TXVodDownloadMedialnfo();
medialnfo.url = "http://1500005830.vod2.myqcloud.com/43843ec0vodtranscq1500005830/0
TXVodDownloadController.instance.startDonwload(medialnfo);
```

Step 3. Receive the task information

Before receiving the task information, you need to set the callback listener first.

```
TXVodDownloadController.instance.setDownloadObserver((event, info) {

}, (errorCode, errorMsg, info) {

});
```

You may receive the following task events:

Event	Description
EVENT_DOWNLOAD_START	The task started, that is, the SDK started the download.
EVENT_DOWNLOAD_PROGRESS	The task progress. During download, the SDK will frequently call back this API. You can use `mediaInfo.getProgress()` to get the current progress.
EVENT_DOWNLOAD_STOP	The task stopped. When you call `stopDownload` to stop the download, if this message is received, the download is stopped successfully.
EVENT_DOWNLOAD_FINISH	Download was completed. If this callback is received, the entire file has been downloaded, and the downloaded file can be played back by `TXVodPlayer`.

If the `downlodOnErrorListener` method is called back, a download error occurred. If the network is disconnected during download, this API will be called back and the download task will stop.
As the downloader can download multiple files at a time, the callback API carries the `TXVodDownloadMedialnfo` object. You can access the URL or `dataSource` to determine the download

source and get other information such as download progress and file size.

Step 4. Stop the download

You can call the `TXVodDownloadController.instance.stopDownload()` method to stop the download. The parameter is the `TXVodDownloadMedialnfo` object passed in when download starts. **The SDK supports checkpoint restart.** If the download directory is not changed, when you resume downloading a file, the download will start from the point where it stopped.

Step 5. Manage downloads

You can get the download lists of all accounts or the specified account.

```
// Get the download lists of all users
// You can distinguish between the download lists of different users by `userName`
List<TXVodDownloadMedialnfo> downloadInfoList = await TXVodDownloadController.insta
```

To get the download information of a `Fileid`, such as the current download status and progress, you need to pass in `AppID`, `Fileid`, and `qualityId`.

```
// Get the download information of a video
TXVodDownloadMedialnfo downloadInfo = await TXVodDownloadController.instance.getDow
int? duration = downloadInfo.duration;  // Get the total duration
int? playableDuration = downloadInfo.playableDuration; // Get the playable duration
double? progress = downloadInfo.progress;       // Get the download progress
String? playPath = downloadInfo.playPath; // Get the offline playback path, which c
int? downloadState = downloadInfo.downloadState; // Get the download status. For mo
```

```
// Delete the download information
bool result = await TXVodDownloadController.instance.deleteDownloadMediaInfo(medial
```

4. Encrypted playback

The video encryption solution is used in scenarios where the video copyright needs to be protected, such as online education. To encrypt your video resources, you need to alter the player and encrypt and transcode video sources. For more information, see Media Encryption and Copyright Protection Overview.

After you get the `appId` as well as the encrypted video's `fileId` and `psign` in the Tencent Cloud console, you can play back the video as follows:

```
// `psign` is a player signature. For more information on the signature and how to
TXPlayInfoParams params = TXPlayInfoParams(appId: 1252463788,
        fileId: "4564972819220421305", psign: "psignxxxxxx");
_controller.startVodPlayWithParams(params);
```

5. Player configuration

Before calling `statPlay`, you can call `setConfig` to configure the player parameters, such as player connection timeout period, progress callback interval, and maximum number of cached files. `TXVodPlayConfig` allows you to configure detailed parameters. For more information, see Basic Configuration API. Below is the configuration sample code:

```
FTXVodPlayConfig config = FTXVodPlayConfig();
// If `preferredResolution` is not configured, the 720x1280 resolution stream will
config.preferredResolution = 720 * 1280;
config.enableAccurateSeek = true;  // Set whether to enable accurate seek. Default
config.progressInterval = 200;  // Set the progress callback interval in millisecon
config.maxBufferSize = 50;  //  Set the maximum preloading buffer size in MB
_controller.setPlayConfig(config);
```

Specifying resolution when playback starts

When playing back an HLS multi-bitrate video source, if you know the video stream resolution information in advance, you can specify the preferred resolution before playback starts, and the player will select and play back the stream at or below the preferred resolution. In this way, after playback starts, you don't need to call `setBitrateIndex` to switch to the required bitstream.

```
FTXVodPlayConfig config = FTXVodPlayConfig();
//  The parameter passed in is the product of the video width and height. You can p
config.preferredResolution = 720 * 1280;
_controller.setPlayConfig(config);
```

Specify media type before broadcasting

When the type of media asset to be played is known in advance, the playback speed can be enhanced by configuring `TXVodPlayConfig#mediaType` to reduce the internal playback type detection of the player SDK.

```
FTXVodPlayConfig config = FTXVodPlayConfig();
config.mediaType = TXVodPlayEvent.MEDIA_TYPE_FILE_VOD;    // Used to improve MP4 pla
config.mediaType = TXVodPlayEvent.MEDIA_TYPE_HLS_VOD;    // Used to improve HLS pla
_controller.setPlayConfig(config);
```

Setting playback progress callback interval

```
FTXVodPlayConfig config = FTXVodPlayConfig();
config.progressInterval = 200;  // Set the progress callback interval in millisecon
_controller.setPlayConfig(config);
```

Player Event Listening

You can listen on the player's playback events through `onPlayerEventBroadcast` of `TXVodPlayerController` to sync information to your application.

Playback event notifications (`onPlayerEventBroadcast`)

Event ID	Code	Description
PLAY_EVT_PLAY_BEGIN	2004	Video playback started.
PLAY_EVT_PLAY_PROGRESS	2005	The video playback progress (including the current playback progress, loading progress, and total video duration).
PLAY_EVT_PLAY_LOADING	2007	The video is being loaded. The `LOADING_END` event will be reported if video playback resumes.
PLAY_EVT_VOD_LOADING_END	2014	Video loading ended, and video playback resumed.
VOD_PLAY_EVT_SEEK_COMPLETE	2019	Seeking was completed. The seeking feature is supported by v10.3 or later.

Stop events

Event ID	Code	Description
PLAY_EVT_PLAY_END	2006	Video playback ended.
PLAY_ERR_NET_DISCONNECT	-2301	The network was disconnected and could not be reconnected after multiple retries. You can restart the player to perform more connection retries.
PLAY_ERR_HLS_KEY	-2305	Failed to get the HLS decryption key.

Warning events

You can ignore the following events, which are only used to notify you of some internal events of the SDK.

Event ID	Code	Description
PLAY_WARNING_VIDEO_DECODE_FAIL	2101	Failed to decode the current video frame.
PLAY_WARNING_AUDIO_DECODE_FAIL	2102	Failed to decode the current audio frame.
PLAY_WARNING_RECONNECT	2103	The network was disconnected, and automatic

		reconnection was performed (the `PLAY_ERR_NET_DISCONNECT` event will be thrown after three failed attempts).
PLAY_WARNING_HW_ACCELERATION_FAIL	2106	Failed to start the hardware decoder, and the software decoder was used instead.

Connection events

The following server connection events are mainly used to measure and collect the server connection time:

Event ID	Code	Description
PLAY_EVT_VOD_PLAY_PREPARED	2013	The player has been prepared and can start playback. If `autoPlay` is set to `false`, you need to call `resume` after receiving this event to start playback.
PLAY_EVT_RCV_FIRST_I_FRAME	2003	The network received the first renderable video data packet (IDR).
VOD_PLAY_EVT_VOD_PLAY_FIRST_VIDEO_PACKET	2017	Receive the first frame data packet event, supported starting from version 12.0.

Image quality events

The following events are used to get image change information:

Event ID	Code	Description
PLAY_EVT_CHANGE_RESOLUTION	2009	The video resolution changed.
PLAY_EVT_CHANGE_ROTATION	2011	The MP4 video was rotated.

Video information events

Event ID	Code	Description
PLAY_EVT_GET_PLAYINFO_SUCC	2010	Obtained the information of the file played back successfully.

If you play back a video through `fileId` and the playback request succeeds (called API: `startVodPlay(TXPlayerAuthBuilder authBuilder)`), the SDK will notify the upper layer of some request

information, and you can parse `param` to get the video information after receiving the `TXLiveConstants.PLAY_EVT_GET_PLAYINFO_SUCC` event.

Video Information	Description
EVT_PLAY_COVER_URL	Video thumbnail URL
EVT_PLAY_URL	Video playback address
EVT_PLAY_DURATION	Video duration
EVT_TIME	Event occurrence time
EVT_UTC_TIME	UTC time
EVT_DESCRIPTION	Event description
EVT_PLAY_NAME	Video name
EVT_IMAGESPRIT_WEBVTTURL	The download URL of the image sprite WebVTT file, which is supported by v10.2 or later.
EVT_IMAGESPRIT_IMAGEURL_LIST	The download URL of the image sprite image, which is supported by v10.2 or later.
EVT_DRM_TYPE	The encryption type, which is supported by v10.2 or later.

Below is the sample code for using `onPlayerEventBroadcast` to get the video playback information:

```
_controller.onPlayerEventBroadcast.listen((event) async {
  if (event["event"] == TXVodPlayEvent.PLAY_EVT_PLAY_BEGIN || event["event"] == TXV
  // code ...
  } else if (event["event"] == TXVodPlayEvent.PLAY_EVT_PLAY_PROGRESS) {
  // code ...
  }
});
```

Playback status feedback (`onPlayerNetStatusBroadcast`)

The status feedback is triggered once every 0.5 seconds to provide real-time feedback on the current status of the pusher. It can act as a dashboard to inform you of what is happening inside the SDK so you can better understand the current video playback status.

Parameter	Description
NET_STATUS_CPU_USAGE	Current instantaneous CPU utilization

NET_STATUS_VIDEO_WIDTH	Video resolution - width
NET_STATUS_VIDEO_HEIGHT	Video resolution - height
NET_STATUS_NET_SPEED	Current network data reception speed
NET_STATUS_VIDEO_FPS	Current video frame rate of streaming media
NET_STATUS_VIDEO_BITRATE	Current video bitrate in Kbps of streaming media
NET_STATUS_AUDIO_BITRATE	Current audio bitrate in Kbps of streaming media
NET_STATUS_VIDEO_CACHE	Buffer (`jitterbuffer`) size. If the current buffer length is 0, lag will occur soon.
NET_STATUS_SERVER_IP	Connected server IP

Below is the sample code of using `onNetStatus` to get the video playback information:

```
_controller.onPlayerNetStatusBroadcast.listen((event) async {
  int videoWidth = event[TXVodNetEvent.NET_STATUS_VIDEO_WIDTH];
});
```

Video playback status feedback

The video playback status will be notified every time the playback status changes.

The enumeration class `TXPlayerState` is used to transfer the event.

Status	Description
paused	The playback was paused
failed	The playback failed
buffering	Buffering
playing	Playing back
stopped	The playback was stopped
disposed	The control was released

Below is the sample code for using `onPlayerState` to get the video playback status:

```
_controller.onPlayerState.listen((val) { });
```

System volume level listening

To help you monitor the video playback volume level, the SDK encapsulates the volume level change notification into an event at the Flutter layer. You can directly use `SuperPlayerPlugin` to listen on the volume level change of the current device.

Below is the sample code for using `onEventBroadcast` to get the volume level status of the device:

```
SuperPlayerPlugin.instance.onEventBroadcast.listen((event) {
  int eventCode = event["event"];
});
```

The relevant events are as described below:

Status	Code	Description
EVENT_VOLUME_CHANGED	1	The volume level changed.
EVENT_AUDIO_FOCUS_PAUSE	2	The volume level output and playback focus were lost. This event is applicable only to Android.
EVENT_AUDIO_FOCUS_PLAY	3	Obtained the volume level output focus successfully. This event is applicable only to Android.

PiP event listening

As PiP used by the SDK is based on the PiP capabilities of the system, after you enter the PiP mode, a series of notifications are provided to help you adjust the UI accordingly.

Status	Code	Description
EVENT_PIP_MODE_ALREADY_ENTER	1	The player has entered the PiP mode.
EVENT_PIP_MODE_ALREADY_EXIT	2	The player has exited the PiP mode.
EVENT_PIP_MODE_REQUEST_START	3	The player requests to enter the PiP mode.
EVENT_PIP_MODE_UI_STATE_CHANGED	4	The PiP UI status changed. This event takes effect only on Android 31 or later.
EVENT_IOS_PIP_MODE_RESTORE_UI	5	The UI was reset. This event takes effect only on iOS.
EVENT_IOS_PIP_MODE_WILL_EXIT	6	The player will exit the PiP mode. This event takes effect only on iOS.

Below is the sample code for using `onExtraEventBroadcast` to listen on PiP events:

```
SuperPlayerPlugin.instance.onExtraEventBroadcast.listen((event) {
  int eventCode = event["event"];
});
```

Live Streaming Scenario

Last updated：2025-05-12 17:34:57

Basic knowledge

This article mainly introduces the live broadcast function of the Video Cloud SDK. Before that, it will be helpful to understand the following basic knowledge:

Live and on-demand

The video source of live broadcast (LIVE) is pushed by the anchor in real time. Therefore, after the anchor stops pushing, the picture on the playback end will also stop immediately. And because it is a real-time live broadcast, there is no progress bar when the player plays the live URL.

The video source of on-demand (VOD) is a video file in the cloud. As long as it is not removed from the cloud, the video can be played at any time. During playback, you can control the playback position through the progress bar. Video viewing on video websites such as Tencent Video and Youku Tudou is a typical on-demand scenario.

Protocol support

The commonly used live broadcast protocols are as follows. The App end recommends using the live broadcast address of the FLV protocol (starting with "http" and ending with ".flv"):

Protocol	Advantage	Disadvantage	Playback Latency
HLS	Mature, well adapted to high-concurrency scenarios	SDK integration is required	3–5 seconds
FLV	Mature, well adapted to high-concurrency scenarios	SDK integration is required.	2-3 seconds
RTMP	Relatively low latency	Poor performance in high-concurrency scenarios	1-3 seconds
WebRTC	Lowest latency	SDK integration is required	< 1 second

Special Note

Are there any restrictions?

The Video Cloud SDK does not restrict the source of the playback address, that is, you can use it to play playback addresses in Tencent Cloud or non-Tencent Cloud. However, the player in the Video Cloud SDK only supports live

broadcast addresses in three formats: FLV, RTMP, and HLS (m3u8), and on-demand addresses in three formats: MP4, HLS (m3u8), and FLV.

Historical factors

In the early versions of the SDK, there was only one class, TXLivePlayer, to carry the live broadcast and on-demand functions. However, as the on-demand functions became more and more, we finally separated the on-demand functions from SDK version 3.5 and handed them over to TXVodPlayer. However, in order to ensure that the compilation passes, you can still see on-demand functions such as seek in TXLivePlayer.

Integration Guide

Step 1: Create Player

The TXLivePlayer module in the Video Cloud SDK is responsible for implementing the live broadcast function. The corresponding Flutter module is TXLivePlayerController.

```
TXLivePlayerController controller = TXLivePlayerController();
```

Step 2: Render View

Next, we need to find a place to display the video screen of the player. The Flutter system uses Widget as the basic interface rendering unit, so you only need to prepare a Widget and adjust the layout. You can directly use TXPlayerVideo or inherit it to display, or refer to the source code to implement a custom view.

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Live Steaming'),
      ),
      body: SafeArea(
        child: Column(
          children: [
            Container(
              height: 150,
              color: Colors.black,
              child: Center(
                child: _aspectRatio>0?AspectRatio(
                  aspectRatio: _aspectRatio,
                  child: TXPlayerVideo(
                    onRenderViewCreatedListener: (viewId) {
                      _controller.setPlayerView(viewId);
                    },
                  ),
```

```
                ):Container(),
              ),
            ),],
        ),
      ),
    );
  }
```

Step 3: Start playback

```
String flvUrl = "http://liteavapp.qcloud.com/live/liteavdemoplayerstreamid_demo1080
await _controller.startLivePlay(flvUrl);
// If you are using a version lower than 12.0, you need to pass in playType, as sho
await _controller.startLivePlay(flvUrl, playType: TXPlayType.LIVE_FLV);
```

Note：

If you integrate a version lower than 12.0 (not including 12.0), you need to pass in playType when playing the live stream.

The meaning of playType is as follows:

Optional value	Enumeration value	Meaning
PLAY_TYPE_LIVE_RTMP	0	The incoming URL is the RTMP live broadcast address
PLAY_TYPE_LIVE_FLV	1	The incoming URL is the FLV live broadcast address
PLAY_TYPE_LIVE_RTMP_ACC	5	Low latency link address (only suitable for microphone-connected scenarios)
PLAY_TYPE_VOD_HLS	3	The incoming URL is the HLS (m3u8) playback address

Note：

Regarding HLS (m3u8), we do not recommend using the HLS playback protocol to play live video sources on the App (although it is very suitable for on-demand playback) because the latency is too high. We recommend using the LIVE_FLV or LIVE_RTMP playback protocol on the App.

Step 4: Pause playback

For live broadcast, there is no real pause. The so-called live broadcast pause only freezes the picture and turns off the sound. The cloud video source is still updating, so when you call resume, it will start playing from the latest time point,

which is very different from on-demand (the pause and resume of the on-demand player is the same as when playing local video files).

```
// Pause
_controller.pause();
//
_controller.resume();
```

Step 5: End playback

```
// Stop playing
_controller.stop();
```

Step 6: Message Reception

This function can send some custom messages directly to the audience along with the audio and video lines on the streaming end. Applicable scenarios include:
Top Conference: The streaming end sends the topic to the audience, which can achieve perfect synchronization of "audio-video-topic".
Live Show: The streaming end sends the lyrics to the audience, and the lyrics special effects can be drawn in real time on the playback end, so it is not affected by the quality degradation of video encoding.
Online Education: The streaming end sends the laser pen and graffiti operation to the audience, and circles and lines can be drawn in real time on the playback end.
This function can be used through the following schemes:
Monitor messages through onPlayerEventBroadcast, message number: PLAY_EVT_GET_MESSAGE (2012).

```
_controller.onPlayerEventBroadcast.listen((event) {//Subscribe to event distributio
  if(event["event"] == 2012) {
      String msg = event["EVT_GET_MSG"];
  }
});
```

Step 7: Seamless switching of clarity

In daily use, network conditions are constantly changing. In the case of poor network, it is best to moderately reduce the image quality to reduce lag; conversely, with a good network speed, you can watch higher quality images.
The traditional way of switching streams is generally to replay, which will cause problems such as disconnection between the images before and after switching, black screen, and lag. Using a seamless switching solution, you can directly switch to another stream without interrupting the live broadcast.
The clarity switch can be called at any time after the live broadcast starts. The calling method is as follows:

```
// The stream currently being played is http://5815.liveplay.myqcloud.com/live/5815
```



```
// Now switch to a new stream with a bit rate of 900kbps
_controller.switchStream("http://5815.liveplay.myqcloud.com/live/5815_62fe94d692ab1
```

Note：

The seamless clarity switching function requires PTS alignment to be configured in the background. If you need it, you can submit a work order to apply for its use.

Delay adjustment

The live broadcast function of Tencent Cloud SDK is not based on ffmpeg for secondary development, but uses a self-developed playback engine. Therefore, compared with open source players, it has better performance in live broadcast delay control. We provide three delay adjustment modes, which are suitable for: show, game and mixed scenarios.

The characteristics of the three modes are compared in the following table:

Control mode	Stuttering rate	Average delay	Applicable scenarios	Principle description
Extreme mode	Smoother and higher	2s - 3s	Beauty show (top contest)	Advantageous in delay control, suitable for scenes that are sensitive to delay size
Smooth Mode	Lowest lag rate	>= 5s	Game live broadcast (Penguin E-sports)	Very suitable for live broadcast of games with ultra-high bitrate (such as PlayerUnknown's Battlegrounds), with the lowest lag rate
Automatic mode	Network adaptation	2s - 8s	Mixed scenarios	The better the network on the audience side, the lower the delay; the worse the network on the audience side, the higher the delay

The docking codes for the three modes are as follows:

```
// Automatic mode
_controller.setLiveMode(TXPlayerLiveMode.Automatic);
// Speed mode
_controller.setLiveMode(TXPlayerLiveMode.Speed);
// Smooth mode
_controller.setLiveMode(TXPlayerLiveMode.Smooth);

// Start playback after setting is complete
```

Note：

For more technical knowledge about lag and latency optimization, you can read Video Stutter.

Function Usage

The following will introduce how to use the common live broadcast function.

1. Pause Playback

For live broadcast, there is no real pause. The so-called live broadcast pause is just freezing the picture and turning off the sound. The cloud video source is still updating, so when you call resume, it will start playing from the latest time point, which is very different from on-demand (the pause and resume of the on-demand player is the same as when playing local video files).

```
// Pause
_controller.pause();
// Continue
_controller.resume();
```

2. Message Reception

This feature allows the broadcaster to send custom messages along with the audio and video stream directly to the audience. The applicable scenarios are as follows:
Quiz Shows: The broadcaster sends the questions to the audience, achieving perfect synchronization of "audio-video-question".
Show Live: The broadcaster sends lyrics to the audience, allowing real-time rendering of lyric effects on the playback end, unaffected by video encoding quality loss.
Online Education: The broadcaster sends laser pen and doodle operations to the audience, allowing real-time drawing on the playback end.
You can use this feature through the following method:
Set the `enableMessage` switch in `FTXLivePlayConfig` to `true` .
Listen for messages in `TXLivePlayerController` via `onPlayerEventBroadcast` , message ID: **PLAY_EVT_GET_MESSAGE (2012)**.

```
_controller.onPlayerEventBroadcast.listen((event) {
  if (event["event"] == TXVodPlayEvent.PLAY_EVT_GET_MESSAGE) {
    String msg = event[TXVodPlayEvent.EVT_GET_MSG];
  } else if (event["event"] == TXVodPlayEvent.PLAY_ERR_NET_DISCONNECT) {
    print("Network disconnected, streaming failed");
  }
});
```

3. Seamless Quality Switch

During daily use, network conditions are constantly changing. In poor network conditions, it is best to appropriately reduce the video quality to reduce stuttering; conversely, if the network speed is good, you can watch higher-quality videos. Traditional stream switching usually involves restarting playback, which can cause issues like discontinuity, black screens, and stuttering. Using the seamless switching scheme allows you to switch streams directly without interrupting the live broadcast.
Quality switches can be called at any time after the live broadcast starts. The calling method is as follows:

```
// The stream currently being played is http://5815.liveplay.myqcloud.com/live/5815
// Now switch to the new stream with a bit rate of 900kbps
_controller.switchStream("http://5815.liveplay.myqcloud.com/live/5815_62fe94d692ab1
```

Note：
The seamless clarity switching function requires PTS alignment to be configured in the background. If you need it, you can submit a work order to apply for its use.

4. Latency Adjustment

The live streaming (LVB) functionality of the Tencent Cloud SDK is not based on secondary development using ffmpeg but uses a self-developed playback engine. Therefore, compared to open-source players, it performs better in terms of live streaming latency control. We provide three latency adjustment modes suitable for different scenarios: shows, gaming, and mixed scenarios.
Comparison of the three modes:

Control mode	Stuttering rate	Average delay	Applicable scenarios	Principle description
Extreme mode	Smoother and higher	2s - 3s	Beauty show (top contest)	Advantageous in delay control, suitable for scenes that are sensitive to delay size
Smooth Mode	Lowest lag rate	>= 5s	Game live broadcast (Penguin E-sports)	Very suitable for live broadcast of games with ultra-high bitrate (such as PlayerUnknown's Battlegrounds), with the lowest lag rate
Automatic mode	Network adaptation	2s - 8s	Mixed scenarios	The better the network on the audience side, the lower the delay; the worse the network on the audience side, the higher the delay

Three modes of docking code

```
//Automatic mode
_controller.setCacheParams(1, 5);
//Extreme mode
```

```
_controller.setCacheParams(1, 1);
//Smooth mode
_controller.setCacheParams(5, 5);
```

Note：

For more technical knowledge about lag and latency optimization, you can read Video Stutter.

5. Getting Video Information

The Player SDK plays videos through a URL string, which itself does not contain video information. To obtain related information, it is necessary to load the video information by accessing the cloud server. Therefore, the SDK can only send the video information to your application via event notifications. For more content, refer to Event Listening.
For example, you can obtain the width and height of a video through the `onPlayerNetStatusBroadcast` 's `NET_STATUS_VIDEO_WIDTH` and `NET_STATUS_VIDEO_HEIGHT` . For specific usage, see Status Feedback (onPlayerNetStatusBroadcast).

6. Enter Picture-in-Picture

Currently, both platforms support Picture-in-Picture capability. On the Android side, you can pass a custom window image with a size limit of 1MB. If not provided, the default icon will be used. For IOS live streaming Picture-in-Picture, the premium version of SDK permission is required.

```
_playerController.enterPictureInPictureMode(
    backIconForAndroid: backIconForAndroid,
    playIconForAndroid: playIconForAndroid,
    pauseIconForAndroid: pauseIconForAndroid,
    forwardIconForAndroid: forwardIconForAndroid);
```

For Flutter iOS to use live streaming Picture-in-Picture (PiP), advanced edition capabilities are required, along with the addition of corresponding resource files. For details, please refer to the iOS Advanced Picture-in-Picture capabilities.

7. Bind Texture

After laying out TXPlayerVideo in the build, it needs to be bound to the player to display the picture in TXPlayerVideo. An example is as follows:

```
  controller.setPlayerView(viewId);
```

8. Set Tiling Mode

The controller provides two tiling modes. One is `FTXPlayerRenderMode.ADJUST_RESOLUTION` , which prioritizes displaying the complete video frame while maintaining the aspect ratio. The other is

`FTXPlayerRenderMode.FULL_FILL_CONTAINER` , which fills the container while maintaining the aspect ratio. An example is as follows:

```
_controller.setRenderMode(FTXPlayerRenderMode.ADJUST_RESOLUTION);
```

Listening for events

You can monitor the `TXLivePlayerController` 's `onPlayerEventBroadcast` and `onPlayerNetStatusBroadcast` , after which all internal status information of the SDK will be notified to you through `onPlayerEventBroadcast` (event notification) and `onPlayerNetStatusBroadcast` (status feedback).

Event Notification (onPlayerEventBroadcast)

1. Playback events

Event ID	Numerical Value	Description
PLAY_EVT_PLAY_BEGIN	2004	Video playback starts
PLAY_EVT_PLAY_LOADING	2007	Video playback loading, if it can be restored, there will be a LOADING_END event afterward
PLAY_EVT_VOD_LOADING_END	2014	Video playback loading ends, video continues to play

Do not hide the playback screen after receiving PLAY_LOADING: The duration from PLAY_LOADING to PLAY_BEGIN is uncertain, it could be 5s or 5ms. Some users consider hiding the screen during LOADING and displaying it at BEGIN, which causes severe screen flickering (especially in live streaming scenarios).**The recommended practice is to overlay a semi-transparent loading animation on the video playback screen.**

2. Stop events

Event ID	Numerical Value	Description
PLAY_EVT_PLAY_END	2006	Video playback ends
PLAY_ERR_NET_DISCONNECT	-2301	Network disconnected, and after multiple reconnections fails to restore, for more retries please restart playback manually
PLAY_ERR_HLS_KEY	-2305	Failed to get HLS decryption key

How can one determine if a live stream has ended?

Due to different implementation principles based on various standards, many live streams usually do not emit an 'end event' (2006). In such cases, the expected behavior is: after the broadcaster stops streaming, the SDK will quickly detect a failure in data stream retrieval (PLAY_WARNING_RECONNECT), then begin to retry until after three failed attempts, it will then emit a PLAY_ERR_NET_DISCONNECT event.

Therefore, both error codes 2006 and -2301 need to be listened for and used to determine the end of live streaming.

3. Warning events

You can ignore the following events, they are used to inform you about some internal events of the SDK.

Event ID	Numerical Value	Description
PLAY_WARNING_VIDEO_DECODE_FAIL	2101	Failed to decode the current video frame
PLAY_WARNING_AUDIO_DECODE_FAIL	2102	Failed to decode the current audio frame
PLAY_WARNING_RECONNECT	2103	Network Disconnection, automatic reconnection initiated (if reconnection fails more than three times, it will directly throw the PLAY_ERR_NET_DISCONNECT)
PLAY_WARNING_HW_ACCELERATION_FAIL	2106	Failed to enable the hardware decoder, and the software decoder was used

Status feedback (onPlayerNetStatusBroadcast)

The notification is triggered once every second to provide real-time feedback on the current status of the pusher. It can act as a dashboard to inform you of what is happening inside the SDK so you can better understand the current network conditions and video information.

Evaluation parameters	Description
NET_STATUS_CPU_USAGE	Current instantaneous CPU utilization
NET_STATUS_VIDEO_WIDTH	Video resolution - width
NET_STATUS_VIDEO_HEIGHT	Video resolution - height
NET_STATUS_NET_SPEED	Current network data reception speed
NET_STATUS_VIDEO_FPS	Current video frame rate of streaming media
NET_STATUS_VIDEO_BITRATE	Current video bitrate in Kbps of streaming media

NET_STATUS_AUDIO_BITRATE	Current audio bitrate in Kbps of streaming media
NET_STATUS_CACHE_SIZE	Jitterbuffer size; if the current length of the buffer zone is 0, it indicates that lag is imminent
NET_STATUS_SERVER_IP	Connected server IP

Advanced Features
Web Advanced Features
Security Check Plugin (TCPlayerSafeCheckPlugin)

Last updated：2024-04-11 16:18:08

The TCPlayerSafeCheckPlugin is used to check if the playback environment and status are normal, ensuring playback safety. It should be used in conjunction with TCPlayer.

Use Conditions

Currently, the Web Player SDK version 5.0.0 and above support the use of the VR playback plugin.
VR playback requires access to Player Premium Version License（Web） for use.
In the process of long-term maintenance of the player, various types of attacks have been encountered. Against behaviors where video resources can be stolen using third-party tools, this plugin has implemented targeted prevention in the following three aspects:

1. MSE Environment Detection

Some browser plugins or scripts can hijack the current playback environment by modifying the Media Source Extensions API (MSE) to intercept playback data, ultimately enabling video download. This plugin can detect and prevent such attacks.

2. Security Structure Inspection

Third-party tools or scripts can modify the player's structure, removing playback markers, watermarks, etc., and enabling screen recording. This plugin monitors whether the player's structure has been tampered with. If such behavior is detected, playback is immediately halted.

3. API Response Integrity Verification

During the use of the player, it is necessary to interact with the Video on Demand server. If the interface data is modified, it will affect the normal playback behavior. This plugin can detect and prevent such types of attacks.

Use Method

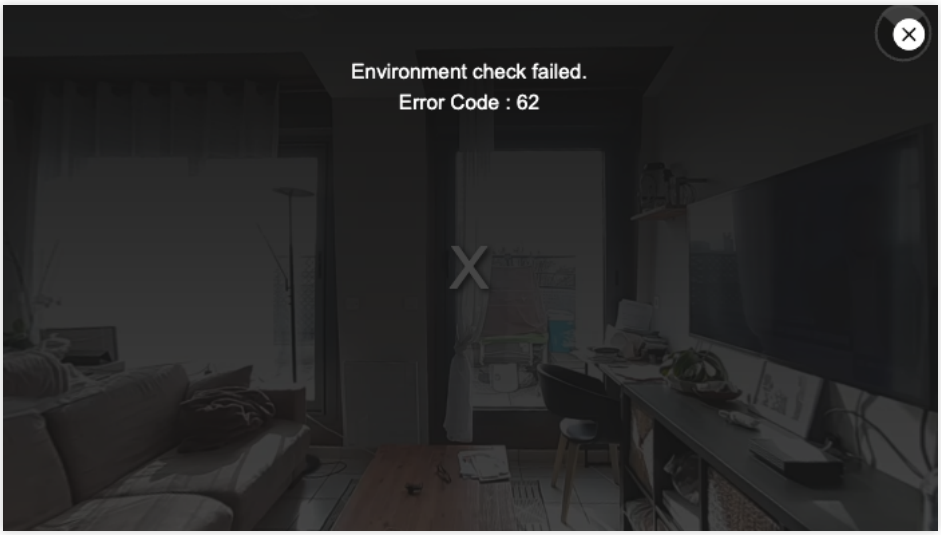
For integration with TCPlayer, refer to TCPlayer Integration Guide and API Documentation.

When creating a player instance, you can enable security check capabilities by claiming plugins. Once enabled, the player will automatically load and use this plugin:

```
const player = TCPlayer('player-container-id', {
  plugins: {
    SafeCheck: true,
  }
});
```

Effect

After enabling the plugin, the player will automatically check if the environment is safe. If an attack is detected, playback will be automatically terminated, and a corresponding prompt will be displayed, as shown below:



The error codes prompted by the plugin are as follows:

Error code	Description
60	Security structure check exception
61	API response integrity exception
62	MSE environment detection exception

VR Playback Plugin (TCPlayerVRPlugin)

Last updated：2024-04-11 16:18:08

The TCPlayerVRPlugin can be used for VR panoramic video playback. During playback, you can change the viewing angle through gyroscope rotation or gesture operations. It offers various properties and methods to control playback performance and supports both PC and mobile platforms.

Use Conditions

Currently, the Web Player SDK version 5.0.0 and above support the use of the VR playback plugin.
VR playback requires access to Player Premium Version License（Web） for use.

Connection Method

For the player initialization process, see TCPlayer Integration Guide and API Documentation.
When initializing the player instance, you can enable VR playback by claiming plugins. Once enabled, the player will automatically load and use this plugin:

```
const player = TCPlayer('player-container-id', { // player-container-id is the play
  plugins: {
    VR: {
      isEnableController: true,
      ...
    },
  }
});
```

VR Plugin Property Description

Name	Description	Default Value
isEnableController	Enable VR Controller	true
isEnableZoom	Allow Image Scaling	true
yaw	Initialize left and right viewing angles, in degrees	0
pitch	Initialize vertical viewing angles, in degrees	0
fov	Initialize the field of view, in degrees	65
yawRange	Limit the range of view movement	[-180, 180]
pitchRange	Limit the range of view movement	[-90, 90]

fovRange	Limit the range of view movement	[30, 110]

VR Plugin Method Description

After the VR plugin is initialized, it generates an instance. After instantiation, it enters the VR pattern to play videos. The VR instance can be found on the player instance, and related methods can be called through the VR instance:

- lookAt

Move to a specific angle of view through animation over a period of time.

```
player.plugins['VR'].lookAt({ yaw: 30 }, 1000);
```

- setGyroMode

If your device has motion sensors (gyroscope, accelerometer), you can change the viewing angle through the device's motion. This method can be set to `'VR' | 'none' | 'yawPitch'`.

```
player.plugins['VR'].setGyroMode('none');
```

- enableSensor

Access permission to use motion sensors. Typically, on Android devices, motion sensors are enabled by default, while on iOS 13+, manual access permission is required through user interaction.

```
player.plugins['VR'].enableSensor();
```

Note:

1. In a browser hijacking environment, it is impossible to support the playback of VR videos.
2. After initialization, the Android player will default to the VR pattern and activate the gyroscope.
3. On the iOS side, performance may vary depending on the system version:
3.1 For system versions 13+, you need to manually click on the page to trigger user interaction and access permissions before the gyroscope can be activated.
3.2 For system versions 12.2 to 13, you need to go into the system Settings to manually enable the motion sensor. The usual path to do this is Settings > Safari > Motion & Orientation Access. After enabling the sensor, refresh the page to activate it.

Sample code

Click here to see the sample code.

Advanced Mobile Features
Picture-in-Picture Component (TUIPIP)
iOS

Last updated：2024-04-17 11:49:14

Introduction

Application scenarios

Advanced Picture-in-Picture (PIP) is an upgraded version of the basic PIP, mainly used for encrypted video PIP, offline playback PIP, and seamless switching from foreground to PIP scenes. It optimizes the implementation and logic, and achieves a true "instant switch" effect without long waiting times.
Advantages of Advanced PIP:
Encrypted video PIP: integrated with the existing player for encrypted playback, realizing PIP playback based on encrypted templates without the need to switch player types.
Offline playback PIP: supports local video PIP playback, including ordinary videos, encrypted videos, etc.
"Instant switch" effect: no need to click the PIP button to switch, just go to the background and PIP will start immediately, achieving a true "instant switch".

Requirements

System version: iOS ≥ 14.0, iPad ≥ 9.0 .
Hardware devices: iPhone 8 and above.
SDK version: 11.4 or above.

Integration Steps

Upgrade SDK version and configure resources

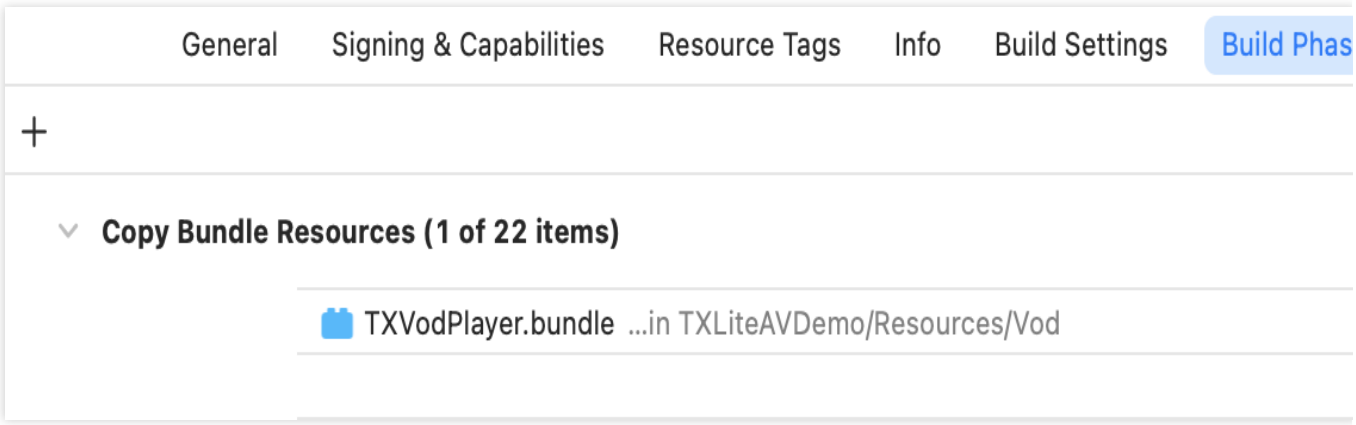
1. Upgrade SDK version:

Advanced Picture-in-Picture requires the use of the SDK. Before using the advanced Picture-in-Picture version, the SDK version needs to be upgraded to version 11.3 or above for the advanced version, or version 11.4 or above for the basic version, otherwise it cannot be used. At the same time, the basic and advanced versions of Picture-in-Picture

can coexist with compatibility and no functional conflicts. If you want to upgrade the SDK version, please refer to the SDK integration guide.

2. Introduce bundle resources

Because the SDK needs to use resources from TXVodPlayer.bundle, the bundle file needs to be downloaded and introduced into the project before compilation. Please do not modify the bundle or the names of the resources used inside, as doing so may cause seamless switching of picture-in-picture to fail.



3. Activate the premium version license for the player

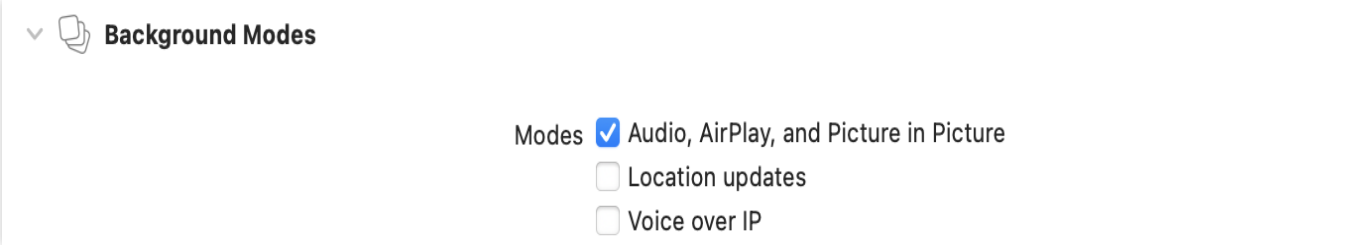
The advanced picture-in-picture version requires the mobile player premium version license. You can refer to the mobile player license guide to obtain it. If you have obtained the corresponding License, you can go to Tencent Cloud Cube Console > License Management > Mobile License to obtain the corresponding LicenseURL and LicenseKey. If you do not apply for the Player Advanced Package License, entering Picture-in-Picture will be invalid.
After obtaining the License information, before calling the relevant interface of the SDK, initialize the License through the following interface. It is recommended to make the following settings in - [AppDelegate application:didFinishLaunchingWithOptions:]:

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSD
    NSString * const licenceURL = @"<your license Url>";
    NSString * const licenceKey = @"<your license key>";

    //TXLiveBase is in "TXLiveBase.h"
    [TXLiveBase setLicenceURL:licenceURL key:licenceKey];
    NSLog(@"SDK Version = %@", [TXLiveBase getSDKVersionStr]);
}
```

Quick access to picture-in-picture function

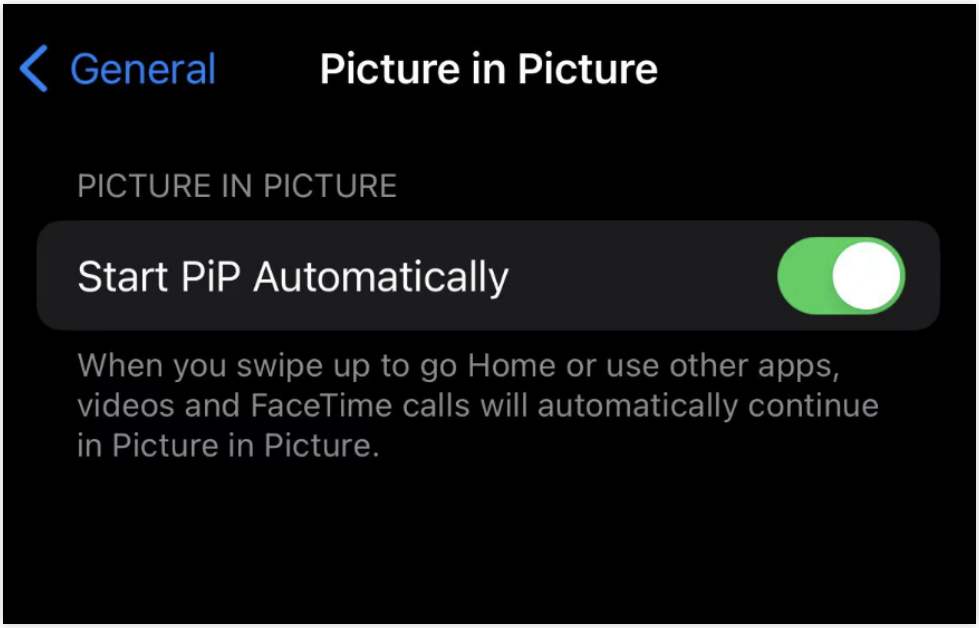
1. Permission granted

Picture-in-picture (PictureInPicture) has been launched in iOS 9, but it was only available on the iPad before. To use picture-in-picture, the iPhone needs to be updated to iOS 14. Currently, Tencent Cloud Player can support in-app and out-of-app picture-in-picture capabilities, which greatly meets the needs of users. You need to enable background mode before use. The steps are: Select the corresponding Target > Signing & Capabilities > Background Modes in XCode and check "Audio, AirPlay, and Picture in Picture".



2. Set configuration options

In order to use the automatic picture-in-picture function, you need to turn on the automatic picture-in-picture button in the settings. The specific path is to select on iPhone or iPad: Settings > General > Picture-in-Picture > Automatically turn on Picture-in-Picture, select to turn it on.



3. Set proxy

In order to facilitate monitoring of the picture-in-picture status, vodDelegate needs to be set to implement the picture-in-picture related callbacks in TXVodPlayListener. You can perform related business operations based on various status and error messages in the callback, such as continuing to play, pausing or exiting picture-in-picture, etc.

```
/**
 * Picture-in-picture status callback
 *
 */
- (void)onPlayer:(TXVodPlayer *)player pictureInPictureStateDidChange:(TX_VOD_PLAYE

/**
 * Picture-in-picture error message callback
 *
 */
- (void)onPlayer:(TXVodPlayer *)player pictureInPictureErrorDidOccur:(TX_VOD_PLAYER
```

4. Code example using picture-in-picture capability

Note：

To use the automatic picture-in-picture function, make sure the player is in the playing state. If the player is paused or stopped, the automatic picture-in-picture function cannot be used.

The isSupportSeamlessPictureInPicture interface can only be used after the application loads the advanced version License. At the same time, this interface can only determine whether the device itself supports automatic switching of picture-in-picture. Due to system limitations, it cannot determine the user's setting permission for automatic picture-in-picture, and needs to be guided by itself.

Before playing, set whether to allow "automatic switching of picture-in-picture function".

```
// 1. Before playing, set whether the "automatic switching Picture-In-Picture funct
// YES means allowed, NO means not allowed, the default is NO
[TXVodPlayer setPictureInPictureSeamlessEnabled:YES];

// 2. Enter picture-in-picture
 if (![TXVodPlayer isSupportPictureInPicture]) {
   //The device does not support picture-in-picture and exit directly.
    return;
 }

// Manually call to enter picture-in-picture
 [_vodPlayer enterPictureInPicture];

// 3. Exit the background operation. If the device supports seamless switching of p
// Note: The isSupportSeamlessPictureInPicture interface can only be used after the
// Whether automatic switching of picture-in-picture is supported? Due to system li
if ([self.vodplayer isSupportSeamlessPictureInPicture]) {
         // No processing
} else {
   // Pause playback
   [self.vodplayer pause];
```

```
}

// 4. Exit picture-in-picture
[_vodPlayer exitPictureInPicture];
```

TUIPlayerShortVideo iOS

Last updated：2024-06-17 17:06:27

Component Introduction

TUIPlayerShortVideo component is a short video component launched by Tencent Cloud with excellent performance, supporting ultra-fast first frame and smooth sliding of videos, and providing high-quality playback experience.
First frame opens in seconds: First frame time is one of the core indicators of short video applications, which directly affects the user's viewing experience. Short video components use technologies such as pre-playback, pre-download, player reuse and precise traffic control to achieve a high-quality playback experience with extremely fast first frame and smooth sliding, thereby increasing user playback volume and retention time.
Excellent performance: Through the optimization of player reuse and loading strategies, memory and CPU consumption are always kept at a low level while ensuring excellent smoothness.
Quick integration: The component encapsulates complex playback operations, provides a default playback UI, and supports both FileId and URL playback, which can be quickly integrated into your project at low cost.

Effect comparison

From the example video below you can see the difference before and after implementing the best short video strategy. There is an obvious first frame lag before optimization.
After optimization, the playback is smooth and the average playback time after optimization reaches 10 milliseconds - 30 milliseconds.

Not optimized for short videos	Optimized short video

TUIPlayerKit Download

TUIPlayerKit SDK and Demo can be downloaded by clicking here.

Integration Guide

1. Dependencies

The SDKs that TUIPlayerShortVideo depends on are：
TUIPlayerCore
TXLiteAVSDK ≥ 11.4
SDWebImage
Masonry

2. Environmental requirements

System version：≥ iOS 9.0
Development Environment：≥ Xcode 14.0 (It is recommended to use the latest version)

3. Integrated TUIPlayerCore

Unzip the downloaded TUIPlayerKit resource package, add the TUIPlayerCore.xcframework component SDK to the appropriate location of the Xcode Project in your project, select the appropriate target, and check Do Not Embed.

4. Integrated TUIPlayerShortVideo

Unzip the downloaded TUIPlayerKit resource package, add the TUIPlayerShortVideo.xcframework component SDK to the appropriate location of your Xcode Project, select the appropriate target, and check Do Not Embed.

5. Integrated TXLiteAVSDK

For TXLiteAVSDK integration methods, please refer to TXLiteAVSDK Integration Guide.

6. Integrated SDWebImage

For downloading and integrating SDWebImage, please refer to the GitHub instructions.

7. Integrated Masonry

For downloading and integrating Masonry, please refer to the GitHub instructions.

8. Pod Integration

If your project supports pod, you can also integrate it through the spec file we provide, as follows:

```
pod 'TUIPlayerShortVideo' ,:path => '../../../SDK/TUIPlayerShortVideoSDK/'
pod 'TUIPlayerCore' ,:path => '../../../SDK/TUIPlayerCoreSDK/'
```

Attention：

Please configure Path according to your own project file path.

Remote Pod integration is not currently supported.

Interface Instructions

1. Quick access

1.1. Configure Player Premium License

The use of TUIPlayer Kit components requires the use of the Mobile Player Premium License, which you can obtain by referring to the Mobile Player License Guide. If you have already obtained the corresponding license, you can go to Tencent Cloud Visual Cube Console > License Management > Mobile License to obtain the corresponding LicenseURL and LicenseKey. If you do not apply for the Mobile Player Premium License, video playback failures, black screens, etc. will occur.

Before calling related functions, configure Licence in your project. It is recommended to reference the TUIPlayerCore module in `- [AppDelegate application:didFinishLaunchingWithOptions:]` make the following configuration：

```
NSString * const licenceURL = @"<Obtained licenseUrl>";
NSString * const licenceKey = @"<The key obtained>";
[TXLiveBase setLicenceURL:licenceUrl key:licenceKey];
[[TXLiveBase sharedInstance] setDelegate:self];
```

1.2. Play

Initialize TUIShortVideoView, as follows:

```
- (TUIShortVideoView *)videoView {

    if (!_videoView) {
        ///Setting up a custom UI
        TUIPlayerShortVideoUIManager *uiManager = [[TUIPlayerShortVideoUIManager al
        [uiManager setControlViewClass: TUIPlayerShortVideoControlView.class];
        [uiManager setControlViewClass: TUIPSControlLiveView.class viewType:TUI_ITE
        [uiManager setControlViewClass: TUIPSControlCustomView.class viewType:TUI_I
        [uiManager setLoadingView:[[TUIPSDLoadingView alloc] init]];

        _videoView = [[TUIShortVideoView alloc] initWithUIManager:uiManager];
        _videoView.delegate = self;
        _videoView.customCallbackDelegate = self;
        //_videoView.isAutoPlay = NO;
```

```
        // Set your playback strategy
        TUIPlayerVodStrategyModel *model = [[TUIPlayerVodStrategyModel alloc] init]
        model.mPreloadConcurrentCount = 1;
        model.preDownloadSize = 1;
        model.enableAutoBitrate = NO;

        // live strategy
        TUIPlayerLiveStrateyModel *liveStrateyModel = [[TUIPlayerLiveStrateyModel a
        [_videoView setShortVideoLiveStrategyModel:liveStrateyModel];
    }
    return _videoView;
}
```

Add an instance of TUIShortVideoView to the View you want to present, as shown in the following code:

```
videoView.frame = self.view.bounds;
[self.view addSubview:self.videoView];
```

Then add your videos array:

```
TUIPlayerVideoModel *model1 = [[TUIPlayerVideoModel alloc] init]; ///Video Data
TUIPlayerLiveModel *model2 = [[TUIPlayerLiveModel alloc] init]; ///Live data
TUIPlayerDataModel *model3 = [[TUIPlayerDataModel alloc] init]; ///Custom Data
/// Here, you can decide the amount of data per page based on your business situati
NSArray *videos1 = @[model1,model2,model3];
[self.videoView setShortVideoModels:videos1];
```

After the first set of videos is played, you need to continue to insert your second set of video data in the TUIShortVideoViewDelegate delegate method:

```
TUIPlayerVideoModel *model1 = [[TUIPlayerVideoModel alloc] init]; ///Video Data
TUIPlayerLiveModel *model2 = [[TUIPlayerLiveModel alloc] init]; ///Live data
TUIPlayerDataModel *model3 = [[TUIPlayerDataModel alloc] init]; ///Custom Data
/// Here, you can decide the amount of data per page based on your business situati
NSArray *videos2 = @[model1,model2,model3];
-(void)onReachLast {
    ///Here you can make a data index record and continue to insert your 3rd 4th 5t
    [self.videoView appendShortVideoModels:videos2];
}
```

1.3. TUIShortVideoView

The main interfaces of TUIShortVideoView are as follows:

Parameter name	Implication
isAutoPlay	Whether to automatically play the first video when loading for the first time, default is YES
videos	Read-only property, get the data currently in the video list
currentVideoModel	The video model currently playing
currentVideoIndex	Index of the video currently playing
currentPlayerStatus	The current player's playback status
isPlaying	Is the current player playing
delegate	delegate
refreshControl	Set the pull-down refresh control
initWithUIManager	Initialization
setShortVideoStrategyModel	Set the live broadcast strategy
setShortVideoLiveStrategyModel	Set the live broadcast strategy
setShortVideoModels	Setting up a data source for the first time
appendShortVideoModels	Add video data source
removeAllVideoModels	Delete all video data
setPlaymode	Video playback mode, single loop or list loop, the former is the default
pause	Pause
resume	Resume
destoryPlayer	Destroy the player
didScrollToCellWithIndex	Jump to the video with the specified index
startLoading	Display loading image
stopLoading	Hide loading icon
currentPlayerSupportedBitrates	The bitrate supported by the currently playing video
bitrateIndex	Get the bitrate index of the current playback
switchResolution:index:	Switch resolution

pausePreload	Pause preloading
resumePreload	Resume preload
getDataManager	Get Data Manager
getVodStrategyManager	Get the On-Demand Policy Manager
getLiveStrategyManager	Get the Live Strategy Manager

2. Global Configuration

You can set some global configurations in TUIPlayerCore via the TUIPlayerConfig model.

The main parameters of TUIPlayerConfig are shown in the table below:

Parameter name	Implication
enableLog	Whether to allow printing logs, the default is NO

Then configure globally through TUIPlayerCore:

```
TUIPlayerConfig *config = [TUIPlayerConfig new];
config.enableLog = YES;
[[TUIPlayerCore shareInstance] setPlayerConfig:config];
```

3. Player policy configuration

3.1. VOD Playback Policy Settings

You can use the TUIPlayerVodStrategyModel model to configure the on-demand playback strategy.

The main parameters of TUIPlayerVodStrategyModel are shown in the following table:

Parameter name	Implication
mPreloadConcurrentCount	The number of video caches, default is 3
mPreloadBufferSizeInMB	Pre-play size, in MB, default 0.5MB
preferredResolution	Preferred resolution, default 720 * 1280
progressInterval	The progress bar callback interval, in milliseconds, default is 500ms
renderMode	Canvas fill style, the default image adapts to the screen to keep the picture intact
extInfoMap	Additional parameters, reserved

enableAutoBitrate	Whether to enable adaptive bitrate, default is NO
mediaType	Set the media type
maxBufferSize	Maximum preload size, in MB, default 10MB, this setting will affect playableDuration, the larger the setting, the more pre-cached
mResumeModel	Resume mode, default value is TUI_RESUM_MODEL_NONE
preDownloadSize	Pre-download size, in MB, default 1MB
enableAccurateSeek	Whether to seek accurately, the default is YES. After turning on accurate seek, the seek time will be 200ms longer on average
audioNormalization	Volume balance. Loudness range: -70～0 (LUFS). This configuration requires LiteAVSDK 11.7 and above. The following constants are for reference: Off: AUDIO_NORMALIZATION_OFF (TXVodPlayConfig.h) On (standard loudness): AUDIO_NORMALIZATION_STANDARD (TXVodPlayConfig.h) On (low loudness): AUDIO_NORMALIZATION_LOW (TXVodPlayConfig.h) On (high loudness): AUDIO_NORMALIZATION_HIGH (TXVodPlayConfig.h) The default value is AUDIO_NORMALIZATION_OFF.
isLastPrePlay	Whether to keep the last pre-play, the default is NO
subtitleRenderModel	Subtitle style

Then configure the player strategy：

```
TUIPlayerStrategyModel *model = [[TUIPlayerStrategyModel alloc] init];
model.mPreloadConcurrentCount = 1;
model.preDownloadSize = 1;
model.enableAutoBitrate = NO;
model.mRenderMode = TUI_RENDER_MODE_FILL_SCREEN;
model.mResumeModel = TUI_RESUM_MODEL_LAST;
[_videoView setShortVideoStrategyModel:model];
```

3.2. Live broadcast strategy settings

You can use the TUIPlayerLiveStrategyModel model to configure the on-demand playback strategy.

The main parameters of TUIPlayerLiveStrategyModel are shown in the following table:

Parameter name	Implication

isLastPrePlay	Whether to keep the last pre-play, the default is NO
mRenderMode	Canvas fill style, default V2TXLiveFillModeFill
enablePictureInPicture	YES: Enable the PIP function; NO: Disable the PIP function. Default value: NO.
volume	The volume of the player, ranging from 0 to 100. Default value: 100.
maxAutoAdjustCacheTime	The maximum time for automatic adjustment of the player cache, in seconds. The value must be greater than 0. The default value is 5.
minAutoAdjustCacheTime	The minimum time for automatic adjustment of the player cache, in seconds. The value must be greater than 0. The default value is 1.
isShowDebugView	Whether to display the debug overlay of player status information. Default value: NO.

Then configure the player strategy：

```
TUIPlayerLiveStrategyModel *liveStrategyModel = [[TUIPlayerLiveStrategyModel alloc]
[_videoView setShortVideoLiveStrategyModel:liveStrategyModel];
```

3.3. Dynamic policy adjustment

Both on-demand and live broadcast strategies support dynamic adjustment. The steps are as follows：

1、Get the live broadcast & on-demand strategy management class through TUIShortVideoView.

```
TUIPlayerVodStrategyManager *VodStrategyManager = [_videoView getVodStrategyManager
TUIPlayerVodStrategyManager *LiveStrategyManager = [_videoView getLiveStrategyManag
```

2、Adjust playback strategies through VodStrategyManager and LiveStrategyManager.

```
[VodStrategyManager setRenderMode:TUI_RENDER_MODE_FILL_EDGE];
[LiveStrategyManager setRenderMode:V2TXLiveFillModeFill];
```

4.Data Management

4.1. Data Model

The original data model of TUIShortVideoView includes:

Parameter name	Implication

TUIPlayerDataModel	Basic data types
TUIPlayerVideoModel	Video data type, inherited from TUIPlayerDataModel
TUIPlayerLiveModel	Vive data type, inherited from TUIPlayerDataModel

TUIPlayerDataModel

Parameter name	Implication
modelType	Model Type
extInfo	Business data
onExtInfoChangedBlock	ExtInfo The block where the data has changed
extInfoChangeNotify	Notify extInfo that data has changed
asVodModel	Force conversion to TUIPlayerVideoModel type
asLiveModel	Force conversion to TUIPlayerLiveModel type

TUIPlayerVideoModel

Parameter name	Implication
videoUrl	Video URL
coverPictureUrl	Cover picture url
duration	duration
appId	appid
fileId	fileId
pSign	Signature String
subtitles	Subtitle information
config	Separate configuration of video, see TUIPlayerVideoConfig for details

TUIPlayerLiveModel

Parameter name	Implication
liveUrl	Live URL

coverPictureUrl	cover picture url

4.2. Model Construction

Build a set of on-demand data models

```
TUIPlayerVideoModel *model = [[TUIPlayerVideoModel alloc] init];
model.videoUrl = @"xxxx";
model.coverPictureUrl = @"xxxx";
model.duration = @"xxxx";
model.appId = @"xxxx";
model.fileId = @"xxxx";
model.pSign = @"xxxx";
NSDictionary *extr = @{
    @"name":@"@Mars",
    @"titile":@"This is a vod broadcast interface",
    @"des":@"This is a vod broadcast interface"
};
model.extInfo = extr;
[modelArray addObject:model];
```

Build a set of live data models

```
TUIPlayerLiveModel *model = [[TUIPlayerLiveModel alloc] init];
model.liveUrl = @"xxxx";
model.coverPictureUrl = @"xxxx";
NSDictionary *extr = @{
    @"name":@"@Mars",
    @"liveTitile":@"This is a live broadcast interface",
    @"liveDes":@"This is a live broadcast interface"
};
model.extInfo = extr;
```

Build a set of other types of data models

```
/// 1 Carousel
TUIPlayerDataModel *model = [[TUIPlayerDataModel alloc] init];
NSDictionary *extr = @{
    @"images":@"xxxx",
    @"url":@"https://cloud.tencent.com",
    @"titile":@"This is a picture carousel display interface",
    @"des":@"This is a picture carousel display interface",
    @"name":@"@Mars",
    @"type":@"imageCycle"
```

```
};
model.extInfo = extr;
[modelArray insertObject:model atIndex:1];

/// 2 Graphic Ads
TUIPlayerDataModel *model1 = [[TUIPlayerDataModel alloc] init];
NSDictionary *extr1 = @{
    @"adUrl":@"https://cloud.tencent.com",
    @"adUrl":@"https://cloud.tencent.com/document/product",
    @"adTitile":@"This is a web display interface",
    @"adDes":@"This is a web display interface",
    @"name":@"@Mars",
    @"type":@"ad"
};
model1.extInfo = extr1;
[modelArray insertObject:model1 atIndex:1];
```

Attention：

TUIPlayerDataModel is a large class that applies to all non-on-demand and live data types.

Users can use extInfo to perform more detailed classification. For example, in the figure above, two types of data, "carousel" and "graphic advertisement", are constructed through TUIPlayerDataModel, which can be classified by extInfo/type.

extInfo is a flexible word, and users can design the data structure they need at will.

4.3. Dynamic adjustment of data

TUIShortVideoView provides a data management class TUIShortVideoDataManager for external data operations. Its main function is to perform basic operations such as adding, deleting, modifying, and checking the data in the current player list. See the following demonstration code:

```
///1、Delete the data and view at index 1
[[self.videoView getDataManager] removeData:1];
///2、Add a set of data to index 9
TUIPlayerVideoModel *model = [[TUIPlayerVideoModel alloc] init];
model.viewType = TUI_ITEM_VIEW_TYPE_CUSTOM;
[[self.videoView getDataManager]  addData:model index:9];
```

A more detailed interface description is as follows:

Parameter name	Implication
removeData	Remove data by index
removeRangeData	Remove data by range
removeDataByIndex	Remove data by index array

addData:index	Add data by index
addRangeData:startIndex	Add data from a certain index according to the model array
replaceData:index	Replace data by index
replaceRangeData:startIndex	Replace data from a certain index in the model array
getDataByPageIndex	Read data at a certain index
getCurrentDataCount	Get the total number of data in the current playlist
getCurrentIndex	Get the data index of the current playback interface
getCurrentModel	Get the data model of the current playback interface

Explanation：

The UI interface is automatically refreshed after the DataManager interface is called.

If the current playback interface is not operated, there will be no refresh.

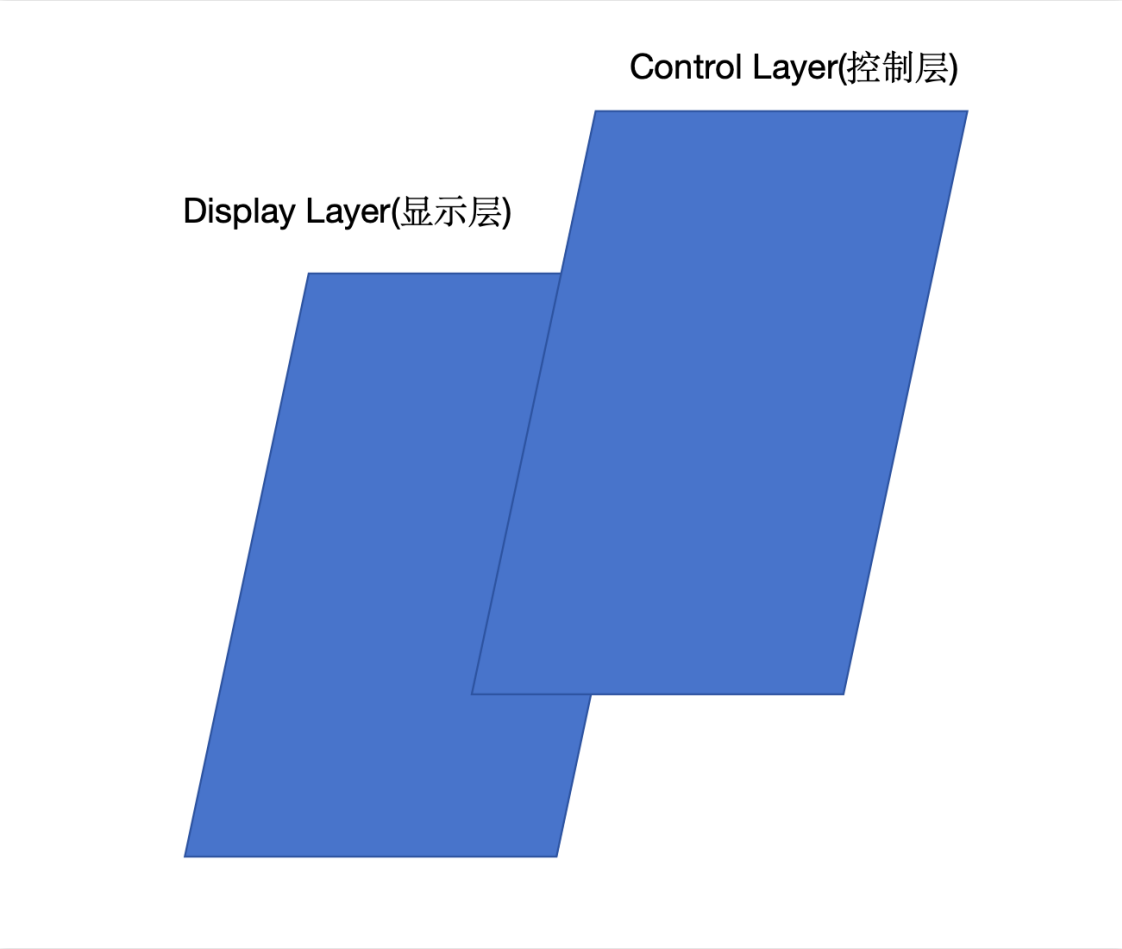
Operating the current interface will refresh the current interface.

If the current playback interface is deleted, the next one will be played automatically. If there is no data for the next one (it has reached the end), the previous one will be played.

5. Custom UI Layers

5.1. Hierarchy

The hierarchical structure of TUIPlayerShortVideo is as follows:



It is divided into a display layer and a control layer, and the two are combined in a stacked manner.

The display layer is responsible for content display, on-demand, live broadcast, advertising promotion pages, etc. This layer is managed internally by the SDK.

The control layer is responsible for interaction, likes, comments, etc. This layer is left to the user to achieve a high degree of customization.

5.2. TUIPlayerShortVideoUIManager

You can use the TUIPlayerShortVideoUIManager interface and the protocol control layer interface to implement your custom UI. See the following code:

```
TUIPlayerShortVideoUIManager *uiManager = [[TUIPlayerShortVideoUIManager alloc] ini
[uiManager setControlViewClass: TUIPSControlView.class];
[uiManager setLoadingView:[[TUIPSLoadingView alloc] init]];
[uiManager setBackgroundView:[UIView new]];
_videoView = [[TUIShortVideoView alloc] initWithUIManager:uiManager];
```

The above code customizes the video control layer (progress bar, time, etc.) named TUIPSControlView and the loading control named TUIPSLoadingView through TUIPlayerShortVideoUIManager.

The interface of TUIPlayerShortVideoUIManager is shown in the following table:

Parameter name	Implication
setLoadingView	Setting up the loading graph
setBackgroundView	Set the background image
setErrorView	Setting the error interface
setControlViewClass	Set the video control layer * @param ViewClass control layer class, ViewClass is the video control View you have encapsulated, including controls such as progress bar, time label, etc. * It will be covered on the video window as a whole, and its size is consistent with the video window
setControlViewClass :viewType	Set up different types of video control layers
getLoadingView	Get the loading image View instance
getBackgroundView	Get the background image View instance
getErrorView	Get the error interface View instance
getControlViewClass	Get the video control interface View class
getControlViewClassWithViewType	Get different types of video control interface classes

The playback layer currently supports two types:
TUI_ITEM_VIEW_TYPE_VOD ///video
TUI_ITEM_VIEW_TYPE_LIVE ///live
TUI_ITEM_VIEW_TYPE_CUSTOM /// Custom type (such as advertising page)
The corresponding control layer protocol also supports two types:
TUIPlayerShortVideoControl ///Video Control Layer Protocol
TUIPlayerShortVideoLiveControl ///Live broadcast control layer protocol
TUIPlayerShortVideoCustomControl ///Custom control layer protocol
explanation:
All non-VOD and live interfaces are presented with TUI_ITEM_VIEW_TYPE_VOD and TUIPlayerShortVideoCustomControl. Custom is a large category, and the specific subdivisions need to be defined by the user on this whiteboard. As mentioned in the TUIPlayerDataModel model construction above, more fine-grained divisions can be made through extInfo/type or other fields under extInfo.
The specific interface description of TUIPlayerShortVideoControl protocol is as follows:

Parameter name	Implication

delegate	A reverse proxy for interaction between the control layer and the playback layer
model	The currently playing video model
currentPlayerStatus	The current player's playback status
showCenterView	Display center view
hideCenterView	Hide center view
showLoadingView	Display loading graph
hiddenLoadingView	Hide loading image
setDurationTime	Total video time
setCurrentTime	Current playback time
setProgress	Progress bar progress
showSlider	Show progress bar
hideSlider	Hide progress bar
reloadControlData	Triggering a view refresh
getPlayer	Get the player object
onPlayEvent	Get player events
getVideoLayerRect	Get the changes of the video rendering area
getVideoWidget	Get the video rendering layer object

The specific interface description of TUIPlayerShortVideoLiveControl protocol is as follows:

Parameter name	Implication
delegate	A reverse proxy for interaction between the control layer and the playback layer
model	Current playback data model
reloadControlData	Triggering a view refresh
getPlayer	Get the player object
getVideoLayerRect	Get the video layer area

getVideoWidget	Get the video rendering layer

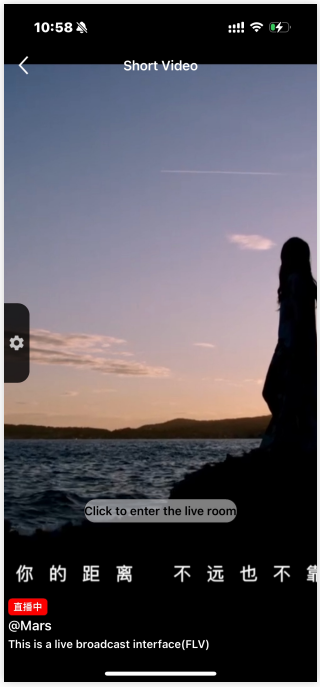
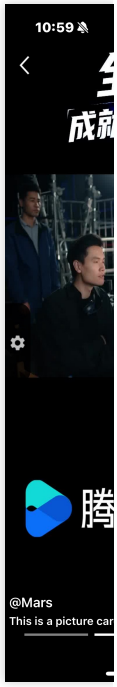
The specific interface description of TUIPlayerShortVideoCustomControl protocol is as follows:

Parameter name	Implication
delegate	A reverse proxy for interaction between the control layer and the playback layer
model	Current playback data model
reloadControlData	Triggering a view refresh

Attention：

The custom control layer View passed in needs to comply with the relevant protocols, otherwise the compilation will fail.

5.3. Example

Type	video	live	Custom (slidesh
example			

5.3.1. Define UI styles under different styles

vod style (TUIPlayerShortVideoControl)

```
@interface TUIPSControlView : UIView<TUIPlayerShortVideoControl>
@property (nonatomic, strong) TUIPlayerVideoModel *videoModel
@end
@implementation TUIPSControlView

-(instancetype)initWithFrame:(CGRect)frame {
    if ([super initWithFrame:frame]){
        /// UI layout code
    }
    return self;
}

-(void)setModel:(TUIPlayerVideoModel *)model {
    _model = model;
    /// data
}
@end
```

Live broadcast style (TUIPlayerShortVideoLiveControl)

```
@interface TUIPSControlLiveView : UIView<TUIPlayerShortVideoLiveControl>
@property (nonatomic, strong) TUIPlayerLiveModel *videoModel
@end
@implementation TUIPSControlLiveView
-(instancetype)initWithFrame:(CGRect)frame {
    if ([super initWithFrame:frame]){
        /// UI layout code
    }
    return self;
}

-(void)setModel:(TUIPlayerLiveModel *)model {
    _model = model;
    /// data
}
@end
```

Other styles such as carousel & graphic ads (TUIPlayerShortVideoCustomControl)

```
@interface TUIPSControlCustomView : UIView<TUIPlayerShortVideoCustomControl>
@property (nonatomic, strong) TUIPlayerDataModel *videoModel
@end
@implementation TUIPSControlCustomView
```

```
-(instancetype)initWithFrame:(CGRect)frame {
    if ([super initWithFrame:frame]){
        /// UI layout code
    }
    return self;
}

-(void)setModel:(TUIPlayerDataModel *)model {
    _model = model;
    /// data

    NSDictionary *dic = model.extInfo;
    NSString *adTitile = [dic objectForKey:@"adTitile"];
    NSString *adDes = [dic objectForKey:@"adDes"];
    NSString *adUrl = [dic objectForKey:@"adUrl"];
    NSString *name = [dic objectForKey:@"name"];
    NSString *type = [dic objectForKey:@"type"];

    if ([type isEqualToString:@"ad"]) {  ///Graphic Ads
        self.webView.hidden = NO;
        self.cycleScrollView.hidden = YES;
        self.desLabel.textColor = [UIColor blackColor];
        self.nameLabel.textColor = [UIColor blackColor];
        [self.webView loadRequest:[NSURLRequest requestWithURL:[NSURL URLWithString
    } else if ([type isEqualToString:@"imageCycle"]) { ///Carousel
        self.webView.hidden = YES;
        self.cycleScrollView.hidden = NO;
        self.desLabel.textColor = [UIColor whiteColor];
        self.nameLabel.textColor = [UIColor whiteColor];
        NSString *imagesStr = [dic objectForKey:@"images"];
        NSArray *imagesArray = [imagesStr componentsSeparatedByString:@"<:>"];
        self.cycleScrollView.imageURLStringsGroup = imagesArray;
    }

}
@end
```

5.3.2. Register the created style through TUIPlayerShortVideoUIManager

```
TUIPlayerShortVideoUIManager *uiManager = [[TUIPlayerShortVideoUIManager alloc] ini
[uiManager setControlViewClass: TUIPSControlView.class viewType:TUI_ITEM_VIEW_TYPE_
[uiManager setControlViewClass: TUIPSControlLiveView.class viewType:TUI_ITEM_VIEW_T
[uiManager setControlViewClass: TUIPSControlCustomView.class viewType:TUI_ITEM_VIEW
```

5.3.2. Initialize TUIShortVideoView through TUIPlayerShortVideoUIManager

```
_videoView = [[TUIShortVideoView alloc] initWithUIManager:uiManager];
```

5.3.3. Use setShortVideoStrategyModel and setShortVideoLiveStrategyModel to set the strategies for video on demand and live streaming.

```
// Set your playback strategy
 TUIPlayerVodStrategyModel *model = [[TUIPlayerVodStrategyModel alloc] init];
 model.mPreloadConcurrentCount = 1;
 model.preDownloadSize = 1;
 model.enableAutoBitrate = NO;
[_videoView setShortVideoStrategyModel:model];

 // live strategy
 TUIPlayerLiveStrateyModel *liveStrateyModel = [[TUIPlayerLiveStrateyModel alloc] i
 [_videoView setShortVideoLiveStrategyModel:liveStrateyModel];
```

5.3.4. The relationship between UI and data

TUIPSControlView & TUIPSControlLiveView & TUIPSControlCustomView can be understood as preset templates. When TUIShortVideoView is initialized, various types of templates have been preset. When the corresponding data type is slipped, the current template will be displayed.

Type	Data model	UI Templates
Vod	TUIPlayerVideoModel	TUIPSControlView
Live	TUIPlayerLiveModel	TUIPSControlLiveView
Custom types (carousel, graphic ads, etc.)	TUIPlayerDataModel	TUIPSControlCustomView

UI templates are pre-set
The UI style is displayed according to the driver of the corresponding data type.

5.3.4. Interaction between UI template and TUIShortVideoView

The custom UI template interacts with TUIShortVideoView through the corresponding protocol. TUIShortVideoView passes messages to TUIPSControlView & TUIPSControlLiveView & TUIPSControlCustomView through the protocol methods of TUIPlayerShortVideoControl &TUIPlayerShortVideoLiveControl &TUIPlayerShortVideoCustomControl.
As VOD:

```
-(void)setModel:(TUIPlayerVideoModel *)model {

    if ([_model observationInfo]) {
```

```
        [_model removeObserver:self forKeyPath:@"preloadState"];

    }
    _model = model;
    [model addObserver:self forKeyPath:@"preloadState" options:NSKeyValueObservingO


    NSDictionary *dic = model.extInfo;
    NSString *iconUrl = [dic objectForKey:@"iconUrl"];
    NSString *advertise = [dic objectForKey:@"advertise"];
    NSString *name = [dic objectForKey:@"name"];
    NSString *title = [dic objectForKey:@"title"];
    NSString *topic = [dic objectForKey:@"topic"];
    self.iconImageView.image = [UIImage imageNamed:iconUrl];
    [self.adButton setTitle:advertise forState:UIControlStateNormal];
    self.nameLabel.text= name;
    self.themeLabel.text = topic;
    self.desLabel.text = title;
    [self updatePreloadState];
    [self updateLickCount];
    model.onExtInfoChangedBlock = ^(id  _Nonnull extInfo) {
        [self updateLickCount];
    };
}
```

Live：

```
-(void)setModel:(TUIPlayerLiveModel *)model {

    _model = model;
    NSDictionary *dic = model.extInfo;
    NSString *adTitile = [dic objectForKey:@"liveTitile"];
    NSString *adDes = [dic objectForKey:@"liveDes"];
    NSString *name = [dic objectForKey:@"name"];

    self.nameLabel.text = name;
    self.desLabel.text = adTitile;
}
```

Custom (carousel, pictures and text, etc.)

```
-(void)setModel:(TUIPlayerDataModel *)model {
```

```
    _model = model;
    NSDictionary *dic = model.extInfo;
    NSString *adTitile = [dic objectForKey:@"adTitile"];
    NSString *adDes = [dic objectForKey:@"adDes"];
    NSString *adUrl = [dic objectForKey:@"adUrl"];
    NSString *name = [dic objectForKey:@"name"];
    NSString *type = [dic objectForKey:@"type"];
    self.desLabel.text = adTitile;
    self.nameLabel.text = name;
    if ([type isEqualToString:@"web"]) {
        self.webView.hidden = NO;
        self.cycleScrollView.hidden = YES;
        self.desLabel.textColor = [UIColor blackColor];
        self.nameLabel.textColor = [UIColor blackColor];
        [self.webView loadRequest:[NSURLRequest requestWithURL:[NSURL URLWithString
    } else if ([type isEqualToString:@"imageCycle"]) {
        self.webView.hidden = YES;
        self.cycleScrollView.hidden = NO;
        self.desLabel.textColor = [UIColor whiteColor];
        self.nameLabel.textColor = [UIColor whiteColor];
        NSString *imagesStr = [dic objectForKey:@"images"];
        NSArray *imagesArray = [imagesStr componentsSeparatedByString:@"<:>"];
        self.cycleScrollView.imageURLStringsGroup = imagesArray;
    }
}
```

TUIPSControlView & TUIPSControlLiveView & TUIPSControlCustomView pass messages to TUIShortVideoView through the delegate of TUIPlayerShortVideoControl &TUIPlayerShortVideoLiveControl &TUIPlayerShortVideoCustomControl.

As VOD：

```
@protocol TUIPlayerShortVideoControlDelegate <NSObject>

/**
 * Pause
 */
- (void)pause;
/**
 * Resume
 */
- (void)resume;

/**
 * Sliding scroll bar processing
```

```
 * @param time  Slide distance
 */
- (void)seekToTime:(float)time;

/**
 * Is it playing
 */
- (BOOL)isPlaying;

/**
 * Reset video playback container
 * Used for scenarios where the video playback container needs to be reset after be
 */
- (void)resetVideoWeigetContainer;

@optional
/**
 * Custom callback events
 */
- (void)customCallbackEvent:(id)info;
@end


/////transfer
if (self.delegate && [self.delegate respondsToSelector:@selector(pause)]) {
    [self.delegate pause];
}
```

Live

```

@protocol TUIPlayerShortVideoLiveControlDelegate <NSObject>

/**
 * pause
 */
- (void)pause;

/**
 * resume
 */
- (void)resume;

/**
 * Reset video playback container
 * Used for scenarios where the video playback container needs to be reset after be
 */
```

```
- (void)resetVideoWeigetContainer;

@optional
/**
 * Custom callback events
 */
- (void)customCallbackEvent:(id)info;
@end

/////transfer
if (self.delegate && [self.delegate respondsToSelector:@selector(pause)]) {
    [self.delegate pause];
}
```

Custom (carousel, pictures and text, etc.)

```
@protocol TUIPlayerShortVideoCustomControlDelegate <NSObject>

@optional
/**
 * Custom callback events
 */
- (void)customCallbackEvent:(id)info;
@end

/////transfer
if (self.delegate && [self.delegate respondsToSelector:@selector(customCallbackEven
    [self.delegate customCallbackEvent:@"test"];
}
```

explanation：

See Demo for the complete example.

Advanced Features

Business notification message to the page layer

TUI provides a message interface for users to notify the current layer of data in real time. After obtaining the video object through the data operation object, the notification can be made. The example is as follows:

```
/// Get Data Manager
TUIShortVideoDataManager *dataManager = [self.videoView getDataManager];
///Get the data model
TUIPlayerDataModel *model = [dataManager getDataByPageIndex:1];
```

```
///Modify the data model
model.extInfo = @{@"key":@"value"}
///Notify data model changes
[model extInfoChangeNotify];
```

The notification is then received in the onExtInfoChanged callback of the UI control layer, so that the UI of the current page can be modified. The example is as follows:

```
model.onExtInfoChangedBlock = ^(id  _Nonnull extInfo) {
      [self updateLickCount];
};
```

Attention：

This function is only valid for the extInfo field.

Volume balance on demand

The player supports automatic volume adjustment when playing audio, so that the volume of all audio is consistent. This can avoid the problem of some audio being too loud or too quiet, and provide a better listening experience.

By setting the volume balance, the loudness range is: -70～0 (LUFS), and custom values are also supported.

Attention：

Supported by Player Premium 11.7.

```
/// Volume balance. Loudness range: -70～0(LUFS). This configuration requires LiteAV
/// The following constants are for reference
/// Off: AUDIO_NORMALIZATION_OFF (TXVodPlayConfig.h)
/// On (standard loudness): AUDIO_NORMALIZATION_STANDARD (TXVodPlayConfig.h)
/// On (low loudness): AUDIO_NORMALIZATION_LOW (TXVodPlayConfig.h)
/// On (high loudness): AUDIO_NORMALIZATION_HIGH (TXVodPlayConfig.h)
/// The default value is AUDIO_NORMALIZATION_OFF.

TUIPlayerVodStrategyModel *model = [[TUIPlayerVodStrategyModel alloc] init];
model.audioNormalization = AUDIO_NORMALIZATION_STANDARD;

[_videoView setShortVideoStrategyModel:model];
```

Android

Last updated：2025-02-08 11:47:02

Component Introduction

The TUIPlayerShortVideo component is an excellent-performance short video component launched by Tencent Cloud, which supports ultra-fast first frame display and smooth sliding, providing a high-quality playback experience.
Instant First Frame: The time to the first frame is one of the core indicators for short video applications, directly affecting the user's viewing experience. The short video component achieves ultra-fast first frame and smooth sliding playback through technologies such as pre-playing, pre-downloading, player reuse, and precise traffic control, thereby enhancing user playback volume and dwell time.
Excellent Performance: Through player reuse and optimization of loading strategies, while ensuring excellent smoothness, memory and CPU consumption are always kept at a low level.
Rapid Integration: The component encapsulates complex playback operations, provides a default playback UI, and supports both FileId and Url playback, allowing for low-cost and rapid integration into your project.

Effect Comparison

The following video demonstrates the comparative differences in short video usage under the same environment, before and after optimization.
Before optimization, there is a noticeable lag when starting the video playback.
After optimization, a seamless startup experience can be achieved, with the average startup duration after optimization reaching 10 milliseconds to 30 milliseconds.

Unoptimized Short Video	Optimized Short Video

TUIPlayerKit Download

The TUIPlayerKit SDK and Demo can be downloaded by clicking here.

Integrate the TUIPlayerShortVideo Component

Environment Preparation

Minimum Android system version required: Android SDK >= 19

Add the dependencies required for short videos:

```
// If you are using the professional version of the SDK,
// use: api 'com.tencent.liteav:LiteAVSDK_Professional:latest.release'
api 'com.tencent.liteav:LiteAVSDK_Player:latest.release'
implementation (name:'tuiplayercore-release_x.x.x', ext:'aar')
implementation (name:'tuiplayershortvideo-release_x.x.x', ext:'aar')
implementation 'androidx.appcompat:appcompat:1.0.0'
implementation 'androidx.viewpager2:viewpager2:1.0.0'
```

Note：

Where the `x.x.x` in `tuiplayercore-release` and `tuiplayershortvideo-release` represents the version number, note that the version numbers of the two aar files must match.

Permissions Required by the SDK:

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

Set ProGuard rules:

In the proguard-rules.pro file, add the relevant classes to the list of classes not to be obfuscated:

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

Apply for the Advanced Player License

To use the TUIPlayer Kit component, you need to use the Advanced Mobile Player License. You can refer to the advanced version of the mobile player License to obtain it. If you have already obtained the corresponding License, you can go to Tencent Cloud Visual Cube Console > License Management > Mobile License to get the corresponding LicenseURL and LicenseKey. If you have not applied for the Advanced Mobile Player License, issues such as video playback failure and black screen may occur.

Set Up License

The short video component needs to have a License set up before it can be used.

```
TUIPlayerConfig config = new TUIPlayerConfig.Builder()
        .enableLog(true)
```

```
        .licenseKey("Your license key")
        .licenseUrl("Your license url")
        .build();
TUIPlayerCore.init(context, config);
```

In your layout file

add the short video UI component.

```
<com.tencent.qcloud.tuiplayer.shortvideo.ui.view.TUIShortVideoView
    android:id="@+id/my_video_view"
    android:layout_height="match_parent"
    android:layout_width="match_parent"/>
```

Set up the lifecycle

After setting up the lifecycle, the component will automatically pause, resume playback, and release itself based on the current Activity lifecycle. For example, when the app goes to the background, the short video will pause playback automatically. When you return to the app, the video will continue playing. You can also choose not to set this lifecycle and control the short video according to your business needs.

```
mSuperShortVideoView.setActivityLifecycle(getLifecycle());
```

Configure listening

The TUIShortVideoView listener offers multiple callbacks, as illustrated in the code below:

```
mSuperShortVideoView.setListener(new TUIShortVideoListener() {
  @Override
  public void onPageChanged(int index, TUIVideoSource videoSource) {
      if (index >= mSuperShortVideoView.getCurrentDataCount() - 1) {
        // append next page data
        mSuperShortVideoView.appendModels(data);
      }
  }

  @Override
  public void onCreateVodLayer(TUIVodLayerManager layerManger, int viewType) {
      // add your vod layer to here
      layerManger.addLayer(new TUICoverLayer());
  }

  @Override
  public void onCreateLiveLayer(TUILiveLayerManager layerManager, int viewType) {
```

```
        // add your live layer to here
    }

    @Override
    public void onCreateCustomLayer(TUICustomLayerManager layerManager, int viewType)
        // add your custom layer to here
    }

    // Optional
    @Override
    public void onNetStatus(TUIVideoSource model, Bundle bundle) {
    }

     // Optional
    @Override
    public void onPageScrolled(int layOutPos, float positionOffset, int positionOffse
    }

    // Optional
    @Override
    public void onVodPlayerReady(ITUIVodPlayer player, TUIVideoSource source) {
    }
  });
```

Whenever the page position changes, the `onPageChanged` method will be called back, where you can implement capabilities similar to paginated loading.

When the list creates its layout, it will call back the `onCreateVodLayer` , `onCreateLiveLayer` , or `onCreateCustomLayer` methods based on the type of data you add. Specifically, if the data is of type `TUIVideoSource` when setting models, `onCreateVodLayer` will be called back; if it's of type `TUILiveSource` , `onCreateLiveLayer` will be called back; and if it's a custom data type implemented by inheriting from `TUIPlaySource` , `onCreateCustomLayer` will be called back.

Here, the `extViewType` from `TUIPlayerSource` will also be called back, which can be used for the business to distinguish different layer groups based on the viewType. The creation of layers will be discussed in the section on custom layers.

After the video starts playing, the network status of the currently playing video will be called back through `onNetStatus` . For details on the callback, please refer to here.

Populate data

Using `setModels` can set the data and clear the original data. The video will also restart playback from the first video of the newly set data source. Using `appendModels` can append data, which is used for pagination operations. **After filling the data, it will automatically start playing from the first video. If automatic playback is not needed, you can call `setAutoPlay` on the first video and set it to `false` .**

```
// vod, you can inherit TUIVideoSource to extend attributes.
TUIVideoSource videoSource = new TUIVideoSource();
videoSource.setCoverPictureUrl(model.placeholderImage);
// Fill in the fileId. Only one of fileId and url needs to be filled in.
videoSource.setAppId(model.appid);
videoSource.setPSign(model.pSign);
videoSource.setFileId(model.fileid);
// Fill in the fileId url
videoSource.setVideoURL(model.videoURL);
shortVideoData.add(videoSource);

// live, you can inherit TUILiveSource to extend attributes.
TUILiveSource liveSource = new TUILiveSource();
// live url
liveSource.setUrl(liveUrl);
liveSource.setCoverPictureUrl(coverUrl);
shortVideoData.add(liveSource);

// Non-player pages, such as images, advertisements, etc.,
// where different data can be customized according to business needs,
// just by inheriting TUIPlaySource.
DemoImgSource imgSource = new DemoImgSource("imgUrl");
shortVideoData.add(imgSource);

// set data
mSuperShortVideoView.setModels(shortVideoData);

// While setting the initial data, you can specify the initial position;
// an index exceeding the collection size will be invalid.
mSuperShortVideoView.setModels(shortVideoData, index);

// For pagination operations, you can choose to append data.
mSuperShortVideoView.appendModels(shortVideoData);
```

Custom Layer

I. Introduction

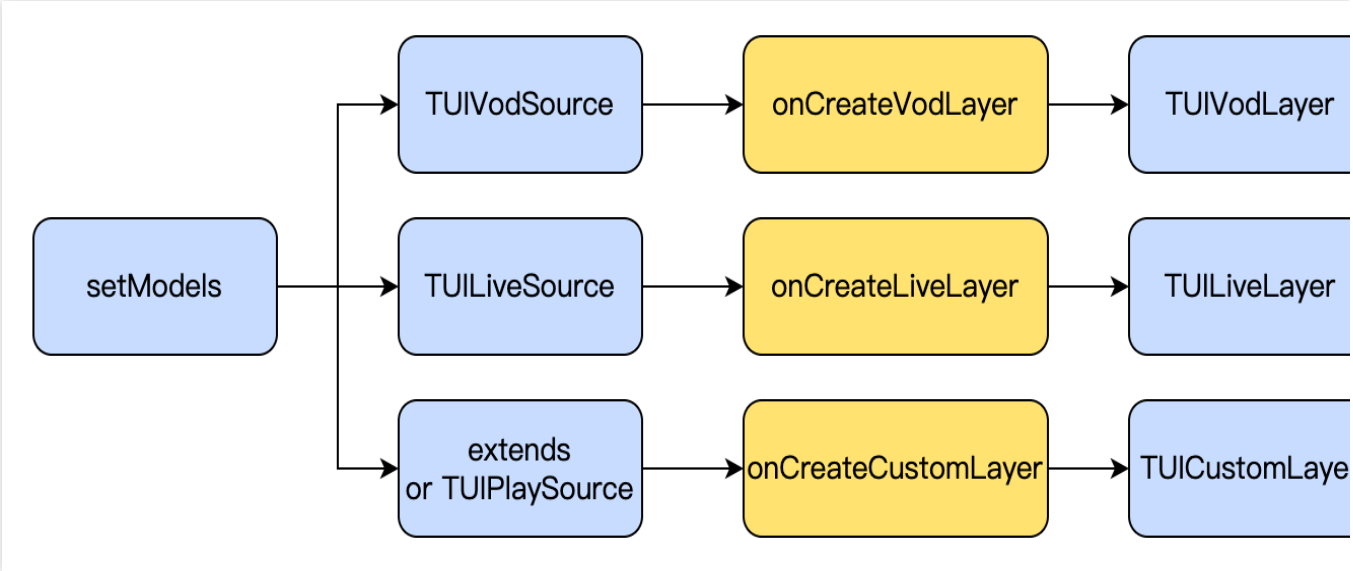
1. Layer Categories

If it is necessary to customize the UI for TUI short videos, it is required to use the layer capabilities of the short videos. Android TUI short videos adopt a layer management approach, providing each short video page with the ability to customize UI. Through the layer manager, each page can be object-managed, better handling the interaction between

video UI and player and video components.

Currently, there are three types of pages: on-demand, live broadcast, and custom pages. The corresponding layers for these three pages are `TUIVodLayer`, `TUILiveLayer`, and `TUICustomLayer`, respectively. Different layers can be inherited according to requirements. Different layer base classes provide interfaces and callbacks that fit their scenarios.

The corresponding relationship is shown in the figure below:

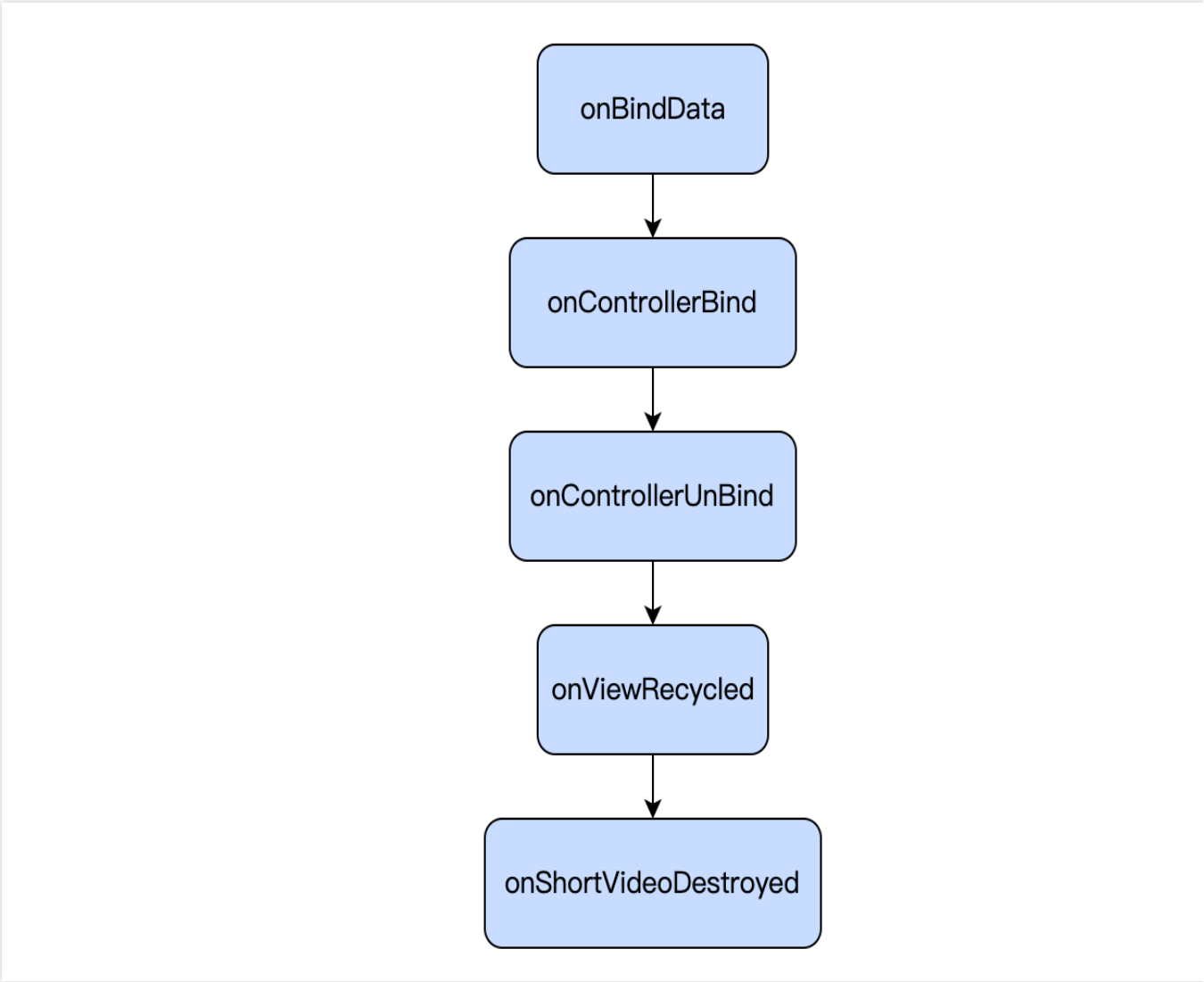


The display and hiding of layers will directly add and remove Views, without causing excessive interface rendering. Layers determine the display order of the interface based on the order they are added, with those added first appearing on the top layer and prioritized for display, while those added later are on the bottom layer and will be covered by those added before.

Due to the page reuse mechanism in the short video list, when there is business data-related UI display in the layer, it is necessary to reset the interface UI or set new values when binding data in `onBindData`.

2. Layer Lifecycle

The lifecycle of the three types of Layers is shown in the figure below:



The meanings of the lifecycle are as follows:

LifeCycle Name	Description
onBindData	Indicates that the current Layer has been bound to data. In this lifecycle, some static UI data initialization work can be done.
onControllerBind	This page is already the one currently displayed in the short video list. **Except for the custom page `TUICustomLayer`, calling `getPlayer()` and `getController()` at this time will no longer return null**, allowing operations on the player and page container.
onControllerUnBind	The page has been swiped away, and after this lifecycle, it will no longer be possible to obtain the player and videoView. This lifecycle can be used to

	perform resource recycling and interface resetting tasks.
onViewRecycled	The page has been recycled and will be used with other data and players. It is recommended to reset all data on the interface and recycle related resources in this lifecycle.
onShortVideoDestroyed	The TUI component has been destroyed, meaning the TUIShortVideoView's release method has been called.

II. Creating a Custom Layer

1. Create custom layer layouts

To create a custom layer, taking the on-demand layer as an example, you need to inherit from TUIVodLayer and then implement the layer you need.
Taking the video details layer as an example, you first need to implement the createView and tag methods. createView is the method for creating the layer view, and tag is a string label used by the business side to distinguish layers. createView will be called when the layer is added to the LayerManager.

```
@Override
public View createView(ViewGroup parent) {
    LayoutInflater inflater = LayoutInflater.from(parent.getContext());
    View view = inflater.inflate(R.layout.player_video_info_layer, parent, false);
    mSeekBar = view.findViewById(R.id.vsb_tui_video_progress);
    mTvProgress = view.findViewById(R.id.tv_tui_progress_time);
    mIvPause = view.findViewById(R.id.iv_tui_pause);
    mSeekBar.setListener(this);
    return view;
}

@Override
public String tag() {
    return "TUIVideoInfoLayer";
}
```

In createView, a View is created and returned. Here, you can use LayoutInflater to load a layout from XML, or you can create the layout directly using code.

2. Display Layout

After the View is created, it needs to be displayed at the appropriate time. TUIBaseLayer provides a rich set of event callbacks. For detailed information, see Layer Callbacks.

The video details layer can display the layout once it has obtained the data. Therefore, display the layer in onBindData.

```
@Override
public void onBindData(TUIVideoSource videoSource) {
    show();
}
```

3. Operating your own components

You can also operate on components you have created in other events. Taking the on-demand layer as an example, you need to declare your component as a member variable first, assign a value in onCreateView, and then handle player events such as the display and hiding of the pause button, as well as the callback for playback progress. See the following code.

```
@Override
public void onPlayBegin() {
    super.onPlayBegin();
    if (null != mIvPause) {
        mIvPause.setVisibility(View.GONE);
    }
}

@Override
public void onPlayPause() {
    super.onPlayPause();
    if (null != mIvPause) {
        mIvPause.setVisibility(View.VISIBLE);
    }
}

@Override
public void onPlayProgress(long current, long duration, long playable) {
    videoDuration = duration;
    if (null != mSeekBar) {
        // ensure a refresh at every percentage point
        int progressInt = (int) (((1.0F * current) / duration) * 100);
        if(lastProgressInt != progressInt) {
            setProgress(progressInt / 100F);
            lastProgressInt = progressInt;
        }
    }
}
```

4. Controlling the player

In addition to receiving events from the player, you can also control the player. For example, call the player to perform a seek operation.

```
@Override
public void onDragDone(VideoSeekBar seekBar) {
    TUIPlayerController controller = getPlayerController();
    if (null != controller && videoDuration > 0) {
        controller.seekTo((int) ((videoDuration * seekBar.getBarProgress()) / 1000)
    }
    if (null != mTvProgress) {
        mTvProgress.setVisibility(View.GONE);
    }
}
```

Currently, in TUIBaseLayer, you can obtain the VideoView object of the current page through getVideoView, obtain the playback controller of the current video through getPlayerController (only available when the current page is the video currently playing in the short video list), and get the current player object through getPlayer. Since the player and video view may be released or reused during scrolling, these three objects might be null when obtained, so it is necessary to check for null values.

5. Release when the layer is recycled

When the layer is recycled, some release operations need to be performed. This prevents external objects from holding onto the layer, causing memory leaks.

```
@Override
public void onViewRecycled(TUIBaseVideoView videoView) {
    // release your resource
}
```

6. Listening for whether the current layer is scrolled to

If you need to listen for whether the current page is the currently playing video, you can listen to the controller. When onControllerBind is triggered, the layer is bound by the controller, indicating that the page of the layer is about to be displayed and start playing. When onControllerUnBind is triggered, the controller is unbound, indicating that the page has been swiped away.

```
@Override
public void onControllerUnBind(TUIPlayerController controller) {
    super.onControllerUnBind(controller);
    show();
}
```

The above code is executed after the page is swiped away and the controller is unbound. To prevent the screen from going black when the page is swiped back later, it triggers the display of the cover image.

7. Vod playback obtains video information through the onRecFileVideoInfo callback.

When a fileId video source is set, the layer will callback the video information through onRecFileVideoInfo. See the following code:

```
@Override
public void onRecFileVideoInfo(TUIFileVideoInfo params) {
    if(isShowing()) {
        TUIBaseVideoView videoView = getVideoView();
        if (null != videoView && null != params) {
            String coverUrl = params.getCoverUrl();
            if (!TextUtils.isEmpty(coverUrl)) {
                ImageView imageView = getView();
                Glide.with(videoView).load(coverUrl)
                        .centerCrop()
                        .into(imageView);
                coverUrlFromServer = coverUrl;
            }
        }
    }
}
```

This method will be called back when playing only using fileID. It will return information such as the video URL link, cover image, duration, and sprite image.

It is recommended to pass the short video component for playback through the URL as much as possible, and pre-assign the cover image URL in advance, which can improve the loading performance of short videos.

8. Determine if the first frame of the video has arrived through the onRcvFirstIframe method

Usage example:

```
@Override
public void onRcvFirstIframe() {
    hidden();
}
```

For example, in scenarios such as cover images, you need to hide the cover image after receiving the first frame event.

III. Managing Layers

After integrating the short video component TUIShortVideoView, setting up a listener with TUIShortVideoView will callback the item creation method onCreateItemLayer at the appropriate time, allowing you to add or manage custom layers.

```
mSuperShortVideoView.setListener(new TUIShortVideoListener() {
```

```
    // ......

  @Override
   public void onCreateVodLayer(TUIVodLayerManager layerManger, int viewType) {
       layerManger.addLayer(new TUIVideoInfoLayer(mShortVideoView, ShortVideoFragmen
       layerManger.addLayer(new TUICoverLayer());
       layerManger.addLayer(new TUILoadingLayer());
       layerManger.addLayer(new TUIErrorLayer());
   }

   @Override
   public void onCreateLiveLayer(TUILiveLayerManager layerManager, int viewType) {
       layerManager.addLayer(new TUILiveEntranceLayer(mShortVideoView, ShortVideoFra
       layerManager.addLayer(new TUILiveLoadingLayer());
       layerManager.addLayer(new TUILiveErrorLayer());
   }

   @Override
   public void onCreateCustomLayer(TUICustomLayerManager layerManager, int viewType)
       if (viewType == SVDemoConstants.CustomSourceType.SINGLE_IMG_TYPE) {
           layerManager.addLayer(new PicDisplayLayer());
       }
   }
});
```

onCreateItemLayer has two parameters: layerManager, which is the layer manager that can add, remove, and query layers. The method of adding is shown in the figure above. `viewType` is the video type of the current page. If you have customized `extViewType` in `TUIPlayerSource`, the `viewType` here will be what you defined. If not defined, it will return `ITEM_TYPE_VOD`, `ITEM_TYPE_LIVE`, or `ITEM_TYPE_CUSTOM` according to the page type.

If no layer is needed, you can remove the layer**. After removal, the `unbindLayerManager` method will be called back in the layer.**

```
layerManger.removeLayer(layer);
```

If you need to obtain the layer hierarchy for interactive operations on layers, you can get the layer hierarchy through the following methods:

```
layerManger.indexOfLayer(layer);
```

IV. Creating an Image Display Page Using Custom Layers

1. Implement your own custom data

Taking displaying images as an example, create data with image links.

```
public class DemoImgSource extends TUIPlaySource {

    private String mImgUrl;

    public DemoImgSource(String imgUrl) {
        mImgUrl = imgUrl;
        // You can specify different viewType parameters to distinguish between typ
        setExtViewType(SVDemoConstants.CustomSourceType.SINGLE_IMG_TYPE);
    }

    public String getImgUrl() {
        return mImgUrl;
    }

    public void setImgUrl(String imgUrl) {
        this.mImgUrl = imgUrl;
    }
}
```

2. Implement the UI of a custom page

After implementing the data, it is necessary to customize the UI of your own custom page. Taking displaying images as an example, you need to inherit from TUICustomLayer to implement the layer.

```
public class PicDisplayLayer extends TUICustomLayer {

    private ImageView mDisplayImgView;

    @Override
    public View createView(ViewGroup parent) {
        // Constructing a Page view
        LayoutInflater inflater = LayoutInflater.from(parent.getContext());
        View view = inflater.inflate(R.layout.tuiplayer_img_display_layer, parent,
        mDisplayImgView = view.findViewById(R.id.iv_img_display);
        return view;
    }

    @Override
    public void onBindData(TUIPlaySource videoSource) {
        super.onBindData(videoSource);
        // Data is bound to the page, allowing access to the corresponding data sou
        if (videoSource.getExtViewType() == SVDemoConstants.CustomSourceType.SINGLE
            DemoImgSource source = (DemoImgSource) videoSource;
            Glide.with(mDisplayImgView).load(source.getImgUrl())
                    .into(mDisplayImgView);
        }
```

```
    }

    @Override
    public String tag() {
        return "PicDisplayLayer";
    }
}
```

Add your own layer into the TUI short video callback

```
mSuperShortVideoView.setListener(new TUIShortVideoListener() {

   // ......

  @Override
  public void onCreateCustomLayer(TUICustomLayerManager layerManager, int viewType)
     // custom layer
      if (viewType == SVDemoConstants.CustomSourceType.SINGLE_IMG_TYPE) {
          layerManager.addLayer(new PicDisplayLayer());
      }
  }
});
```

3. Populate data into TUI short video

```
// Customize data, DemoImgSource inherits from TUIPlaySource, customize data,
// different data can be customized here according to business needs
DemoImgSource imgSource = new DemoImgSource("imgUrl");
shortVideoData.add(imgSource);

// fill data
mSuperShortVideoView.setModels(shortVideoData);
```

Subsequently, the custom page will be displayed at the corresponding page position in the short video list, according to the position you added in the list.

TUI short video interface

1. Configure License

To use the TUI component, you need to configure the corresponding premium license. An example is as follows:

```
TUIPlayerConfig config = new TUIPlayerConfig.Builder()
        .enableLog(true)
        .licenseKey(LICENCE_KEY)
```

```
        .licenseUrl(LICENCE_URL)
        .build();
TUIPlayerCore.init(getApplicationContext(), config);
```

2. Set up lifecycle listening

Used for lifecycle control of TUIShortVideoView. Internally, it automatically pauses, plays, and destroys the list videos according to the lifeCycle status. This interface can be unset, and the business can take over the call.

```
mSuperShortVideoView.setActivityLifecycle(getLifecycle());
```

3. Set up short video listening

Used to listen to events from TUIShortVideoView, including the timing of loading paginated data, and the callback during page creation, where layers can be added.

```
mSuperShortVideoView.setListener(new TUIShortVideoListener() {
    @Override
    public void onCreateVodLayer(TUIVodLayerManager layerManger, int viewType) {
        layerManger.addLayer(new TUIVideoInfoLayer(mShortVideoView, ShortVideoFragm
        layerManger.addLayer(new TUICoverLayer());
        layerManger.addLayer(new TUILoadingLayer());
        layerManger.addLayer(new TUIErrorLayer());
    }

    @Override
    public void onCreateLiveLayer(TUILiveLayerManager layerManager, int viewType) {
        layerManager.addLayer(new TUILiveEntranceLayer(mShortVideoView, ShortVideoF
        layerManager.addLayer(new TUILiveLoadingLayer());
        layerManager.addLayer(new TUILiveErrorLayer());
    }

    @Override
    public void onCreateCustomLayer(TUICustomLayerManager layerManager, int viewTyp
        if (viewType == SVDemoConstants.CustomSourceType.SINGLE_IMG_TYPE) {
            layerManager.addLayer(new PicDisplayLayer());
        }
    }

    @Override
    public void onPageChanged(int index, TUIPlaySource videoSource) {
        if (index >= mShortVideoView.getCurrentDataCount() - 1) {
            mShortViewRefresh.setRefreshing(true);
            ShortVideoModel.getInstance().loadMore(false);
        }
    }
```

```
});
```

4. Set the video playback strategy

Set various strategies during the video playback process.

The TUIPlayerVodStrategy parameter.

It needs to be constructed using the Builder.

Method	Description
setPreloadCount	Set the maximum concurrent preload quantity, default is 3.
setPreDownloadSize	Set the preload cache size, default is 1MB, unit is MB.
setPreLoadBufferSize	Set the pre-play cache size, default is 0.5MB, unit is MB.
setMaxBufferSize	Set the video cache size during playback, default is 10MB, unit is MB.
setPreferredResolution	Set the preferred resolution for video playback, default is 720 x 1280.
setProgressInterval	Playback progress callback interval, default is 500 milliseconds, unit is milliseconds.
setRenderMode	Render tiling mode, default is 0. In liteavPlayer, 0 represents full screen, 1 represents rendering according to the actual ratio of the video, which may have black borders.
setExtInfo	Set additional information.
setMediaType	When the media type to be played is known in advance, the media type can be set through this interface to reduce the detection of playback types within the player SDK and improve the startup speed.
enableAutoBitrate	Set whether to enable bitrate adaptation.
setResumeMode	Set the resume mode, which is divided into three modes: TUIConstants.TUIResumeMode.NONE: Do not resume playback. TUIConstants.TUIResumeMode.RESUME_LAST: Resume playback from the last time played. TUIConstants.TUIResumeMode.RESUME_PLAYED: Resume playback of all watched videos.
setDisplayViewFactory	Set a custom video layer, which can be customized by implementing IDisplayViewFactory
setEnableAccurateSeek	Set whether to enable precise seeking. After enabling precise seeking, the accuracy of seeking will be greatly improved, but seeking will take time. After disabling it, the

	actual seeking time may differ from the expected time, depending on the distribution of key frames in the video, but the seeking time will become shorter.
setAudioNormalization	Set volume equalization, with a loudness range of -70 to 0 (LUFS), and also supports custom values. Note: Supported starting from version 11.7 of the advanced player. Predefined values can be filled in, related constant class TXVodConstants, Off: AUDIO_NORMALIZATION_OFF On: AUDIO_NORMALIZATION_STANDARD (standard) AUDIO_NORMALIZATION_LOW (low) AUDIO_NORMALIZATION_HIGH (high)
setIsRetainPreVod	Whether to keep the previous player to speed up the startup of the previous player
setRetryCount	Set the number of on-demand retry attempts under weak network conditions. If set to 0, no retries will be performed.

5. Fill in the data

Fill data into TUIShortVideoView:

```
mSuperShortVideoView.setModels(shortVideoBeanList);
```

Append data:

```
mSuperShortVideoView.appendModels(shortVideoBeanList);
```

TUIVideoSource class

Method	Param Type	
setVideoURL	String	Video link, it is recommended to fill in this field, which will speed up the preloading process.
setCoverPictureUrl	String	Video cover, which will be callback to the layer, and handled by the customer.
setAppId	int	Video appId.
setFileId	String	Video fileId.
setPSign	String	Video encryption pSign.
setExtViewType	int	Custom page type, this value will be callback through the second parameter of the Layer creation callback, for the business to distinguish different types of pages.

setExtInfoAndNotify	Object	For business to extend additional parameters on their own, using this method can notify messages to the existing layer in real-time. This method will only be valid after obtaining Source through TUIDataManager.
setVideoConfig	TUIPlayerVideoConfig	Video independent configuration.
setExternalSubtitle	List<TUISubtitleSource>	Set external subtitles, which will be loaded into the on-demand player automatically, and must be supported by the advanced version of the player.
setAutoPlay	boolean	Set whether this video should play automatically.

TUIPlayerVideoConfig class

Method	Param Type	Description
setPreloadBufferSizeInMB	float	Set a separate pre-play cache size for the video, optional.
setPreferredResolution	long	Set a separate playback and preload resolution for the video, optional.
setPreDownloadSize	float	Set a separate pre-download cache size for the video, optional.

TUILiveSource class

Method	Param Type	Description
setUrl	String	Live streaming link
setCoverPictureUrl	String	Video Cover, which will be callback to the layer, and handled by the customer.
setExtViewType	int	Custom page type, this value will be callback through the second parameter of the Layer creation callback, for the business to distinguish different types of pages.
setExtInfoAndNotify	Object	For business to extend additional parameters on their own, using this method can notify messages to the existing layer in real-time. This method will only be valid after obtaining Source through TUIDataManager.
setLiveConfig	TUIPlayerLiveConfig	Independent configuration.

setAutoPlay	boolean	Set whether to play automatically. When this field is set to false for live streaming, there will be no picture at first, and a cover image is needed to cover it.

TUIPlayerLiveConfig class

Method	Param Type	Description
setCacheMinTime	float	Minimum time for automatic cache adjustment, the value must be greater than 0. [Default Value]: 1.
setCacheMaxTime	float	Maximum time for automatic cache adjustment, the value must be greater than 0. [Default Value]: 5

TUIPlaySource class

Method	Param Type	Description
setExtViewType	int	Custom page type, this value will be callback out through the second parameter of the Layer creation callback, for the business to distinguish different types of pages.
setExtInfoAndNotify	Object	Used for the business to extend additional parameters on their own, this method can notify messages to an existing layer in the interface in real-time. This method will only be valid after being called by a Source obtained through TUIDataManager. 6. Operation list data

TUI Short Video provides data operation interfaces that allow real-time modification of data already added to the list. You can obtain the data operation object by calling the getDataManager() method of TUIShortVideoView, as shown below:

```
// Obtain the data operation object
TUIShortVideoDataManager dataManager = mSuperShortVideoView.getDataManager();
```

After obtaining the data operation object, you can manipulate the list data in real-time. The interfaces are as follows:

Method	Returned Param	Passed Param	Description
removeData	void	index: The position of the page to be removed	Remove the corresponding page and

			data
removeRangeData	void	index: The starting position of the page to be removed count: The number to be removed from the starting position, excluding the last digit of count	Remove a range of pages and data
removeDataByIndex	void	removeIndexList: A collection of positions of the pages to be removed	Remove all pages and data within the index collection according to the passed index
addData	void	source: The data to be inserted index: The position where the data is to be inserted	Insert the passed data into the specified position based on the position where the data is to be inserted
addRangeData	void	sources: A collection of data to be inserted startIndex: The starting position for inserting data	Insert the data collection into the specified position based on the passed starting position
replaceData	void	source: The data to be replaced index: The position to be replaced	Replace the data at the specified position with the passed data based on the passed position
replaceRangeData	void	sources: A collection of data to be replaced startIndex: The starting position for replacement	Replace the specified position with the passed data collection based on the passed starting position
getDataByPageIndex	TUIVideoSource	index: Page position	Obtain the page data at the specified position based on the passed position
getCurrentDataCount	int	-	Get the total number of all data in the current list
getCurrentIndex	int	-	Get the position of the currently displayed page

getCurrentModel	TUIVideoSource	-	Get the data of the currently displayed page

7. Obtain the currently playing video resource

Obtain the currently playing video resource, see the following code for usage:

```
mSuperShortVideoView.getCurrentModel()
```

8. pause

Pause the currently playing video.

```
mSuperShortVideoView.pause();

// All videos will pause after starting playback until resume is called to interrup
mSuperShortVideoView.pause(true)
```

9. Play from a specified position.

Start playing from a specified position. This method can directly jump to the specified position during playback, and does not smooth jump by default. See the following code for usage:

```
// index is the specified page position.
mSuperShortVideoView.startPlayIndex(index);


// index is the position to navigate to, true indicates that smooth switching is re
// and the default is false.
mSuperShortVideoView.startPlayIndex(index, true);
```

10. Set the short video playback mode.

Currently, there are two short video playback modes: list loop playback `TUIVideoConst.ListPlayMode.MODE_LIST_LOOP`, which automatically plays the next video after the current one finishes, and after playing the last one, it automatically returns to the first video to continue playback. And single video loop playback `TUIVideoConst.ListPlayMode.MODE_ONE_LOOP`, which will keep repeating the current video until the user manually swipes to flip the page. It can also be set to TUIVideoConst.ListPlayMode.MODE_CUSTOM, allowing the business to take over the playback logic. When not set, the default is `MODE_ONE_LOOP`. The setting method is as follows:

```
// set to MODE_ONE_LOOP
mSuperShortVideoView.setPlayMode(TUIVideoConst.ListPlayMode.MODE_ONE_LOOP);
```

11. Destroy the control.

Destroy the control and resources.

```
mSuperShortVideoView.release()
```

12. Resume playing the current video.

Resume playing the current video. See the following code for usage:

```
mSuperShortVideoView.resume()
```

13. Real-time resolution switching

The TUI short video can switch the current video resolution and the global video resolution in real-time. The interface is as follows:

```
mSuperShortVideoView.switchResolution(720 * 1080, TUIConstants.TUIResolutionType.CU
```

In addition to passing TUIResolutionType, switchType can also directly specify the index of the video that needs to be switched to change the video resolution. The meaning of switchType is as follows:

Param	Description
GLOBAL	Set global resolution
CURRENT	Set current video resolution
Other values greater than or equal to 0	Set resolution at specified position

Current priority of resolution, the setting priority of the current video resolution is higher than the global resolution.

14. Pause and resume preloading

TUI short video can pause and resume preloading tasks in real-time.

```
// pause all preload
mSuperShortVideoView.pausePreload();
// start preload from current video index
mSuperShortVideoView.resumePreload();
```

When calling to resume preloading, it will continue preloading from the current video onwards.

15. Add a layer

TUI short video enables custom UI on the playback interface by adding layers. When the onCreateItemLayer callback occurs, you can utilize the LayerManager provided by the method to add and manage layers. You can inherit from TUIBaseLayer to customize the layers you require. These layers will appear above the VideoView.
The visibility of layers is managed by adding and removing Views, which avoids issues related to excessive rendering.

onCreateVodLayer, onCreateLiveLayer, and onCreateCustomLayer parameters

Param	Type	Description
layerManger	TUIVodLayerManager, TUILiveLayerManager And TUICustomLayerManager	Layer management object
viewType	int	Current video playback type TUIVideoConst.ItemType.ITEM_TYPE_VOD: On-demand TUIVideoConst.ItemType.ITEM_TYPE_LIVE: Live broadcast TUIVideoConst.ItemType.ITEM_TYPE_CUSTOM: Custom interface Other custom types passed by PlayerSource

```
layerManger.addLayer(new TUICoverLayer());
layerManger.addLayer(new TUIVideoInfoLayer());
layerManger.addLayer(new TUILoadingLayer());
layerManger.addLayer(new TUIErrorLayer());
```

16. Remove the specified layer

Remove the layer from the layer manager.

```
layerManger.removeLayer(layer);
```

17. Remove all layers

Remove all layers from the layer manager.

```
layerManger.removeAllLayer();
```

18. Get the current layer level

Pass in the layer to obtain its current level, which can be used to determine the order of layer display and the sequence of touch event propagation.

```
layerManger.indexOfLayer(layer);
```

19. Custom list scrolling speed

TUI short video provides an interface for customizing the scrolling speed of the list, which can be achieved through the setPageScroller method of TUIShortVideoView. Here's an example:

```
mSuperShortVideoView.setPageScroller(new LinearSmoothScroller(getContext()){
    @Override
    protected float calculateSpeedPerPixel(DisplayMetrics displayMetrics) {
        // Return the time taken for each pixel to slide, measured in milliseconds
        return 25f/displayMetrics.densityDpi;
    }
});
```

20. Disable list sliding

You can disable or enable list sliding by using the `setUserInputEnabled` method of `TUIShortVideoView`. Here's an example:

```
// false:disables user sliding
mSuperShortVideoView.setUserInputEnabled(false);
```

21. Layer callback

Layer callbacks include the base class `ITUILayer` callbacks, player event notifications, and component events from the videoView. After inheriting from `TUIVodLayer`, `TUILiveLayer`, and `TUICustomLayer`, you can receive callbacks for video playback according to your functional requirements. Currently, the callback functions are as follows:

TUIVodLayer class

Method	Return Type	Param	Description
isShowing	boolean	-	Whether the current layer is being displayed.
createView	View	parent: Layer container	Abstract method, needs to be implemented by yourself, used to create the View of the layer.
tag	String	-	The tag of the layer, used to distinguish different layers.

unBindLayerManager	void	-	The layer is unbound from the manager, which generally happens when the layer is removed.
show	void	-	Display the current layer.
hidden	void	-	Hide the current layer.
getView	T extends View	-	Get the View of the current layer.
getVideoView	TUIBaseVideoView	-	Get the current VideoView, if the layerManager has not yet bound with the VideoView, it will return empty.
getPlayerController	TUIPlayerController	-	Get the current playback Controller, if it has not yet bound with the Controller, it will return empty.
getPlayer	ITUIVodPlayer	-	Get the current player, if it has not yet bound with the Controller, it will return empty.
getRenderMode	int	-	Get the rendering fill mode of the current player screen.
onControllerBind	void	controller: Current video playback controller	The current VideoView is bound to the playback controller, that is, the current video becomes the video playing in the list.
onControllerUnBind	void	controller: Current video playback controller	The VideoView is unbound from the playback controller, which generally indicates that the page has slid out of the interface.
onBindData	void	videoSource: Video data	The current VideoView has bound video data.
onViewRecycled	void	videoView: Current player view container	The current videoView has been recycled.
onExtInfoChanged	void	videoSource: Changed video source	When extInfo is set through TUIPlaySource, the layer will notify events through this callback.
onShortVideoDestroyed	void	-	When the entire short video component is

			destroyed, it will notify the layer through this callback for the layer to release resources.

TUILiveLayer class

Method	Return Type	Param	Description
isShowing	boolean	-	Whether the current layer is currently being displayed.
createView	View	parent: Layer container	Abstract method that needs to be implemented by oneself, used for creating the View of the layer.
tag	String	-	The tag of the layer, used to differentiate between different layers.
unBindLayerManager	void	-	The layer is unbound from the manager, which typically occurs when the layer is removed.
show	void	-	Show the current layer.
hidden	void	-	Hide the current layer.
getView	T extends View	-	Obtain the View of the current layer.
getVideoView	TUIBaseVideoView	-	Get the current VideoView; if the layerManager has not yet bound to the VideoView, it will return empty.
getPlayerController	TUIPlayerController	-	Obtain the current playback Controller; if it has not yet bound to the Controller, it will return empty.
getPlayer	ITUILivePlayer	-	Get the current player; if it has not yet bound to the Controller, it will return empty.
getRenderMode	int	-	Get the rendering fill mode of the current player screen.
onControllerBind	void	controller: Current video playback controller	The current VideoView is bound to the playback controller, meaning the current video becomes the one playing in the list.

onControllerUnBind	void	controller: Current video playback controller	The VideoView is unbound from the playback controller, typically indicating that the page has slid off the interface.
onBindData	void	videoSource: Video data	The current VideoView has bound video data.
onViewRecycled	void	videoView: Current player view container	The current videoView has been recycled.
onExtInfoChanged	void	videoSource: Changed video source	After setting extInfo through TUIPlaySource, the layer will notify events through this callback.
onShortVideoDestroyed	void	-	When the entire short video component is destroyed, it will notify the layer through this callback so that the layer can release resources.

TUICustomLayer class

Method	Return Type	Param	Description
isShowing	boolean	-	Whether the current layer is being displayed.
createView	View	parent: Layer container	Abstract method that needs to be implemented by oneself, used for creating the View of the layer.
tag	String	-	The tag of the layer, used to distinguish different layers.
unBindLayerManager	void	-	The layer is unbound from the manager, which generally occurs when the layer is removed.
show	void	-	Display the current layer.
hidden	void	-	Hide the current layer.
getView	T extends View	-	Obtain the View of the current layer.
onControllerBind	void	-	The current VideoView is bound to the playback controller, meaning the current video

			is displayed in the list.
onControllerUnBind	void	-	The VideoView is unbound from the playback controller, typically indicating that the page has slid off the interface.
onBindData	void	videoSource: Video data	The current VideoView has bound video data.
onViewRecycled	void	videoView: Current player view container	The current videoView has been recycled.
onExtInfoChanged	void	videoSource: Changed video source	After setting extInfo through TUIPlaySource, the layer will notify events through this callback.
onShortVideoDestroyed	void	-	When the entire short video component is destroyed, it will notify the layer through this callback so that the layer can release resources.

TUIVodObserver class

Method	Return Type	Param	Description
onPlayPrepare	void	-	Video is ready.
onPlayBegin	void	-	Video starts playing.
onPlayPause	void	-	Video pauses.
onPlayStop	void	-	Video stops.
onPlayLoading	void	-	Video begins loading.
onPlayLoadingEnd	void	-	Video loading completes.
onPlayProgress	void	current: The current video playback progress, in milliseconds, of type long. duration: The total duration of the current video, in milliseconds, of type long.	Video playback progress.

		playable: The playable duration of the current video, in milliseconds, of type long.	
onSeek	void	position: The progress of the video to jump to. In seconds, of type int.	Video progress seeked.
onError	void	code: Video error code. message: Error description.	An error occurred during video playback.
onRcvFirstIframe	void	-	Received the first frame event.
onRcvAudioTrackInformation	void	infoList：Audio Track information.	Receive audio track information.
onRcvSubTitleTrackInformation	void	infoList: Video subtitle information.	Received video subtitle information.
onRecFileVideoInfo	void	params: Video file information.	Received video file information, which generally only triggers this callback when playing using fileId.
onResolutionChanged	void	width: Video width. height: Video height.	The current video resolution changes.
onPlayEvent	void	player: Player. event: Event ID. bundle: Event content.	All event notifications from the player.
onPlayEnd	void	-	The current video playback has ended.

TUILiveObserver class

Medthod	Param	Description
onError	player: Current live broadcast player code: Error code msg: Error message extraInfo: Additional message	An error occurred during playback.

onWarning	player: Current live broadcast player code: Error code msg: Warning message extraInfo: Additional message	A warning occurred in the player.
onVideoResolutionChanged	player: Current live broadcast player width: Video width height: Video height	The player's resolution has changed.
onConnected	player: Current live broadcast player extraInfo: Additional information	Data stream connection successful.
onVideoPlaying	player: Current live broadcast player firstPlay: Whether it is the first playback, **which can be used to determine the first frame** extraInfo: Additional information	Video playback has started.
onAudioPlaying	player: Current live broadcast player firstPlay: Whether it is the first playback extraInfo: Additional information	Audio playback has started.
onVideoLoading	player: Current live broadcast player extraInfo: Additional information	Video loading has begun.
onAudioLoading	player: Current live broadcast player extraInfo: Additional information	Audio loading has begun.
onPlayoutVolumeUpdate	player: Current live broadcast player volume: Changed volume	Player volume size callback.

onStatisticsUpdate	player: Current live broadcast player statistics: Data details	Live broadcast player statistics callback.
onSnapshotComplete	player: Current live broadcast player image: Screenshot image	Screenshot callback.
onRenderVideoFrame	player: Current live broadcast player videoFrame: Image frame	Custom video rendering callback, enabled by calling enableObserveVideoFrame.
onPlayoutAudioFrame	player: Current live broadcast player audioFrame: Audio frame	Custom audio data callback, enabled by calling enableObserveAudioFrame.
onReceiveSeiMessage	player: Current live broadcast player payloadType: SEI payloadType of the callback data. data: Data	Callback for receiving SEI messages, sent by the sender through sendSeiMessage in V2TXLivePusher.
onStreamSwitched	player: Current live broadcast player url: Switched URL code: Status code, 0: Success, -1: Switching timed out, -2: Switching failed, server error, -3: Switching failed, client error	Resolution seamless switching callback.
onLocalRecordBegin	player: Current live broadcast player code: Status code. 0: Recording task started successfully. -1: Internal error caused recording task startup failure. -2: Incorrect file extension (such as unsupported recording format). -6: Recording has already started, need to stop recording first. -7: Recording file already	Event callback for the start of recording tasks.

	exists, need to delete the file first. -8: Recording directory has no write permission, please check the directory permission issue. storagePath: Recorded file address.	
onLocalRecording	player: Current live broadcast player durationMs: Recording duration storagePath: Recorded file address.	Progress event callback for recording tasks in progress.
onLocalRecordComplete	player: Current live broadcast player code: Status code. 0: End recording task successfully. -1: Recording failed. -2: Switching resolution or landscape/portrait screen caused recording to end. -3: Recording time is too short, or no video or audio data has been collected, please check the recording duration, or whether audio and video collection has been enabled. storagePath: Recorded file address.	Event callback for recording tasks that have ended.

22. Player functions

In the layer, you can obtain the player object through getPlayer. The interface functions of the player ITUIVodPlayer and ITUILivePlayer are as follows:

ITUIVodPlayer class

Medthod	Return Type	Param	Desc
prePlay	void	model: Video data, type TUIVideoSource	Prelo dedu

			inside
resumePlay	void	-	Conti video
seekTo	void	time: The time point to jump to, in seconds, of type int.	Jump playb
isPlaying	boolean	-	Chec playir
startPlay	void	model: Video data, type TUIVideoSource.	Play
pause	void	-	Paus
stop	void	needClearLastImg: Optional parameter, whether to clear the current image.	Stop
getCurrentPlayTime	float	-	Get tl in sec
setDisplayView	void	videoView: Video rendering View, type TUIVideoRenderView.	Set th by the
setSurface	void	surface：render surface	Set th by the
addPlayerObserver	void	observer: Player observer, type TUIVodObserver.	Set th
removePlayerObserver	void	observer: Player observer, type TUIVodObserver.	Remo
setConfig	void	config: Video configuration, type TXVodPlayConfig.	Set th
setRenderMode	void	renderMode: Rendering mode, FULL_FILL_SCREEN: Maintain aspect ratio and fill the screen ADJUST_RESOLUTION: Maintain aspect ratio and display the video	Set th
setStartTime	void	startTime: Start playing time, in seconds. float type	Set th valid takes playb begir

setLoop	void	isLoop: Whether to loop	Set w loope
setBitrateIndex	void	index: Bitrate index	Set th
switchResolution	void	resolution: Resolution, i.e., width × height	Set th this p corre will b effect
getSupportResolution	List<TUIPlayerBitrateItem>	-	Get th the vi
getBitrateIndex	int	-	Get th video
setRate	void	rate: Video speed, normal speed is 1.0	Set th curre
getCurrentPlaybackTime	float	-	Get th the cu
getBufferDuration	float	-	Get th curre
getDuration	float	-	Get th curre
getPlayableDuration	float	-	Get th curre
getWidth	int	-	Get th being
getHeight	int	-	Get th being
setRenderRotation	void	rotation: Angle	Set th curre
enableHardwareDecode	boolean	enable: Whether to enable hardware decoding	Set w uses
setMute	void	mute: Whether to mute	Set w playir
setAudioPlayoutVolume	void	volume: Video volume, 0 to 100	Set th

			curre
setRequestAudioFocus	boolean	requestFocus: Whether to get audio focus	Set w gets a
snapshot	void	listener: TXLivePlayer.ITXSnapshotListener type, screenshot callback	Take video curre
setMirror	void	mirror: Whether to mirror	Set w is mir
setToken	void	token: Encrypted HLS token	Set th
isLoop	boolean	-	Chec in loo
attachTRTC	void	trtcCloud: trtc service object	Push strea TRTC can b overv
detachTRTC	void	-	Unbir playe
setStringOption	void	key: Business parameter key value value: Business parameter value	Set th of the
setSubtitleView	void	subtitleView: Subtitle component	Set th the vi
addSubtitleSource	void	url: Subtitle link name: Subtitle name mimeType: Subtitle format	Add s video
selectTrack	void	trackIndex: Audio/video track	Add/s comn track
deselectTrack	void	trackIndex: Audio/video track	Remo track
setAudioNormalization	void	value: Loudness range: -70 to 0 (LUFS), with custom values supported. Note: The player advanced version 11.7 starts to support this feature. Fill in the preset value, related constants	Set v

		TXVodConstants, Off: AUDIO_NORMALIZATION_OFF On: AUDIO_NORMALIZATION_STANDARD (Standard) AUDIO_NORMALIZATION_LOW (Low) AUDIO_NORMALIZATION_HIGH (High)	
setSubtitleStyle	void	renderModel: Subtitle format	Set s
getSubtitleTrackInfo	List<TXTrackInfo>	-	Get ir inforr prepa
getAudioTrackInfo	List<TXTrackInfo>	-	Get ir inforr prepa
setVodSubtitleDataListener	void	listener:Custom subtitle callback.	Set a The p subtit featu parar TUIV TUIP = new TUIP Map< new H map. config sourc

TUIPlayerBitrateItem class

Method	**Return Type**	Description
getIndex	int	Current resolution bitrate index.
getWidth	int	Current resolution video width.
getHeight	int	Current resolution video height.
getBitrate	int	Current resolution video bitrate.

ITUILivePlayer class

Medthod	Return Type	Param	De
prePlay	void	model: Video data, type TUILiveSource	Pre me
resumePlay	void	-	Co
setConfig	void	config: Live streaming configuration	Se
addLiveObserver	void	observer: Live event observer	Su
removeLiveObserver	void	observer: Live event observer	Un
switchStream	int	model: The data source to be switched	Sw cu su
setRenderRotation	int	rotation: Rotation direction	Se
setPlayoutVolume	int	volume: Volume size, ranging from 0 to 100. 【Default】: 100	Se
getStreamList	ArrayList<V2TXLiveDef.V2TXLiveStreamInfo>	-	Ge
enableVolumeEvaluation	int	value: Determines the trigger interval of the onPlayoutVolumeUpdate callback, in milliseconds, with a minimum interval of 100ms. If it is less than or equal to 0, the callback will be turned off, and it is recommended to set it to 300ms;【Default】: 0, not enabled	En
snapshot	int	-	Ca Re V V

			is i scr
enableReceiveSeiMessage	int	enable: true: Enable receiving SEI messages; false: Disable receiving SEI messages. 【Default】: false. payloadType: Specify the payloadType to receive SEI messages, supporting 5 and 242, and keep it consistent with the payloadType of the sending end.	En
showDebugView	void	isShow: Whether to display.【Default】: false.	Wh pla
setProperty	int	key: The key corresponding to the advanced API. value: The parameter needed when calling the advanced API corresponding to the key.	Ca V2
startLocalRecording	int	params: Local recording audio and video configuration	Sta Re V2 V2 : In V2 ref
stopLocalRecording	void	-	Sto sto rec rec

23. Set list animation.

TUI short video list supports setting animation effects for elements when UI changes such as being added or removed. The example is as follows:

```
mShortVideoView.setItemAnimator(new DefaultItemAnimator());
```

Wherein, the ItemAnimator is consistent with the ItemAnimator of RecyclerView.

24. Globally listen for video pre-download changes.

TUI Core provides static interfaces that can globally listen for changes in the status of video pre-downloads. The example is as follows:

```
TUIPreloadApi.getInstance().addObserver(new TUIPreloadObserver() {
    @Override
    public void onVodPreloadStatusChanged(TUIVideoSource vodSource, int state, long

    }
});
```

Wherein, the state can refer to `TUIConstants#TUIPreLoadState` .

Advanced features

1. Business notification messages to page layers

TUI provides a message interface for users to notify data in real-time to the current layer. After obtaining the video object through the data operation object, notifications can be made. The example is as follows:

```
// get data controller
TUIShortVideoDataManager dataManager = mSuperShortVideoView.getDataManager();
// get dataSource
TUIPlaySource videoSource = dataManager.getDataByPageIndex(0);
Map<String, String> data = new HashMap<>();
data.put("key1", "data1");
// set extInfo and notify to layer
videoSource.setExtInfoAndNotify(data);
```

Subsequently, the notification will be received in the layer's `onExtInfoChanged` callback, allowing UI modifications to the current page. The example is as follows:

```
@Override
public void onExtInfoChanged(TUIVideoSource videoSource) {
    super.onExtInfoChanged(videoSource);
    Map<String, String> data = (Map<String, String>) videoSource.extInfo;
    Log.i("tag", "get data:" + data);
}
```

Note：

`onExtInfoChanged` will only be triggered after binding data with `onBindData`.

2. Set volume balance for on-demand playback

The player supports automatic volume adjustment while playing audio to ensure that the volume of all audio is consistent. This can avoid issues where some audios are too loud or too quiet, providing a better listening experience. By setting volume balance, the loudness range is -70 to 0 (LUFS), and custom values are also supported.

Note：

Supported starting from version 11.7 of the premium player edition.

```
/*
 * Supports custom numerical values, loudness range: -70 to 0 (LUFS)
 * Preset values: Off: TXVodConstants#AUDIO_NORMALIZATION_OFF
 *                On (Low): TXVodConstants#AUDIO_NORMALIZATION_LOW
 *                On (Standard): TXVodConstants#AUDIO_NORMALIZATION_STANDARD
 *                On (High): TXVodConstants#AUDIO_NORMALIZATION_HIGH
 */
TUIPlayerVodStrategy vodStrategy = new TUIPlayerVodStrategy.Builder()
        .setAudioNormalization(TXVodConstants.AUDIO_NORMALIZATION_STANDARD)
        .build();
mShortVideoView.setVodStrategy(vodStrategy);
```

3. Customize the ability to pull down to refresh and pull up to load more.

TUIShortVideoView exposes three methods: computeVerticalScrollOffset, computeVerticalScrollRange, and computeVerticalScrollExtent, which are used to calculate whether the current list has scrolled to the top or the bottom. Wherein:

`computeVerticalScrollOffset`: The current vertical scroll offset (in pixels).

`computeVerticalScrollRange`: The vertical scroll range of the RecyclerView (total height, in pixels).

`computeComputeVerticalScrollExtent`: The total height of visible items in the current scroll view (in pixels).

During the sliding process, if the `computeVerticalScrollOffset` method returns a value less than or equal to 0, it indicates that the list has scrolled to the top. At this point, the touch event can be passed to the outer layout to perform a pull-to-refresh.

During the sliding process, you can use the sum of the currently visible height `computeVerticalScrollExtent` and the distance already scrolled `computeVerticalScrollOffset`. If the sum is greater than or equal to the total height `computeVerticalScrollRange`, it can be judged whether the list has scrolled to the bottom. The code is as follows:

```
    int verticalScrollOffset = recyclerView.computeVerticalScrollOffset();
    int verticalScrollRange = recyclerView.computeVerticalScrollRange();
    int verticalScrollExtent = recyclerView.computeVerticalScrollExtent();

    boolean isAtBottom = verticalScrollOffset + verticalScrollExtent >= verticalScrol
```

VR Playback Plugin

Last updated：2024-12-02 10:05:59

Function Introduction

The VR playback component can be used for VR panoramic video playback. During playback, the viewing angle can be changed by gyroscope rotation or gesture operations, allowing for 360-degree unobstructed viewing of panoramic videos. Currently, the VR playback component supports configuring monocular or binocular modes. Monocular mode is suitable for naked-eye viewing of panoramic videos, while binocular mode is suitable for viewing with VR glasses and other devices. In addition, the iOS side supports 180-degree hemispherical panoramic videos, which can adapt to more usage scenarios.

Conditions for Use

Currently, the VR playback plugin can be used with mobile player SDK version 11.3 and above, as well as the mobile player SDK (Advanced Edition).
VR playback requires obtaining the advanced version of the mobile player License to use.

Operation Steps

Step 1: Obtain the Plugin

1. The VR playback plugin (hereinafter referred to as the "plugin") is provided as a separate SDK, which you can integrate into your project based on your project requirements. The download address is as follows:

Terminal Category	SDK Download Address
Android Side	VR plugin
iOS Side	VR plugin

2. Integrate the downloaded "plugin plugin_monet-release-v.x.x.x.aar" or "TXCMonetPlugin-release-v.x.x.x.framework" (where x.x.x. is the version number) into the project. After starting the player instance, the host will automatically load the plugin.

Step 2: Initialize License

To use the VR playback function, you need to use the advanced version of the mobile player License.. You can refer to the mobile player License guide to obtain it. If you have obtained the corresponding License, you can go to VOD Console > License Management > Mobile License to get the corresponding LicenseURL and LicenseKey, and then initialize the License through the following interface.

Android

iOS

```
String licenseUrl = "Enter the URL of the License you purchased";
String licenseKey = "Enter the Key of the License you purchased"
TXLiveBase.getInstance().setLicence(context, licenseUrl, licenseKey);
```

```
NSString *licenseURL = "Enter the URL of the License you purchased";
NSString *licenseKey = "Enter the Key of the License you purchased"
[TXLiveBase setLicenceURL:licenseURL key:licenseKey];
```

Step 3: Enable or disable the terminal's VR panoramic video capability

After starting playback (it is recommended to call after receiving the TXLiveConstants.PLAY_EVT_VOD_PLAY_PREPARED event), you can enable and disable the VR panoramic video through the following interface in TXVodPlayer:

Android

iOS

```
@Override
public void onPlayEvent(TXVodPlayer player, int event, Bundle param) {
    if (event == TXLiveConstants.PLAY_EVT_VOD_PLAY_PREPARED) {
        // Enable the terminal VR panoramic video monocular mode. If you need to en
        mVodPlayer.setStringOption("PARAM_MODULE_TYPE", 11);
    }


}


// Disable terminal VR panoramic video.
mVodPlayer.setStringOption("PARAM_MODULE_TYPE", 0);
```

```
// Enable VR panoramic video monocular mode. If you need to enable the binocular mo
// Note: The settings for enabling VR panoramic video need to be set before the pla
// It is recommended to add a switch animation here.
NSMutableDictionary *extInfoMap = [NSMutableDictionary dictionary];
[extInfoMap setObject:@"11" forKey:@"PARAM_MODULE_TYPE"];
[_txVodPlayer setExtentOptionInfo:extInfoMap];
```

```
// Disable terminal VR panoramic video.
[extInfoMap setObject:@"0" forKey:@"PARAM_MODULE_TYPE"];
```

Plugin loading configuration

After adding the terminal ultra-fast HD plugin to the project, it will be automatically loaded by default. If you do not want the player to load this plugin, you can turn it off as follows:

Android

iOS

```
TXVodPlayConfig playConfig = new TXVodPlayConfig();
playConfig.mEnableRenderProcess = false;
mVodPlayer.setConfig(playConfig);
```

```
TXVodPlayConfig *playConfig = [[TXVodPlayConfig alloc] init];
playConfig.enableRenderProcess = NO;
[_txVodPlayer setConfig:playConfig];
```

Setting obfuscation rules

In the proguard-rules.pro file, add the terminal ultra-fast HD related classes to the non-obfuscation list (iOS is not obfuscated by default):

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

Log viewing

Android

iOS

1. Terminal VR panoramic video plugin load success log:

```
D/HostEngine-PluginManger: [loadPlugin], succeed loading pluginId=2 ,pluginClazzNam
```

2. VR panoramic video feature successfully enabled log:

```
D/MonetPlugin-Process: [updateModule], moduleType=11
```

3. If the following log appears, it indicates that the issued License is invalid:

```
E/MonetPlugin-Process: [updateModule], error, reason = license is invalid!!
```

1. Terminal VR panoramic video plugin load success log:

```
[PluginsSDK] plugin config : pluginId = 2, pluginName = Monet
```

2. VR panoramic video feature successfully enabled log:

```
[MonetProcessor] PLUGIN: did update monet module, result = 1
```

3. If the following log appears, it indicates that the issued License is invalid:

```
[MonetProcessor] Monet License invalid, error, set module is null
```

API Documentation
Web

Last updated：2024-10-23 11:06:31

This document is an introduction to the parameters and API of the Web Player (TCPlayer) suitable for live and on-demand playback. It is intended for developers with a basic understanding of JavaScript.

Parameters initialization

Initializing the player requires two parameters: the first is the player's container ID, and the second is the feature parameter object.

```
var player = TCPlayer('player-container-id', options);
```

Options parameter list

The configurable parameters for the options object are as follows:

Name	Type	Default Value	Description
appID	String	No	Mandatory when playing on-demand media files through fileID, as it corresponds to the appID of the Tencent Cloud account
fileID	String	No	Mandatory when playing on-demand media files through fileID, as it is the ID of the on-demand media file
psign	String	No	Player Signature, required when playing through fileID. For more details, see Player Signature
licenseUrl	String	No	Player License address, peek at Player License
sources	Array	No	Player playback address, format: [{ src: '//path/to/video.mp4', type: 'video/mp4' }]
width	String/Number	No	Player zone width, in pixels, set as needed. The player size can be controlled through CSS.
height	String/Number	No	Player zone height, in pixels, set as needed. The player size can be controlled through CSS.

controls	Boolean	true	Whether to show the player's control bar.
poster	String	No	Set the full address of the cover image (if the uploaded video has a generated cover image, it will be used preferentially. For details, please see Video on Demand - Manage Videos).
autoplay	Boolean	false	Whether to autoplay.
playbackRates	Array	[0.5,1,1.25,1.5,2]	Set the adjustable-speed playback options, only valid in HTML5 playback pattern.
loop	Boolean	false	Whether to loop playback.
muted	Boolean	false	Whether to mute playback.
preload	String	auto	Whether preloading is needed, with 3 attributes: "auto", "meta", and "none". Due to system restrictions on mobile devices, setting auto is ineffective.
swf	String	No	URL of the Flash player's SWF file
posterImage	Boolean	true	Whether to display the cover.
bigPlayButton	Boolean	true	Whether to display a centered play button (the browser hijacked embedded play button cannot be removed).
language	String	"zh-CN"	Setting language, the options are "zh-CN"/"en"
languages	Object	No	Setting multilingual dictionary.
controlBar	Object	No	Setting the parameter combination for control bar properties, see controlBar parameter list.
reportable	Boolean	true	Setting whether to enable data reporting.
fakeFullscreen	Boolean	false	Setting up pseudo-full screen through style control to achieve the full screen effect.
plugins	Object	No	Setting the parameter combination for plugin feature attributes, see plugins plugin parameter list.
hlsConfig	Object	No	HLS.js startup configuration, for detailed content, please see the official documentation hls.js.
webrtcConfig	Object	No	WebRTC startup configuration, see webrtcConfig parameter list.

Note:

The parameters controls, playbackRates, loop, preload, posterImage will be ineffective when the browser hijacks playback.

For issues with browser hijacking video playback, see FAQs Description.

controlBar parameter list

controlBar parameters can configure the player's control bar features, with supported properties as listed in the following table:

Name	Type	Default Value	Description
playToggle	Boolean	true	Whether to display the play/pause toggle button.
progressControl	Boolean	true	Whether to display the progress bar.
volumePanel	Boolean	true	Whether to display volume control.
currentTimeDisplay	Boolean	true	Whether to show the current time of the video.
durationDisplay	Boolean	true	Whether to display the video duration.
timeDivider	Boolean	true	Whether to display the time separator.
playbackRateMenuButton	Boolean	true	Whether to display the playback speed selection button.
fullscreenToggle	Boolean	true	Whether to display the full-screen button.
QualitySwitcherMenuButton	Boolean	true	Whether to display the definition switch menu.

Please Note:

The controlBar parameter will be ineffective when the browser hijacks playback.

For issues with browser hijacking video playback, see FAQs Description.

plugins plugin parameter list

plugins parameters can configure the player's plugin features, supported properties include:

Name	Type	Default Value	Description
ContinuePlay	Object	No	Controlling the resume playback feature, supported properties are a

			auto:Boolean Whether it automatically resumes playback. text:String Prompt text. btnText:String Button text.
VttThumbnail	Object	No	Controlling thumbnail display, supported properties are as follows: vttUrl:String Absolute address of vtt file, required. basePath:String Image path, optional. If not provided, the path of vt imgUrl:String Absolute image address, optional.
ProgressMarker	Boolean	No	Controlling the display of the progress bar.
DynamicWatermark	Object	No	Controlling the display of dynamic watermarks, supporting text and are: type:String Watermark type can be text or image, with values text | content:String Text watermark content, required. speed:Number Watermark movement speed, value range 0-1, defa opacity:Number Text watermark opacity, value range 0-1, optional. fontSize:String Font size, default 12px, optional. color: String Text color, optional. left:String Text position, supports percentage and px units. When s ineffective, optional. top,right,bottom: Description same as left. width:String Image watermark width, optional. height:String Image watermark height, optional.
ContextMenu	Object	No	Valid values are as follows: mirror:Boolean Controls whether mirroring display is supported. statistic:Boolean Controls whether displaying the data panel is supp levelSwitch:Object Controls the prompt text when switching definiti `{ open: Boolean Whether to enable notification switchingText: String, Prompt text when star switchedText: String, Prompt text when the s switchErrorText: String, Prompt text when th }`
PlayList	Object	No	Setting the playlist, supported properties are as follows: `{ // Collection of video information to be p data: [{ fileID: String, appID: String, duration: Number, // Video duration text: String, // Video name`

			psign: String, // Player signature img: String, // Cover image }], title: String, // List title loop: Boolean, // Whether to loop playback }
VR	Object	No	Advanced Edition License support, for details see Web advanced f (TCPlayerVRPlugin)
SafeCheck	Object	No	Advanced Edition License support, for details see Web advanced f (TCPlayerSafeCheckPlugin)

webrtcConfig parameter list

webrtcConfig parameters control the behavior during WebRTC playback, with the following attributes supported:

Name	Type	Default Value	Description
connectRetryCount	Number	3	Number of reconnections between the SDK and the server
connectRetryDelay	Number	1	Delay in reconnection between SDK and server
receiveVideo	Boolean	true	Whether to fetch video stream
receiveAudio	Boolean	true	Whether to fetch audio stream
showLog	Boolean	false	Should logs be printed in the console

Object methods

Methods list returned by initialize player object:

Name	Parameters and Types	Return Value and Types	Description
src()	(String)	No	Setting the playback address.
loadVideoByID()	(Object)	No	When playing with fileID, you can switch videos using this method. The parameter is an object consisting of fileID, appID, psign.

ready(function)	(Function)	No	Setting the callback after player initialization is complete.
play()	No	No	Play and resume playback.
pause()	No	No	Pauses playback.
currentTime(seconds)	(Number)	(Number)	Access the current playback time, or set the playback time, which cannot exceed the video duration.
duration()	No	(Number)	Access video duration.
volume(percent)	(Number)[0,1] [Optional]	(Number)/ No return when setting	Access or set the player volume.
muted()	(Boolean)	(Boolean)	Access or set whether the player is muted
playbackRate()	(Number)[0, 1]	(Number)	Access or set the playback speed
poster(src)	(String)	(String)/ No return when setting	Access or set the player thumbnail.
requestFullscreen()	No	No	Enter full screen mode pattern.
exitFullscreen()	No	No	Exit full screen mode pattern.
isFullscreen()	No	Boolean	Return whether full screen mode pattern has been entered.
on(type,listener)	(String, Function)	No	Listening to events.
one(type,listener)	(String, Function)	No	Listening to events, the event handler executes no more than once.
off(type,listener)	(String, Function)	No	Unbind event listener.
buffered()	No	TimeRanges	Returns the video buffer range.
bufferedPercent()	No	Value range [0,1]	Returns the buffer length as a percentage of video duration.
width()	(Number) [Optional]	(Number)/ No return when setting	Access or set the player zone width. If the player size is set via CSS, this method will be ineffective.

height()	(Number) [Optional]	(Number)/ No return when setting	Access or set the player zone height. If the player size is set via CSS, this method will be ineffective.
videoWidth()	No	(Number)	Access video resolution width.
videoHeight()	No	(Number)	Access video resolution height.
dispose()	No	No	Terminate the player.

Note

Object methods cannot be called synchronously; they need to be called after the corresponding event (such as loadedmetadata) is triggered, except for ready, on, one, and off.

Event

The player can perform event listening through the objects returned by the initialization, for example:

```
var player = TCPlayer('player-container-id', options);
// player.on(type, function);
player.on('error', function(error) {
   // Do some processing
});
```

Among them, type is the event type, supported events include:

Name	Features
play	Playback has started, triggered by calling the play() method or setting autoplay to true and it takes effect, with the paused attribute being false.
playing	Triggered when playback resumes after being paused or stopped for buffering, with the paused attribute being false. This event is commonly used to mark the real start of video playback, as the play event only indicates the initiation of playback without actual video rendering.
loadstart	Triggered when starting to load data.
durationchange	Triggered when the video's duration data changes.
loadedmetadata	The video's metadata has been loaded.
loadeddata	This event is triggered when the data for the current frame is loaded, but there is not enough data to play the next frame of the video.

progress	Triggered when accessing media data.
canplay	Triggered when the player is able to start playing the video.
canplaythrough	Triggered when the player estimates it can play through the specified video without stopping for buffering.
error	Triggered when an error occurs during video playback.
pause	Triggered when paused.
blocked	Triggered when autoplay is blocked by the browser.
ratechange	Triggered when the playback speed changes.
seeked	Triggered when the search for a specified playback position ends.
seeking	Triggered when the search for a specified playback position begins.
timeupdate	Triggered when there is a change in the current playback position, which can be understood as a change in currentTime.
volumechange	Triggered when the volume setting or muted property value changes.
waiting	Triggered when playback stops, and the next frame of content is unavailable.
ended	Triggered when video playback has ended. At this point, the value of currentTime is equal to the maximum value of the media resource.
resolutionswitching	Definition switch in progress.
resolutionswitched	Definition switch completed.
fullscreenchange	Triggered during full screen status toggle.
webrtcevent	Event set during webrtc playback.
webrtcstats	Statistics during webrtc playback.
webrtcfallback	Triggered degradation during webrtc playback

WebrtcEvent list

The player can access all events during webrtc playback through webrtcevent, for example:

```
var player = TCPlayer('player-container-id', options);
player.on('webrtcevent', function(event) {
   // Access event status code and related data from the callback parameter event
});
```

webrtcevent status codes are as follows

Status Code	Callback parameters	Features
1001	No	Start pulling stream
1002	No	Connected to server
1003	No	Video playback starts
1004	No	Stop pulling stream, end video playback
1005	No	Failed to connect to server. Auto-reconnect initiated to recover
1006	No	Access stream data is empty
1007	localSdp	Starting signaling server request
1008	remoteSdp	Signaling server request succeeded
1009	No	Streaming lagging, buffering in progress
1010	No	Streaming lag ended, playback resumed

Error code

When the player triggers an error event, the listener function returns an error code, with those having three or more digits being media data interface error codes. Error code list:

Name	Description
-1	The player did not detect an available video address.
-2	Access to video data timed out.
1	Video data loading was interrupted. Possible reasons: Network interrupted. Browser abnormal interruption. Solution: Peek at the network request information in the web console to confirm if network requests are normal. Restart the playback process.
2	Video loading failed due to network issues. Possible cause: Network interrupts. Solution:

	Peek at the network request information in the web console to confirm if network requests are normal. Restart the playback process.
3	An error occurred during video decoding. Possible cause: Abnormal video data, decoder failed to decode. Solution: Try re-transcoding and then play it again to eliminate issues introduced by the transcoding process. Confirm whether the original video is normal. Please contact technical support and provide play parameters for troubleshooting.
4	The video cannot be loaded due to unsupported format or server/network issues. Possible reasons: Cannot access video data, CDN resource does not exist or does not return video data. The current playback environment does not support this video format. Solution: Peek at the network request information in the browser console to confirm if the video data request is normal. Confirm whether the playback script for the corresponding video format has been loaded as per the documentation. Confirm whether the current browser and page environment support the video format to be played. Please contact technical support and provide play parameters for troubleshooting.
5	An error occurred during video decryption. Possible reasons: The decryption key is incorrect. The request for the key API returned an exception. The current playback environment does not support the video decryption feature. Solution: Confirm whether the key is correct and whether the key API returns normally. Please contact technical support and provide play parameters for troubleshooting.
10	The VOD media data interface request timed out. When accessing media data, if the player retries 3 times without any response, this error will be thrown. Possible reasons: The current network environment cannot connect to the media data interface, or the media data interface has been hijacked. Media data interface error. Solution: Try opening the demo page we provided to see if it can play normally. Please contact technical support and provide play parameters for troubleshooting.
11	The VOD media data interface did not return any data. When accessing media data, if the player retries 3 times without any data returned, this error will be thrown. Possible reasons: The current network environment cannot connect to the media data interface, or the media data interface has been hijacked.

	Media data interface error. Solution: Try opening the demo page we provided to see if it can play normally. Please contact technical support and provide play parameters for troubleshooting.
12	The VOD media data interface returns abnormal data. When accessing media data, if the player retries 3 times and still returns data that cannot be parsed, this error will be thrown. Possible reasons: The current network environment cannot connect to the media data interface, or the media data interface has been hijacked. The play parameters are incorrect; the media data interface cannot process them. Media data interface error. Solution: Try opening the demo page we provided to see if it can play normally. Please contact technical support and provide play parameters for troubleshooting.
13	The player did not detect video data that can be played on the current player; please transcode this video.
14	Playing HLS under the HTML5 + hls.js pattern encounters a network error. Details can be peeked at in event.source. For a detailed introduction, please see the official hls.js documentation Network Errors.
15	Playing HLS under the HTML5 + hls.js pattern encounters a Multimedia error. Details can be peeked at in event.source. For a detailed introduction, please see the official hls.js documentation Media Errors.
16	Playing HLS under the HTML5 + hls.js pattern encounters a multiplexing exception. Details can be peeked at in event.source. For a detailed introduction, please see the official hls.js documentation Mux Errors.
17	Playing HLS under the HTML5 + hls.js pattern encounters other exceptions. Details can be peeked at in event.source. For a detailed introduction, please see the official hls.js documentation Other Errors.
1013	Player Signature is missing the contentInfo field
10008	The media data service did not find media data corresponding to the play parameters, please confirm that the appID and fileID request parameters are correct, and that the corresponding media data has not been deleted.
-2002	Live Event Broadcasting pull stream interface returns an error in the backend (for example, the stream does not exist, authentication failure, etc.)
-2006	Live Event Broadcasting multi-resolution smooth switching interface request failed

iOS
TXVodPlayer

Last updated：2025-05-30 14:52:38

TXVodPlayer API Introduction

TXVodPlayer serves as the core playback class, primarily handling playback controls including play/pause operations and speed adjustment. For comprehensive player capabilities, please refer to the Player SDK feature specifications.

Interface Overview

Basic playback interface

API	Description
startVodPlay:	To play HTTP URL-formatted addresses: Starting from version 10.7, startPlay has been renamed to startVodPlay. You must set the Licence via `+[TXLiveBase setLicenceURL:key:]` before playback will succeed; otherwise, playback will fail (black screen). The Licence only needs to be set once globally. Live License, Short Video License, and Player License are all acceptable for this use case. If you haven't obtained any of these licenses yet, you may click 'Player License' to apply. Please note that formal version licenses require purchase.
startVodPlayWithParams:	To play using fileId, pass the TXPlayInfoParams parameter. Starting from version 10.7, 'startPlay' has been renamed to 'startVodPlay'. You must configure the license using `+[TXLiveBase setLicenceURL:key:]` for successful playback - otherwise playback will fail (black screen). The license only needs to be set once globally. Live License, Short Video License, and Player License are all acceptable for this use case. If you haven't obtained any of these licenses yet, you may click 'Player License' to apply. Please note that formal version licenses require purchase.
startPlayDrm:	Start a standard FairPlay DRM playback.
stopPlay	Stops the audio/video stream.

isPlaying	Gets whether playback is ongoing.
pause	Pauses playback. The player will stop pulling data and freeze on the last frame.
resume	Resumes playback. The player will resume pulling data.
seek	Seeks to the specified time point of the video stream (in seconds).
seek:accurateSeek:	Seeks to the specified time point in the video stream, in seconds, with an accuracy of 3 decimal places. Supports precise seek.
currentPlaybackTime	Gets the current playback time point in seconds.
duration	Gets the total video duration in seconds.
playableDuration	Gets the playable video duration in seconds.
width	Gets the video width.
height	Gets the video height.
isAutoPlay	Sets whether to start playback automatically after `startVodPlay` is called. Auto-play is on by default.
enableHWAcceleration	Sets whether to enable hardware acceleration.
setStartTime:	Sets the playback start time.
setupVideoWidget:insertIndex:	Create a Video rendering View, which hosts the display of video content.
removeVideoWidget	Remove Video rendering View .
token	Sets the token for HLS encryption.
getEncryptedPlayKey:	Get the encrypted playback key for content protection.
loop	Set whether to loop playback.
addSubtitleSource:name:mimeType:	Add external subtitles (supported by the premium version of the player).
getSubtitleTrackInfo	Returns the list of subtitle track information (only supported by the premium version of the player).
getAudioTrackInfo	Returns the list of audio track information (supported only by the premium version of the player).
selectTrack:	Select a track (supported only in the premium version of the player).

deselectTrack:	Deselects a track (only supported in premium versions of the player).
seekToPdtTime:	Seek to the specified Program Date Time (PDT) point in the video stream, which enables functions such as fast-forward, rewind, and progress bar jumping. Currently, only HLS video format is supported. This function is supported starting from version 11.6 of the player's premium edition. The parameter unit is milliseconds (ms).

Player configuration APIs

API	Description
config	Configures VOD. For more information on the configuration, see TXVodPlayConfig.
setExtentOptionInfo:	Sets the player business parameters in the format of `<NSString *, id>` .
setSubtitleStyle:	Set the subtitle style information, and the subtitle style can be updated after playback (only supported by the premium version of the player).
setAudioNormalization:	Setting volume normalization, loudness range: -70 to 0 (LUFS). Supported from player version 11.7 of the premium version. You can set the volume normalization to preset values or custom values. The preset values are defined in the related classes or files, such asTXVodConstants for Android and TXVodPlayConfig.hfor iOS: Off: AUDIO_NORMALIZATION_OFF On: AUDIO_NORMALIZATION_STANDARD (standard) AUDIO_NORMALIZATION_LOW (low) AUDIO_NORMALIZATION_HIGH (high) Custom values can be set in the range of -70 to 0 LUFS, from low to high.

Video related interfaces

API	Description
enableHWAcceleration	Sets whether to enable hardware acceleration.
snapshot:	Gets the current video frame image. **Note: Because this operation is time-consuming, the screenshot will be called back asynchronously.**
setMirror:	Sets whether to flip the video image.
setRate:	Sets the VOD playback speed. Default value: 1.0.
bitrateIndex	Returns the current playback bitrate index.

supportedBitrates	When the playback address is a master playlist, return the supported bitrates (resolutions).
setBitrateIndex:	Sets the current playback bitrate index for seamless definition switch. You may need to wait momentarily to switch the definition.
setRenderMode:	Sets the image fill mode.
setRenderRotation:	Sets the rotation.
setAutoMaxBitrate:	Set the highest bitrate that can be switched for adaptive playback.

Picture-in-picture related interfaces

API	Description
isSupportPictureInPicture	Whether Picture-in-Picture (PiP) function is supported.
isSupportSeamlessPictureInPicture	Whether to support seamless switching of picture-in-picture function. (Only available in the Premium version of the player).
setAutoPictureInPictureEnabled:	Set whether to automatically start picture-in-picture.(Only available in the Premium version of the player).
enterPictureInPicture	Enter picture-in-picture.
exitPictureInPicture	Exit picture-in-picture.

Audio related interfaces

API	Description
setMute:	Sets whether to mute the player.
setAudioPlayoutVolume:	Sets the volume level. Value range: 0–100.

Event Notification Interface

API	Description
vodDelegate	Set the player event callback object.
videoProcessDelegate	Set the video rendering callback object.

TRTC video playback related interfaces

You can push the VOD player's audio streams via TRTC using the following APIs. For more TRTC services, refer to the TRTC Product Overview.

API	Description
attachTRTC:	Binds VOD to TRTC.
detachTRTC	Unbinds VOD from TRTC.
publishVideo	Starts pushing the video stream.
unpublishVideo	Cancels pushing the video stream.
publishAudio	Starts pushing the audio stream.
unpublishAudio	Cancels pushing the audio stream.

Interface Details

startVodPlay:

Start playback via a URL.。
Starting from version 10.7, you need to set the license through `+[TXLiveBase setLicenceURL:key:]` to play successfully, otherwise the playback will fail (black screen). You only need to set it once globally.

```
- (int)startVodPlay:(NSString *)url;
```

Parameter Description

Parameter Name	Type	Description
url	NSString	Playback address.

Return Value

0： Playback successful.
Non-zero： Playback failed.

startVodPlayWithParams:

To play using fileId, pass the TXPlayInfoParams parameter.
Starting from version 10.7, 'startPlay' has been renamed to 'startVodPlay'. You must configure the license using `+[TXLiveBase setLicenceURL:key:]` for successful playback - otherwise playback will fail (black screen). The license only needs to be set once globally.

```
- (int)startVodPlayWithParams:(TXPlayerAuthParams *)params;
```

Parameter Description

Parameter Name	Type	Description
params	TXPlayerAuthParams	Video fieldID and information.

Return Value

0： Playback successful.

Non-zero： Playback failed.

startPlayDrm:

Play Drm encrypted video.

Starting from version 10.7, 'startPlay' has been renamed to 'startVodPlay'. You must configure the license using `+[TXLiveBase setLicenceURL:key:]` for successful playback - otherwise playback will fail (black screen). The license only needs to be set once globally.

Note:

Only the premium version of the player supports it.

```
- (int)startPlayDrm:(TXPlayerDrmBuilder *)drmBuilder;
```

Parameter Description

Parameter Name	Type	Description
drmBuilder	TXPlayerDrmBuilder	Drm playback information.

Return Value

0: Playback successful.

Non-zero: Playback failed.

stopPlay

Stop playback.

```
- (int)stopPlay;
```

Return Value

0: Stop successful.

Non-zero: Stop failed.

isPlaying

Whether it is playing.

```
- (bool)isPlaying;
```

pause

Pause playback.

```
- (void)pause;
```

resume

Resume playback.

```
- (void)resume;
```

seek:

Jump to a specified time point in the video stream.

```
- (int)seek:(float)time;
```

Parameter Description

Parameter Name	Type	Description
time	int	Video stream time point, in seconds.

Return Value

0: Seek successful.

Non-zero: Seek failed.

seek:accurateSeek:

Jump to a specified time point in the video stream.

```
- (void)seek:(float)time accurateSeek:(BOOL)isAccurateSeek;
```

Parameter Description

Parameter Name	Type	Description
time	float	Video stream time point, in seconds, accurate to 3 decimal places.
isAccurateSeek	BOOL	Whether the Seek is accurate.

		YES: Indicates an accurate Seek. Must find the current time point. This will take longer. NO: Indicates an imprecise Seek, that is, searching for the previous I-frame.

seekToPdtTime:

Navigate to the specified time point in the video stream.

Note:

Player premium 11.6 edition starts to support.

```
- (void)seekToPdtTime:(long long)pdtTimeMs;
```

Parameter Description

Parameter Name	Type	Description
pdtTimeMs	long long	Video stream PDT time point, unit milliseconds.

currentPlaybackTime

Retrieve the current playback time point, in seconds.

```
- (float)currentPlaybackTime;
```

duration

Retrieve the total video duration, in seconds.

```
- (float)duration;
```

playableDuration

Retrieve the current playable video duration, in seconds.

```
- (float)playableDuration;
```

width

Get video width.

```
- (int)width;
```

height

Retrieve video height.

```
- (int)height;
```

setupVideoWidget:insertIndex:

Set the player to render View Host the display of video content

```
- (void)setupVideoWidget:(NSView *)view insertIndex:(unsigned int)idx;
```

removeVideoWidget

Remove the player's render View.

```
- (void)removeVideoWidget;
```

isAutoPlay

Set whether to autoplay after startPlay. Autoplay is enabled by default.

```
@property BOOL isAutoPlay;
```

setStartTime:

Set the playback start time in seconds. It needs to be set before starting playback.

```
- (void)setStartTime:(CGFloat)startTime;
```

Parameter Description

Parameter Name	Type	Description
startTime	CGFloat	Video stream time point, in seconds, accurate to 3 decimal places.

token

Set the encryption token for HLS. After setting this value, the player automatically adds `voddrm.token` before the file name in the URL.

```
@property(nonatomic, strong) NSString *token;
```

getEncryptedPlayKey:

Retrieve the encrypted playback key for reinforcement.

```
+ (NSString *)getEncryptedPlayKey:(NSString *)key;
```

loop

Whether to loop playback.

```
@property(nonatomic, assign) BOOL loop;
```

addSubtitleSource:name:mimeType:

Add external subtitle.

Note:

Player premium version only supports.

```
- (void)addSubtitleSource:(NSString *)url name:(NSString *)name mimeType:(TX_VOD_PL
```

Parameter Description

Parameter Name	Type	Description
url	NSString	Subtitle address, supports Http link and local storage absolute path.
name	NSString	Subtitle name. If adding multiple subtitles, set the subtitle names to different names for distinguishing from other added subtitles; otherwise, it may lead to subtitle selection error.
mimeType	TX_VOD_PLAYER_SUBTITLE_MIME_TYPE	Subtitle type, only supports VVT and SRT formats. For details, see TX_VOD_PLAYER_SUBTITLE_MIME_TYPE.

getSubtitleTrackInfo

Return to subtitle track information list.

Note:

Player premium version only supports.

```
- (NSArray<TXTrackInfo *> *)getSubtitleTrackInfo;
```

Return Value

NSArray<TXTrackInfo *> * : subtitle track information list.

getAudioTrackInfo

Return to audio track information list.

Note:

Player premium version only supports.

```
- (NSArray<TXTrackInfo *> *)getAudioTrackInfo;
```

Return Value

NSArray<TXTrackInfo *> * : audio track information list.

selectTrack:

Select a track.

Note:

Player premium version only supports.

```
- (void)selectTrack:(NSInteger)trackIndex;
```

Parameter Description

Parameter Name	Type	Description
trackIndex	NSInteger	Track index, obtained through `-[TXTrackInfo getTrackIndex]` .

deselectTrack:

Cancel track selection.

Note:

Player premium version only supports.

```
- (void)deselectTrack:(NSInteger)trackIndex;
```

Parameter Description

Parameter Name	Type	Description
trackIndex	NSInteger	Track index, obtained through `-[TXTrackInfo getTrackIndex]` .

config

Set player configuration information, refer to TXVodPlayConfig.

```
@property(nonatomic, copy) TXVodPlayConfig *config;
```

setExtentOptionInfo:

Set player business parameters, parameter format is `<NSString, id>` .

```
- (void)setExtentOptionInfo:(NSDictionary<NSString *, id> *)extInfo;
```

setSubtitleStyle:

Set subtitle style information, supports pre-play configuration, and can support updating subtitle styles after playback.

Note:

Player premium version only supports.

```
- (void)setSubtitleStyle:(TXPlayerSubtitleRenderModel *)renderModel;
```

Parameter Description

Parameter Name	Type	Description
renderModel	TXPlayerSubtitleRenderModel	Subtitle style configuration parameters.

setAudioNormalization:

Set volume balance, loudness range: -70 to 0 (LUFS).

Note:

Player premium version only supports.

```
- (void)setAudioNormalization:(float)value;
```

Parameter Description

Parameter Name	Type	Description
value	float	Preset value can be filled in (related class or file: Android: TXVodConstants; iOS: TXVodPlayConfig.h) Off: AUDIO_NORMALIZATION_OFF On: AUDIO_NORMALIZATION_ON AUDIO_NORMALIZATION_STANDARD AUDIO_NORMALIZATION_LOW AUDIO_NORMALIZATION_HIGH Custom value can be entered: ranging from low to high, range -70 to 0 LUFS.

setAutoMaxBitrate:

Set switchable adaptive playback with maximum bit rate.

```
- (void)setAutoMaxBitrate:(NSInteger)autoMaxBitrate;
```

enableHWAcceleration

Enable or disable video hardware decoding, hardware decoding is enabled by default.

```
@property(nonatomic, assign) BOOL enableHWAcceleration;
```

snapshot:

Retrieve the current video frame image.

```
- (void)snapshot:(void (^)(UIImage *))snapshotCompletionBlock;
```

Parameter Description

Parameter Name	Type	Description
snapshotCompletionBlock	void (^)(UIImage *)	Screenshot callback interface class

setMirror:

Set mirror video.

```
- (void)setMirror:(BOOL)isMirror;
```

setRate:

Set the playback speed of on-demand video. Default is 1.0.

```
- (void)setRate:(float)rate;
```

Parameter Description

Parameter Name	Type	Description
rate	float	Playback speed [0.5, 3.0].

bitrateIndex

Return the bitrate index of current playback.

```
- (NSInteger)bitrateIndex;
```

supportedBitrates

Return the list of supported bitrates (resolution) when the playback address is HLS.

```
- (NSArray<TXBitrateItem *> *)supportedBitrates;
```

Return Value

NSArray<TXBitrateItem *> : Bitrate list.

setBitrateIndex:

Set the bitrate index of current playback. Tencent Cloud supports multi-bitrate HLS fragment alignment to ensure the best experience. When switching clarity seamlessly, clarity switching may require waiting for a short while.

```
- (void)setBitrateIndex:(NSInteger)index;
```

Parameter Description

Parameter Name	Type	Description
index	NSInteger	bitrate index. index == -1 indicates enabling HLS adaptive bitrate streaming. index > 0 indicates manually switching to the corresponding clarity bitrate. The index value can be set through the API. - Retrieve [TXVodPlayer supportedBitrates].

setRenderMode:

Set the player's image tiling mode.

```
- (void)setRenderMode:(TX_Enum_Type_RenderMode)renderMode;
```

Parameter Description

Parameter Name	Type	Description
renderMode	TX_Enum_Type_RenderMode	Image Tiling Mode, valid values: RENDER_MODE_FILL_SCREEN: The video fills the entire screen with the image proportionally scaled to cover the whole screen, cropping any excess parts. In this mode, there is no black border around the image. RENDER_MODE_FILL_EDGE: The video image self-adaptively scales proportionally to fit the screen, with the scaled width and height not exceeding the display area. It is displayed in the center and may leave black borders.

setRenderRotation:

Set the player's image rendering angle.

```
- (void)setRenderRotation:(TX_Enum_Type_HomeOrientation)rotation;
```

Parameter Description

Parameter Name	Type	Description
rotation	TX_Enum_Type_HomeOrientation	Image rendering angle, valid values: HOME_ORIENTATION_RIGHT: The Home button is on the right. HOME_ORIENTATION_DOWN: The Home button is at the bottom. HOME_ORIENTATION_LEFT: The Home button is on the left. HOME_ORIENTATION_UP: The Home button is at the top.

setMute:

Set whether to play with mute. Default is unmuted playback.

```
- (void)setMute:(BOOL)bEnable;
```

setAudioPlayoutVolume:

Set volume level, range: 0 - 150, default is 100.

```
- (void)setAudioPlayoutVolume:(int)volume;
```

vodDelegate

Set player event callback object.

```
@property(nonatomic, weak) id<TXVodPlayListener> vodDelegate;
```

videoProcessDelegate

Set video rendering callback object.

```
@property(nonatomic, weak) id<TXVideoCustomProcessDelegate> videoProcessDelegate;
```

attachTRTC

Bind on-demand to TRTC service.

```
- (void)attachTRTC:(NSObject *)trtcCloud;
```

detachTRTC

Unbind on-demand from TRTC service.

```
- (void)detachTRTC;
```

publishVideo

Start pushing video stream.

```
- (void)publishVideo;
```

unpublishVideo

Cancel pushing video stream.

```
- (void)unpublishVideo;
```

publishAudio

Start pushing audio stream.

```
- (void)publishAudio;
```

unpublishAudio

Cancel pushing audio stream.

```
- (void)unpublishAudio;
```

isSupportPictureInPicture

Whether Picture-In-Picture functionality is supported.

```
+ (BOOL)isSupportPictureInPicture;
```

isSupportSeamlessPictureInPicture

Whether seamless switching of Picture In Picture functionality is supported. The premium edition player SDK is required.

```
+ (BOOL)isSupportSeamlessPictureInPicture;
```

setAutoPictureInPictureEnabled:

Set whether to automatically start Picture In Picture (automatic startup control switch for Picture In Picture).

```
- (void)setAutoPictureInPictureEnabled:(BOOL)enabled;
```

Parameter Description

Parameter Name	Type	Description
enabled	BOOL	YES: Automatically enters Picture-in-Picture when going into the background. NO: Automatic entry into Picture-in-Picture is not allowed.

enterPictureInPicture

Enter the Picture-in-Picture function (this method needs to be called after Prepared).

```
- (void)enterPictureInPicture;
```

exitPictureInPicture

Exit the Picture-in-Picture function.

```
- (void)exitPictureInPicture;
```

TXVodPlayListener

Last updated：2025-05-30 14:53:26

TXVodPlayListener API Introduction

Playback event and network event callback monitoring API for on-demand player

Callback API Overview

API	Description
onPlayEvent:event:withParam:	Playback event notification.
onNetStatus:withParam:	Playback process network status event callback.
onPlayer:subtitleData:	Subtitle data callback.
onPlayer:pictureInPictureStateDidChange:withParam:	Picture-in-Picture state callback.
onPlayer:pictureInPictureErrorDidOccur:withParam:	Picture-in-Picture error message callback.
onPlayer:airPlayStateDidChange:withParam:	AirPlay state callback (system player only).
onPlayer:airPlayErrorDidOccur:withParam:	AirPlay error message callback (system player only).

Callback API Details

onPlayEvent:event:withParam:

Playback event notification, including start playback, first frame event, Loading event, playback progress, end playback, and other events.

```
- (void)onPlayEvent:(TXVodPlayer *)player event:(int)EvtID withParam:(NSDictionary
```

Parameter Description

Parameter Name	Type	Description
player	TXVodPlayer	Current player object.
EvtID	int	Player event, see TXVODEventID for details.

param	NSDictionary	Playback event carried parameters, saved in (Key, Value) format, where Key can refer to the event parameters in TXVodSDKEventDef.

onNetStatus:withParam:

Playback process network status event callback.

```
- (void)onNetStatus:(TXVodPlayer *)player withParam:(NSDictionary *)param;
```

Parameter Description

Parameter Name	Type	Description
player	TXVodPlayer	Current player object.
param	NSDictionary	Network status parameters during playback, in the format: (Key, Value). Specific Keys can be found in TXVodSDKEventDef.

onPlayer:subtitleData:

Subtitle data callback.

```
- (void)onPlayer:(TXVodPlayer *)player subtitleData:(TXVodSubtitleData *)subtitleDa
```

Parameter Description

Parameter Name	Type	Description
player	TXVodPlayer	Current player object.
subtitleData	TXVodSubtitleData	Subtitle callback data. For details, see TXVodSubtitleData.

onPlayer:pictureInPictureStateDidChange:withParam:

Picture-in-Picture state callback.

```
- (void)onPlayer:(TXVodPlayer *)player pictureInPictureStateDidChange:(TX_VOD_PLAYE
```

Parameter Description

Parameter Name	Type	Description
player	TXVodPlayer	Callback player object.
pipState	TX_VOD_PLAYER_PIP_STATE	Picture-in-picture controller status. Specific values

		are listed in TX_VOD_PLAYER_PIP_STATE.
param	NSDictionary	Additional parameter.

onPlayer:pictureInPictureErrorDidOccur:withParam:

Picture-in-Picture error message callback.

```
- (void)onPlayer:(TXVodPlayer *)player pictureInPictureErrorDidOccur:(TX_VOD_PLAYER
```

Parameter Description

Parameter Name	Type	Description
player	TXVodPlayer	Callback player object.
errorType	TX_VOD_PLAYER_PIP_ERROR_TYPE	Error type. For details, see TX_VOD_PLAYER_PIP_ERROR_TYPE.
param	NSDictionary	Error message.

onPlayer:airPlayStateDidChange:withParam:

AirPlay state callback (system player only).

```
- (void)onPlayer:(TXVodPlayer *)player airPlayStateDidChange:(TX_VOD_PLAYER_AIRPLAY
```

Parameter Description

Parameter Name	Type	Description
player	TXVodPlayer	Callback player object.
airPlayState	TX_VOD_PLAYER_AIRPLAY_STATE	AirPlay status. Specific values are listed in TX_VOD_PLAYER_AIRPLAY_STATE.
param	NSDictionary	Additional parameter.

onPlayer:airPlayErrorDidOccur:withParam:

AirPlay error message callback (system player only).

```
- (void)onPlayer:(TXVodPlayer *)player airPlayErrorDidOccur:(TX_VOD_PLAYER_AIRPLAY_
```

Parameter Description

Parameter Name	Type	Description
player	TXVodPlayer	Callback player object.
errorType	TX_VOD_PLAYER_AIRPLAY_ERROR_TYPE	AirPlay error type. For details, see TX_VOD_PLAYER_AIRPLAY_ERROR_TYPE.
param	NSDictionary	Error message.

Player Config
TXPlayerGlobalSetting

Last updated：2025-05-30 14:54:19

TXPlayerGlobalSetting API Introduction

VOD player global configuration.。

API Overview

API	Description
setCacheFolderPath:	Set the Cache directory for the playback engine.
cacheFolderPath	Retrieve the Cache directory of the playback engine.
setMaxCacheSize:	Set the maximum cache size for the playback engine.
maxCacheSize	Retrieve the maximum cache size of the playback engine.
getOptions:	Judge the feature capability of the player.
setLicenseFlexibleValid:	Set flexible verification for player License. Once enabled, the first 3 playback verifications will default allow after the player's first launch.
setPlayCGIHosts:	Set the list of domain addresses for Tencent Cloud PlayCGI hosts.

API Detail

setCacheFolderPath:

Set the Cache directory for the playback engine. After configuration, the player and preloading will preferentially read from and store in this directory.

```
+ (void)setCacheFolderPath:(NSString *)cacheFolder;
```

Parameter Description

Parameter Name	Type	Description

cacheFolder	NSString	Cache directory path, nil means caching is disabled.

cacheFolderPath

Retrieve the Cache directory set for the playback engine.

```
+ (NSString *)cacheFolderPath;
```

setMaxCacheSize:

Set the maximum cache size for the playback engine. After configuration, files in the Cache directory will be automatically cleaned up based on the set value.

```
+ (void)setMaxCacheSize:(NSInteger)maxCacheSizeMB;
```

Parameter Description

Parameter Name	Type	Description
maxCacheSizeMB	NSInteger	Maximum cache size, unit: MB.

maxCacheSize

Retrieve the set maximum cache size of the playback engine. Unit: MB.

```
+ (NSInteger)maxCacheSize;
```

getOptions:

Judge the feature capability of the player.

```
+ (id)getOptions:(NSNumber *)featureId;
```

setLicenseFlexibleValid:

Enable flexible verification for player License. Once enabled, the first 2 playback verifications will default allow after the player's first launch.

```
+ (void)setLicenseFlexibleValid:(BOOL)value;
```

setPlayCGIHosts:

Set the list of domain addresses for Tencent Cloud PlayCGI hosts. When the request to the built-in domain fails, the set backup domain name will be enabled.

```
+ (void)setPlayCGIHosts:(NSArray<NSString *> *)hosts;
```

Parameter Description

Parameter Name	Type	Description
hosts	NSArray<NSString *>	List of domain addresses, domain name format: playvideo.qcloud.com. Trigger a PlayCGI request and sequentially use the passed-in host addresses. If a request to a certain host fails, automatically switch to the next host and retry the request.

TXVodPlayConfig

Last updated：2025-05-30 14:48:52

TXVodPlayConfig API Introduct

on-demand player playback configuration, need to be set before playback.

TX_Enum_MP4EncryptionLevel (MP4 Encrypted Playback Level)

Value	Parameter Name	Description
0	MP4_ENCRYPTION_LEVEL_NONE	Unencrypted.
1	MP4_ENCRYPTION_LEVEL_L1	L1 (Online Encryption)
2	MP4_ENCRYPTION_LEVEL_L2	L2 (Local Encryption)

TX_Enum_PlayerType (Player Type Definition)

Value	Parameter Name	Description
0	PLAYER_AVPLAYER	system player.
1	PLAYER_THUMB_PLAYER	Based on FFmpeg, supports software decoding, with better compatibility.

TX_Enum_VideoResolution (Player Preferred Resolution)

Value	Parameter Name	Description
720 * 1280	VIDEO_RESOLUTION_720X1280	Resolution 720X1280.
1080 * 1920	VIDEO_RESOLUTION_1080X1920	Resolution 1080X1920.
1440 * 2560	VIDEO_RESOLUTION_1440X2560	Resolution 1440X2560.

2160 * 3840	VIDEO_RESOLUTION_2160X3840	Resolution 2160X3840.

TX_Enum_MediaType (Media Asset Type)

Value	Parameter Name	Description
0	MEDIA_TYPE_AUTO	AUTO type (default value, adaptive bit rate playback not supported).
1	MEDIA_TYPE_HLS_VOD	HLS VOD media.
2	MEDIA_TYPE_HLS_LIVE	HLS live media.
3	MEDIA_TYPE_FILE_VOD	Common file VOD media such as MP4.
4	MEDIA_TYPE_DASH_VOD	DASH VOD media.

TX_Enum_Video_Pixel_Format (Video Frame Output Type)

Value	Parameter Name	Description
0	TX_VIDEO_PIXEL_FORMAT_NONE	Invalid type.
1	TX_VIDEO_PIXEL_FORMAT_VideoToolbox	VIDEO TOOL BOX, direct original video format output.
2	TX_VIDEO_PIXEL_FORMAT_RGBA	RGBA format (Since Apple does not recommend using RGBA, please use BGRA format as an alternative).
3	TX_VIDEO_PIXEL_FORMAT_BGRA	BGRA format.

API Overview

API	Description
connectRetryCount	Set the player's reconnection attempts in abnormal scenarios.

connectRetryInterval	Player connection retry interval, in seconds. Minimum value is 3, maximum value is 30. Default value is
timeout	Set the player connection timeout period.
videoFrameFormatType	Video format for video rendering object callback.
keepLastFrameWhenStop	Retain the last frame after playback stops, default is NO.
firstStartPlayBufferTime	Initial buffer data duration, unit ms, default value 100ms.
nextStartPlayBufferTime	Maximum buffer duration caused by insufficient buffer data or drag buffer during seek, unit ms, default value 250ms.
playerType	Set playe
headers	Set Http header.
enableAccurateSeek	Set whether to seek accurately. The default is YES.
autoRotate	Set whether to automatically rotate the angle when playing MP4. The default is YES.
smoothSwitchBitrate	Set whether to smoothly switch between multiple bitrates for HLS. The default is NO.
progressInterval	Set the interval for progress callbacks. The default is 500ms.
maxBufferSize	Set the maximum playback buffer size for the player.
maxPreloadSize	Set the maximum preload buffer size.
overlayKey	Set encryption key.
overlayIv	Encryption Iv.
enableRenderProcess	Set whether the player is allowed to load post-processing services. Default is NO.
preferredResolution	Set the optimal HLS stream for playback.
encryptedMp4Level	Set MP4 encrypted playback.
mediaType	Set the media asset type for player playback
extInfoMap	Set player expansion parameters.
preferAudioTrack	Set the preferred audio track for startup playback.

API Detail

connectRetryCount

Set the player's reconnection attempts in abnormal scenarios.
When the SDK is disconnected from the server due to an exception, it will attempt to reconnect. Use this function to set the SDK's reconnection attempts, with a default value of 3.

```
@property(nonatomic, assign) int connectRetryCount;
```

connectRetryInterval

Set the player's reconnection interval duration in abnormal scenarios.
in seconds, minimum value is 3, maximum value is 30, default value is 3.

```
@property(nonatomic, assign) int connectRetryInterval;
```

timeout

Set the player connection timeout period, with a default value of 10 seconds.

```
@property(nonatomic, assign) NSTimeInterval timeout;
```

videoFrameFormatType

Set the video format for video rendering object callback. The default value is TX_VIDEO_PIXEL_FORMAT_NONE.

```
@property(nonatomic, assign) TX_Enum_Video_Pixel_Format videoFrameFormatType;
```

Parameter Description

Parameter Name	Type	Description
videoFrameFormatType	TX_Enum_Video_Pixel_Format	Video frame callback type. For details, see TX_Enum_Video_Pixel_Format.

keepLastFrameWhenStop

Retain the last frame after the player stops, default is NO.

```
@property(nonatomic, assign) BOOL keepLastFrameWhenStop;
```

firstStartPlayBufferTime

Initial buffer data duration, unit ms. Default value 100ms.

```
@property(nonatomic, assign) int firstStartPlayBufferTime;
```

Parameter Description

Parameter Name	Type	Description
firstStartPlayBufferTime	int	Duration size.

nextStartPlayBufferTime

Minimum buffer duration required to end buffering due to insufficient buffer data or drag buffer during seek, unit ms, default value 250ms.

```
@property(nonatomic, assign) int nextStartPlayBufferTime;
```

playerType

Set the player type. The default is Tencent Cloud's proprietary player.

```
@property(nonatomic, assign) NSInteger playerType;
```

Parameter Description

Parameter Name	Type	Description
playerType	int	Player type, valid values: PLAYER_AVPLAYER: iOS system player. PLAYER_THUMB_PLAYER: Tencent Cloud's proprietary player (default).

headers

Customize configuration for the player to play Http headers carried during the online process.

```
@property(nonatomic, strong) NSDictionary *headers;
```

Parameter Description

Parameter Name	Type	Description
headers	NSDictionary <NSString *, NSString *>	Custom Http header content.

enableAccurateSeek

Set whether to seek accurately. The default is true.

```
@property(nonatomic, assign) BOOL enableAccurateSeek;
```

Parameter Description

Parameter Name	Type	Description
enableAccurateSeek	BOOL	Set whether to seek accurately.

autoRotate

When playing an MP4 file, if set to YES, it will automatically rotate according to the rotation angle in the file. The rotation angle can be obtained in the VOD_PLAY_EVT_CHANGE_ROTATION event. The default value is YES.

```
@property(nonatomic, assign) BOOL autoRotate;
```

Parameter Description

Parameter Name	Type	Description
autoRotate	BOOL	Whether to automatically rotate based on the rotation angle during playback.

smoothSwitchBitrate

Set whether to smoothly switch between multiple bitrates for HLS. Default is NO.

```
@property(nonatomic, assign) BOOL smoothSwitchBitrate;
```

Parameter Description

Parameter Name	Type	Description
smoothSwitchBitrate	BOOL	Set whether to smoothly switch between multiple bitrates for HLS.

progressInterval

Set the interval for progress callbacks, in milliseconds. The default interval is 500 milliseconds.

```
@property(nonatomic, assign) NSTimeInterval progressInterval;
```

Parameter Description

Parameter Name	Type	Description
progressInterval	NSTimeInterval	Interval time, in milliseconds.

maxBufferSize

Maximum cache size, unit MB. This setting impacts playableDuration. The larger the setting, the more content is cached in advance.

```
@property(nonatomic, assign) float maxBufferSize;
```

Parameter Description

Parameter Name	Type	Description
maxBufferSize	float	Playback buffer size.

maxPreloadSize

Set the maximum preload buffer size, in MB.

```
@property(nonatomic, assign) float maxPreloadSize;
```

Parameter Description

Parameter Name	Type	Description
maxPreloadSize	float	Preload size.

overlayKey

Set encryption key.

```
@property(nonatomic, copy) NSString *overlayKey;
```

overlayIv

Set Encryption Iv.

```
@property(nonatomic, copy) NSString *overlayIv;
```

encryptedMp4Level

Set MP4 encrypted playback.

```
@property(nonatomic, assign) TX_Enum_MP4EncryptionLevel encryptedMp4Level;
```

Parameter Description

Parameter Name	Type	Description
encryptedMp4Level		Set MP4 playback and storage encryption levels,

	TX_Enum_MP4EncryptionLevel	supported starting from Player Premium 12.2. For details, see TX_Enum_MP4EncryptionLevel.

enableRenderProcess

Set the Render display post-processing flag, including features such as super-resolution and VR playback. Using these features requires setting this flag. The default is NO.

```
@property(nonatomic, assign) BOOL enableRenderProcess;
```

Parameter Description

Parameter Name	Type	Description
enableRenderProcess	BOOL	Whether loading post-rendering post-processing services is allowed.

preferredResolution

When there are multiple Hls programs, select the optimal Program for startup based on the configured `preferredResolution`, which is the product of width and height. The valid value for configuration is an integer >= -1, with a default of -1. The playback kernel interprets this as using lower-priority information for configuration and will match the Program with the arithmetic distance closest to a value less than this one.
Priority is bitrateIndex > preferredBitrate > preferredResolution.

```
@property(nonatomic, assign) long preferredResolution;
```

Parameter Description

Parameter Name	Type	Description
preferredResolution	long	Product of video width and height (width * height).

mediaType

Set the media asset type. If adaptive bit rate playback is used, a specific type must be specified. For example, when using adaptive playback for HLS live streaming resources, the MEDIA_TYPE_HLS_LIVE type must be imported.

```
@property(nonatomic, assign) TX_Enum_MediaType mediaType;
```

Parameter Description

Parameter Name	Type	Description
mediaType	TX_Enum_MediaType	Media asset type. For details, see TX_Enum_MediaType.

extInfoMap

Set the player's special configuration.

```
@property(nonatomic, strong) NSDictionary *extInfoMap;
```

Parameter Description

Parameter Name	Type	Description
extInfoMap	NSDictionary	Expansion parameters

preferAudioTrack

Set the preferred audio track name to load at startup. Only the player premium supports this.

```
@property(nonatomic, copy) NSString *preferAudioTrack;
```

Parameter Description

Parameter Name	Type	Description
preferAudioTrack	NSString	audio track name

Media Download TXVodPreloadManagerDelegate

Last updated：2025-05-30 14:54:52

TXVodPreloadManagerDelegate API Introduction

Pre-download URL status callback listener API.

Callback API Overview

API	Description
onStart:fileId:url:param:	Video preloading started.
onComplete:url:	Video preloading completed.
onError:url:error:	Video preloading error.

Callback API Detail

onStart:fileId:url:param:

Preloading completed with successful callback.

```
- (void)onStart:(int)taskID fileId:(NSString *)fileId url:(NSString *)url param:(NS
```

Parameter Description

Parameter Name	Type	Description
taskID	int	Pre-download task ID.
fileId	NSString	fileId for video download. This parameter is nil when using URL caching method.
url	NSString	Pre-download task URL.
param	NSDictionary	Additional parameter.

onComplete:url:

Preloading completed with successful callback.

```
- (void)onComplete:(int)taskID url:(NSString *)url;
```

Parameter Description

Parameter Name	Type	Description
taskID	int	Pre-download task ID.
url	NSString	Pre-download task URL.

onError:url:error:

Pre-download failure callback.

```
- (void)onError:(int)taskID url:(NSString *)url error:(NSError *)error;
```

Parameter Description

Parameter Name	Type	Description
taskID	int	Pre-download task ID.
url	NSString	Pre-download task URL.
error	NSError	Error message.

TXVodDownloadDataSource

Last updated：2025-05-30 14:55:04

TXVodDownloadDataSource Overview

Download a video-on-demand resource object.

Clarity TXVodQuality

Parameter Name	Type	Value	Description
TXVodQualityOD	NSInteger	0	Original video quality.
TXVodQualityFLU	NSInteger	1	Smooth.
TXVodQualitySD	NSInteger	2	Standard definition.
TXVodQualityHD	NSInteger	3	High-definition.
TXVodQualityFHD	NSInteger	4	Full HD.
TXVodQuality2K	NSInteger	5	2K.
TXVodQuality4K	NSInteger	6	4K.
TXVodQuality240P	NSInteger	240	Smooth 240P.
TXVodQuality360P	NSInteger	360	Smooth 360P.
TXVodQuality480P	NSInteger	480	Standard Definition 480P.
TXVodQuality540P	NSInteger	540	Standard Definition 540P.
TXVodQuality720P	NSInteger	720	HD 720P.
TXVodQuality1080p	NSInteger	1080	Full HD 1080P.

API Detail

auth

on-demand fileID authentication information.

```
@property(nonatomic, strong) TXPlayerAuthParams *auth;
```

Parameter Description

Parameter Name	Type	Description
auth	TXPlayerAuthParams	fileid information. For details, refer to TXPlayerAuthParams.

quality

Download clarity. Defaults to high-definition (when retrieving download information, this parameter must be consistent with the parameters used when downloading video).

```
@property(nonatomic, assign) TXVodQuality quality;
```

Parameter Description

Parameter Name	Type	Description
quality	TXVodQuality	Video quality ID. Refer to TXVodQuality enumeration.

token

Set this value, and the player will automatically add voddrm.token.<Token> before the file name in the URL.

```
@property(nonatomic, copy) NSString *token;
```

templateName

Clarity template. If the backend transcoding uses a custom template, enter the template name here. When templateName and quality are set simultaneously, templateName takes precedence.

```
@property(nonatomic, copy) NSString *templateName;
```

fileId

Video file ID.

```
@property(nonatomic, copy) NSString *fileId;
```

pSign

Signature information.

```
@property(nonatomic, copy) NSString *pSign;
```

appId

Application ID.

```
@property(nonatomic, assign) int appId;
```

userName

Account name, default "default".

```
@property(nonatomic, copy) NSString *userName;
```

overlayKey

HLS EXT-X-KEY encryption key.

```
@property(nonatomic, copy) NSString *overlayKey;
```

overlayIv

HLS EXT-X-KEY encryption Iv.

```
@property(nonatomic, copy) NSString *overlayKey;
```

TXVodDownloadMediaInfo

Last updated：2025-05-30 14:55:21

Downloading TXVod Media Information Overview

Download on-demand media asset description.

Status Enumeration

TXVodDownloadMediaInfoState

Parameter Name	Type	Value	Description
TXVodDownloadMediaInfoStateInit	NSInteger	0	Download initial state.
TXVodDownloadMediaInfoStateStart	NSInteger	1	Download started.
TXVodDownloadMediaInfoStateStop	NSInteger	2	Download stopped.
TXVodDownloadMediaInfoStateError	NSInteger	3	Download error.
TXVodDownloadMediaInfoStateFinish	NSInteger	4	Download completed.

API Overview

API	Description
dataSource	Retrieve the source media asset information passed in when downloading with Tencent Cloud Video fileId.
url	Retrieve the actual download address.
userName	Retrieve the download account name.
duration	Get the total duration of the video, in milliseconds.
playableDuration	Retrieve the playable duration of the downloaded content, in milliseconds.
size	Retrieve the total download file size, in Bytes, valid only for the fileId download source.

	Note: The total size refers to the size of the original file uploaded to the Tencent Cloud VOD console. The substream sizes after adaptive dynamic streaming cannot be accessed temporarily.
downloadSize	Retrieve the downloaded file size, in Bytes, valid only for the fileId download source.
segments	Total video segments
downloadSegments	Number of downloaded segments
progress	Retrieve the current download progress.
playPath	Retrieve the playback path of the currently downloaded resource, which can be transmitted to TXVodPlayer for playback.
speed	Retrieve the download speed, in KByte per second. (Supported starting from version 10.9.)
downloadState	Retrieve the download status.
preferredResolution	Retrieve the preferred download resolution.
isResourceBroken	Judge whether the downloaded video resource is damaged. Return true if it is deleted after download or in similar cases. (Supported starting from version 11.0.)
isDownloadFinished	Determine whether the download is complete.

API Detail

dataSource

Retrieve the source media asset information passed in when downloading with Tencent Cloud Video fileId.

```
@property(nonatomic, strong) TXVodDownloadDataSource *dataSource;
```

Return Value

Download resource object information:TXVodDownloadDataSource.

url

Retrieve the actual download address.

```
@property(nonatomic, copy) NSString *url;
```

userName

Retrieve the download account name. Default is default.

```
@property(nonatomic, copy) NSString *userName;
```

duration

Get the total duration of the video, in milliseconds.

```
@property(nonatomic, assign) int duration;
```

playableDuration

Retrieve the playable duration of the downloaded content, in milliseconds.

```
@property(nonatomic, assign) int playableDuration;
```

size

Retrieve the total download file size, in Bytes, valid only for the Tencent Cloud Video fileId download source.
Note: The total size refers to the size of the original file uploaded to the Tencent Cloud VOD console. The substream sizes after adaptive dynamic streaming cannot be accessed temporarily.

```
@property(nonatomic, assign) long size;
```

downloadSize

Retrieve the downloaded file size, in Bytes, valid only for the Tencent Cloud Video fileId download source.

```
@property(nonatomic, assign) long downloadSize;
```

segments

Total video segments

```
@property(nonatomic, assign) int segments;
```

downloadSegments

Total video segments

```
@property(nonatomic, assign) int downloadSegments;
```

progress

Retrieve the current download progress.

```
@property(nonatomic, assign) float progress;
```

playPath

Retrieve the playback path of the currently downloaded resource, which can be transmitted to TXVodPlayer for playback.

```
@property(nonatomic, copy) NSString *playPath;
```

speed

Retrieve the download speed, in KByte per second. (Supported starting from version 10.9.)

```
@property(nonatomic, assign) int speed;
```

downloadState

Retrieve the download status.

```
@property(nonatomic, assign) TXVodDownloadMediaInfoState downloadState;
```

Return Value

Download status. For details, see TXVodDownloadMediaInfoState.

preferredResolution

Retrieve the preferred download resolution.

```
@property(nonatomic, assign) long preferredResolution;
```

isResourceBroken

Judge whether the downloaded video resource is damaged. Return true if it is deleted after download or in similar cases. (Supported starting from version 11.0.)

```
@property(nonatomic, assign) BOOL isResourceBroken;
```

isDownloadFinished

Determine whether the download is complete.

```
- (BOOL)isDownloadFinished;
```

TXVodDownloadDelegate

Last updated：2025-05-30 14:55:35

TXVodDownloadDelegate API Introduction

on-demand player download callback listener API

Callback API Overview

API	Description
onDownloadStart:	Download started.
onDownloadProgress:	Download progress update.
onDownloadStop:	Download stopped.
onDownloadFinish:	Download completed.
onDownloadError:errorCode:errorMsg:	Error encountered during download.

Callback API Details

onDownloadStart:

Download started.

```
- (void)onDownloadStart:(TXVodDownloadMediaInfo *)mediaInfo;
```

Parameter Description

Parameter Name	Type	Description
mediaInfo	TXVodDownloadMediaInfo	Video download information.

onDownloadProgress:

Download progress update.

```
- (void)onDownloadProgress:(TXVodDownloadMediaInfo *)mediaInfo;
```

Parameter Description

Parameter Name	Type	Description
mediaInfo	TXVodDownloadMediaInfo	Video download information.

onDownloadStop:

Download stopped. Calling the `-[TXVodDownloadManager stopDownload:]` method will trigger this callback.

```
- (void)onDownloadStop:(TXVodDownloadMediaInfo *)mediaInfo;
```

Parameter Description

Parameter Name	Type	Description
mediaInfo	TXVodDownloadMediaInfo	Video download information.

onDownloadFinish:

Download completed.

```
- (void)onDownloadFinish:(TXVodDownloadMediaInfo *)mediaInfo;
```

Parameter Description

Parameter Name	Type	Description
mediaInfo	TXVodDownloadMediaInfo	Video download information.

onDownloadError:errorCode:errorMsg:

Error encountered during download.

```
- (void)onDownloadError:(TXVodDownloadMediaInfo *)mediaInfo errorCode:(TXDownloadEr
```

Parameter Description

Parameter Name	Type	Description
mediaInfo	TXVodDownloadMediaInfo	Video download information.
code	TXDownloadError	Download error code.
msg	NSString	Download error message.

TXVodDownloadManager

Last updated：2025-05-30 14:55:49

TXDownloadError

Download Error Code.

Value	Parameter Name	Description
0	TXDownloadSuccess	Download success.
-5001	TXDownloadAuthFaild	File ID authentication failure.
-5003	TXDownloadNoFile	No file with this resolution.
-5004	TXDownloadFormatError	Unsupported format.
-5005	TXDownloadDisconnet	Network disconnection.
-5006	TXDownloadHlsKeyError	Failed to retrieve the HLS decryption key.
-5007	TXDownloadPathError	Download directory access failure.
-5008	TXDownload403Forbidden	Authentication information not approved, such as signature expiration or invalid request.

TXVodDownloadManager API Introduction

on-demand video player download API class
Video download supports downloading MP4 and HLS videos, corresponds to nested HLS videos, requires specifying preferred resolution (preferredResolution)

API Overview

API	Description
shareInstance	Retrieve the TXVodDownloadManager instance object in singleton mode.
setDownloadPath:	Set the root directory for downloading files. If it does not

	exist, it will be automatically created.
startDownload:url:	Start the download using the URL method.
startDownload:	Start the download using the fileId method.
startDownloadUrl:resolution:userName:	Start the download using the URL method, and specify the preferred clarity and account name.
startDownloadDrm:resolution:userName:	Start the download using the Drm method.
stopDownload:	Stop download, stopped successfully when [TXVodDownloadDelegate onDownloadStop:] callback is triggered.
deleteDownloadMediaInfo:	Delete download information.
deleteDownloadFile:	Delete files generated during download.
getDownloadMediaInfoList	Retrieve download list information for all users, time-consuming API, do not call on the main thread.
getDownloadMediaInfo:fileId:qualityId:userName:	Retrieve download information.
getDownloadMediaInfo:resolution:userName:	Retrieve download information.
encryptHexStringHls:	Encrypt.
headers	Set the HTTP request header for downloading.
delegate	Set the download callback proxy object. It must be configured before downloading.
supportPrivateEncryptMode	Set whether private encryption mode is supported (set to NO for system player, YES for custom player). By default, it is set to YES.

API Detail

shareInstance

Retrieve the TXVodDownloadManager instance object in singleton mode.

```
+ (TXVodDownloadManager *)shareInstance;
```

setDownloadPath:

Set the root directory for downloading files. The download directory set here takes precedence over the one set via `[TXPlayerGlobalSetting setCacheFolderPath:]`.

```
- (void)setDownloadPath:(NSString *)path;
```

startDownload:

Start the download using the Tencent Cloud Video fileId method.

```
- (TXVodDownloadMediaInfo *)startDownload:(TXVodDownloadDataSource *)source;
```

Parameter Description

Parameter Name	Type	Description
source	TXVodDownloadDataSource	Download a resource object.

startDownload:url:

Start the download using the URL method.

```
- (TXVodDownloadMediaInfo *)startDownload:(NSString *)username url:(NSString *)url;
```

Parameter Description

Parameter Name	Type	Description
url	NSString	Download address, required.
username	NSString	Account name, optional parameter, defaults to "default" if not provided.

Return Value

Video information and download status, see TXVodDownloadMediaInfo for details.

startDownloadUrl:resolution:userName:

Start the download using the URL method.

```
- (TXVodDownloadMediaInfo *)startDownloadUrl:(NSString *)url resolution:(long)resol
```

Parameter Description

Parameter Name	Type	Description
url	NSString	Download address, required.
resolution	long	Preference for clarity, multiple resolutions url is a required

		parameter, with the value being preference clarity width × height (e.g., 720p input 921600 = 1280 × 720), single clarity input -1.
username	NSString	Account name, optional parameter, defaults to "default" if not provided.

Return Value

Video information and download status, see TXVodDownloadMediaInfo for details.

startDownloadDrm:resolution:userName:

Download a DRM video.

```
- (TXVodDownloadMediaInfo *)startDownloadDrm:(TXPlayerDrmBuilder *)drmBuilder resol
```

Parameter Description

Parameter Name	Type	Description
drmBuilder	TXPlayerDrmBuilder	Download DRM object, refer to TXPlayerDrmBuilder.
resolution	long	Preference clarity, multiple resolutions URL is a required parameter, with the value being preference clarity width * height (e.g., 720p input 921600 = 1280 * 720), single clarity input -1.
username	NSString	Account name, optional parameter, defaults to "default" if not provided.

Return Value

Video information and download status, see TXVodDownloadMediaInfo for details.

stopDownload:

Stop download, stopped successfully when `-[id<TXVodDownloadDelegate> onDownloadStop:]` callback is triggered.

```
- (void)stopDownload:(TXVodDownloadMediaInfo *)media;
```

Parameter Description

Parameter Name	Type	Description
downloadMediaInfo	TXVodDownloadMediaInfo	Download video information.

deleteDownloadMediaInfo:

Delete download information.

```
- (BOOL)deleteDownloadMediaInfo:(TXVodDownloadMediaInfo *)downloadMediaInfo;
```

Parameter Description

Parameter Name	Type	Description
downloadMediaInfo	TXVodDownloadMediaInfo	Download video information.

Return Value

Whether the deletion is successful. YES: Successfully deleted; NO: Deletion failed. Files being downloaded cannot be deleted.

getDownloadMediaInfoList

Retrieve download list information for all users.

```
- (NSArray<TXVodDownloadMediaInfo *> *)getDownloadMediaInfoList;
```

Return Value

Video download information list: NSArray<TXVodDownloadMediaInfo>.

getDownloadMediaInfo:fileId:qualityId:userName:

Retrieve download information.

Call this API to ensure that a download task has been created earlier via `startDownload:`, `startDownload:url:`, or `startDownloadDrm:resolution:userName:`.

```
- (TXVodDownloadMediaInfo *)getDownloadMediaInfo:(int)appId fileId:(NSString *)file
```

Parameter Description

Parameter Name	Type	Description
appId	int	Tencent Cloud Video on Demand (VOD) application appId.
fileId	NSString	Tencent Cloud Video on Demand (VOD) video fileId.
qualityId	int	Video quality Id, refer to the constant TXVodQuality.
userName	NSString	Account name must be consistent with the account name passed in during download. If not passed in during download, pass in the empty character "".

Return Value

Video download information TXVodDownloadMediaInfo.

getDownloadMediaInfo:resolution:userName:

Retrieve download information.

```
- (TXVodDownloadMediaInfo *)getDownloadMediaInfo:(NSString *)url resolution:(long)p
```

Parameter Description

Parameter Name	Type	Description
url	NSString	Download address, required parameter, otherwise download fails.
preferredResolution	long	Download preferred clarity, multiple resolutions URL is a required parameter, with the value being preferred clarity width × height (e.g., 720p input 921600 = 1280 × 720), single clarity input -1.
userName	userName	Account name, optional parameter, defaults to "default" if not provided.

Return Value

Video download information TXVodDownloadMediaInfo.

encryptHexStringHls:

Encrypt.

```
+ (NSString *)encryptHexStringHls:(NSString *)originHexStr
```

Parameter Name	Type	Description
originHexStr	NSString	Initial hexadecimal string.

Return Value

Encrypted string: NSString

headers

Set the http request header.

```
@property(nonatomic, strong) NSDictionary *headers;
```

Parameter Name	Type	Description
headers	NSDictionary	http request header.

delegate

Set the download task callback object.

```
@property(nonatomic, weak) id<TXVodDownloadDelegate> delegate;
```

Parameter Name	Type	Description
delegate	id<TXVodDownloadDelegate>	Download task callback object.

supportPrivateEncryptMode

Set whether private encryption mode is supported (set to NO for system player, YES for custom player). Default is YES.

```
@property(nonatomic, assign) BOOL supportPrivateEncryptMode;
```

Parameter Name	Type	Description
supportPrivateEncryptMode	BOOL	Whether private encryption mode is supported.

TXVodPreloadManagerDelegate

Last updated：2025-05-30 14:56:18

TXVodPreloadManagerDelegate API Introduction

Tencent Cloud Video fileId and URL preload status callback listening API.

Callback API Overview

API	Description
onStart:fileId:url:param:	Video preloading started. For fileId preloading, callback before starting preloading upon link replacement success.
onComplete:url:	Video preloading completed.
onError:url:error:	Video preloading error.

Callback API Detail

onStart:fileId:url:param:

Preloading completed with successful callback.

```
- (void)onStart:(int)taskID fileId:(NSString *)fileId url:(NSString *)url param:(NS
```

Parameter Description

Parameter Name	Type	Description
taskID	int	Pre-download task ID.
fileId	NSString	Pre-downloaded video fileId.
url	NSString	Pre-download task URL, which is the video URL after link replacement, can be used for subsequent playback.
param	NSDictionary	Additional information carried by pre-download.

onComplete:url:

Preloading completed with successful callback.

```
- (void)onComplete:(int)taskID url:(NSString *)url;
```

Parameter Description

Parameter Name	Type	Description
taskID	int	Pre-download task ID.
url	String	Pre-download task URL.

onError:url:error:

Pre-download failure callback.

```
- (void)onError:(int)taskID url:(NSString *)url error:(NSError *)error;
```

Parameter Description

Parameter Name	Type	Description
taskID	int	Pre-download task ID.
url	NSString	Pre-download task URL.
error	NSError	Error message.

TXVodPreloadManager

Last updated：2025-05-30 14:56:32

TXVodPreloadManager API Introduction

VOD player pre-download API class
No need to create a player instance. Pre-download partial video content to accelerate startup speed and provide better playback experience when using the player.

API Overview

API	Description
sharedManager	Retrieve the TXVodPreloadManager instance object in singleton mode.
startPreload:preloadSize:preferredResolution:delegate:	Start preloading through URL.
startPreloadWithModel:preloadSize:preferredResolution:delegate:	Start preloading through fileId or URL. Preferentially use this API.
stopPreload:	Stop preloading.

API Detail

sharedManager

Retrieve the TXVodPreloadManager instance object in singleton mode.

```
+ (instancetype)sharedManager;
```

startPreload:preloadSize:preferredResolution:delegate:

Start preloading through URL.
Before starting preloading, set the playback engine's cache directory `+[TXPlayerGlobalSetting setCacheFolderPath]` and cache size.
`+[TXPlayerGlobalSetting setMaxCacheSize:]`. This setting is a global configuration and must be consistent with the player; otherwise, it can lead to playback cache failure.

```
- (int)startPreload:(NSString *)requestURL preloadSize:(float)preloadSizeMB preferr
```

Parameter Description

Parameter Name	Type	Description
requestURL	NSString	Preloaded video URL.
preloadSizeMB	float	Preloaded size, unit: MB.
preferredResolution	long	Expected download resolution, the product of video width and height (width * height). Pass -1 when multiple resolutions are not supported or no specification is required.
delegate	id<TXVodPreloadManagerDelegate>	Preloading listening status callback object.

Return Value

task ID, use this task ID to stop preloading `-[TXVodPreloadManager stopPreload:]` .

If the return value is -1, it means this task ID is invalid.

startPreloadWithModel:preloadSize:preferredResolution:delegate:

Start preloading, support through Tencent Cloud fileId and video URL.

If `-[TXPlayerAuthParams url]` is not empty, prioritize starting video URL preloading, and at this point, support calling in the mainline.

If `-[TXPlayerAuthParams fileId]` is not empty, start video fileId preloading. At this point, calling in the mainline is not supported.

Note:

1. Preloading is a time-consuming operation. Do not call it in the main thread; calling it in the main thread will throw an illegal call exception.

2. Before starting preloading, set the playback engine's cache directory `+[TXPlayerGlobalSetting setCacheFolderPath:]` and cache size `+[TXPlayerGlobalSetting setMaxCacheSize:]` . This setting is a global configuration and must be consistent with the player; otherwise, it can lead to playback cache failure.

```
- (int)startPreloadWithModel:(TXPlayerAuthParams *)params preloadSize:(float)preloa
```

Parameter Description

Parameter Name	Type	Description
params	TXPlayerAuthParams	Download information.

		Set the http request header for preloading via `-[TXPlayerAuthParams headers]`, and set the audio track name for preloading via `-[TXPlayerAuthParams preferAudioTrack]`.
preloadSizeMB	float	Preloaded size, unit: MB.
preferredResolution	long	Expected download resolution, the product of video width and height (width * height). Pass -1 when multiple resolutions are not supported or no specification is required.
delegate	id<TXVodPreloadManagerDelegate>	Preloading listening status callback object.

Return Value

task ID, use this task ID to stop preloading `-[TXVodPreloadManager stopPreload:]`.

If the return value is -1, it means this task ID is invalid.

stopPreload:

Stop preloading.

```
- (void)stopPreload:(int)taskID;
```

Parameter Description

Parameter Name	Type	Description
taskID	int	task ID. The ID is obtained from the API return value of the startPreload interface.

Type Definition
TXVodSDKEventDef

Last updated：2025-05-30 14:56:59

TXVodSDKEventDef API Introduction

Constant class used in the on-demand player.

Video Resolution

TX_Enum_Type_VideoResolution

Value	Parameter Name	Description
0	VIDEO_RESOLUTION_TYPE_360_640	Recommended bitrate 800kbps.
1	VIDEO_RESOLUTION_TYPE_540_960	Suggested bitrate 1200kbps.
2	VIDEO_RESOLUTION_TYPE_720_1280	Suggested bitrate 1800kbps.
30	VIDEO_RESOLUTION_TYPE_1080_1920	Suggested bitrate 3000kbps.

Picture Quality Tier

TX_Enum_Type_VideoQuality

Value	Parameter Name	Description
1	VIDEO_QUALITY_STANDARD_DEFINITION	Standard Definition: Use a resolution of 360 × 640.
2	VIDEO_QUALITY_HIGH_DEFINITION	High Definition: Use a resolution of 540 × 960.
3	VIDEO_QUALITY_SUPER_DEFINITION	Ultra HD: Use a resolution of 720 × 1280.
4	VIDEO_QUALITY_LINKMIC_MAIN_PUBLISHER	Microphone-connected scenario, used by the main host.
5	VIDEO_QUALITY_LINKMIC_SUB_PUBLISHER	Microphone-connected scenario, used by the co-host (audience in the connection).

7	VIDEO_QUALITY_ULTRA_DEFINITION	Blu-ray: Use a resolution of 1080 × 1920.

Image Rotation Direction

TX_Enum_Type_HomeOrientation

Value	Parameter Name	Description
0	HOME_ORIENTATION_RIGHT	HOME key is on the right, landscape mode.
1	HOME_ORIENTATION_DOWN	HOME key is at the bottom, the most common vertical screen live streaming mode in mobile live streaming.
2	HOME_ORIENTATION_LEFT	HOME key is on the left, landscape mode.
3	HOME_ORIENTATION_UP	HOME key is at the top, vertical screen live streaming.

Video Fill Mode

TX_Enum_Type_RenderMode

Value	Parameter Name	Description
0	RENDER_MODE_FILL_SCREEN	Video fills the screen.
1	RENDER_MODE_FILL_EDGE	Video adapts to the screen.

Playing Event List

Value	Parameter Name	Description
2002	VOD_PLAY_EVT_HIT_CACHE	Start playback cache hit.
2003	VOD_PLAY_EVT_RCV_FIRST_I_FRAME	Network receives the first video packet (IDR).
2004	VOD_PLAY_EVT_PLAY_BEGIN	Playback start.
2005	VOD_PLAY_EVT_PLAY_PROGRESS	Video playback progress.
2006	VOD_PLAY_EVT_PLAY_END	Playback end.

6001	VOD_PLAY_EVT_LOOP_ONCE_COMPLETE	Loop one round, playback ends.
2007	VOD_PLAY_EVT_PLAY_LOADING	Video playback Loading.
2008	VOD_PLAY_EVT_START_VIDEO_DECODER	Decoder starts up.
2009	VOD_PLAY_EVT_CHANGE_RESOLUTION	Video resolution changed.
2010	VOD_PLAY_EVT_GET_PLAYINFO_SUCC	Successfully retrieve on-demand file information.
2011	VOD_PLAY_EVT_CHANGE_ROTATION	Video rotation information.
2013	VOD_PLAY_EVT_VOD_PLAY_PREPARED	Video loading complete.
2014	VOD_PLAY_EVT_VOD_LOADING_END	Video buffer complete.
2017	VOD_PLAY_EVT_VOD_PLAY_FIRST_VIDEO_PACKET	Received the first frame of data (supported starting from version 12.0).
2019	VOD_PLAY_EVT_VOD_PLAY_SEEK_COMPLETE	Seek completed (supported starting from version 10.3).
2020	VOD_PLAY_EVT_SELECT_TRACK_COMPLETE	Track switch completed.
2026	VOD_PLAY_EVT_RCV_FIRST_AUDIO_FRAME	First audio playback.
2103	PLAY_WARNING_RECONNECT	Network disconnection, auto reconnection started.
2030	VOD_PLAY_EVT_VIDEO_SEI	Video sei information event.
2031	VOD_PLAY_EVT_HEVC_DOWNGRADE_PLAYBACK	HEVC downgrade playback
-2301	VOD_PLAY_ERR_NET_DISCONNECT	Network disconnection, multiple reconnection attempts failed.
-2303	VOD_PLAY_ERR_FILE_NOT_FOUND	File does not exist.
-2304	PLAY_ERR_HEVC_DECODE_FAIL	HEVC decoding failed.
-2305	VOD_PLAY_ERR_HLS_KEY	HLS decryption key acquisition failed.
-2306	VOD_PLAY_ERR_GET_PLAYINFO_FAIL	Failed to retrieve on-demand file information.
2106	PLAY_WARNING_HW_ACCELERATION_FAIL	Hardware decoding startup failed, use software decoding.

-5	VOD_PLAY_ERR_LICENCE_CHECK_FAIL	Invalid License, playback failure. Before calling startVodPlay, you must configure license using TXLiveBase#setLicence to achieve successful playback; otherwise, playback will fail (black screen). The license only needs to be set globally once. Live stream publishing license, Short Video License, and player license can all be used. If you have not obtained the above licenses, you can click Player License to apply. An official license requires purchase.
-6004	VOD_PLAY_ERR_SYSTEM_PLAY_FAIL	System player playback error.
-6005	VOD_PLAY_ERR_DEMUXER_TIMEOUT	Decapsulation timeout.
-6006	VOD_PLAY_ERR_DECODE_VIDEO_FAIL	Video decoding error. Video format not supported.
-6007	VOD_PLAY_ERR_DECODE_AUDIO_FAIL	Audio decoding error. Audio format not supported.
-6008	VOD_PLAY_ERR_DECODE_SUBTITLE_FAIL	Subtitle decoding error.
-6009	VOD_PLAY_ERR_RENDER_FAIL	Rendering error in video.
-6010	VOD_PLAY_ERR_PROCESS_VIDEO_FAIL	Video post-processing error.
-6011	VOD_PLAY_ERR_DOWNLOAD_FAIL	Download error in video.
-6101	VOD_PLAY_ERR_DRM	DRM playback failure

Picture-In-Picture Controller Status

Value	Parameter Name	Description
0	TX_VOD_PLAYER_PIP_STATE_UNDEFINED	Invalid state.
1	TX_VOD_PLAYER_PIP_STATE_WILL_START	Picture-in-Picture will start soon.
2	TX_VOD_PLAYER_PIP_STATE_DID_START	Picture-in-Picture has already started.
3	TX_VOD_PLAYER_PIP_STATE_WILL_STOP	Picture-in-Picture will end soon.

4	TX_VOD_PLAYER_PIP_STATE_RESTORE_UI	Reset the UI.

Picture-In-Picture Error Type

Value	Parameter Name	Description
0	TX_VOD_PLAYER_PIP_ERROR_TYPE_NONE	No error.
1	TX_VOD_PLAYER_PIP_ERROR_TYPE_DEVICE_NOT_SUPPORT	Device or system version is not supported (PIP is only supported on iPad iOS9+).
2	TX_VOD_PLAYER_PIP_ERROR_TYPE_PLAYER_NOT_SUPPORT	Player is not supported.
3	TX_VOD_PLAYER_PIP_ERROR_TYPE_VIDEO_NOT_SUPPORT	Video is not supported.
4	TX_VOD_PLAYER_PIP_ERROR_TYPE_PIP_IS_NOT_POSSIBLE	PIP controller is unavailable.
5	TX_VOD_PLAYER_PIP_ERROR_TYPE_ERROR_FROM_SYSTEM	PIP controller reports an error.
10	TX_VOD_PLAYER_PIP_ERROR_TYPE_PLAYER_NOT_EXIST	Player object does not exist.
11	TX_VOD_PLAYER_PIP_ERROR_TYPE_PIP_IS_RUNNING	PIP feature is running.
12	TX_VOD_PLAYER_PIP_ERROR_TYPE_PIP_NOT_RUNNING	PIP feature is not started.
13	TX_VOD_PLAYER_PIP_ERROR_TYPE_PIP_START_TIMEOUT	PIP startup timed out.
20	TX_VOD_PLAYER_PIP_ERROR_TYPE_SEAMLESS_PIP_ERROR	Seamless PIP feature failed to start.
21	TX_VOD_PLAYER_PIP_ERROR_TYPE_SEAMLESS_PIP_NOT_SUPPORT	Seamless switching of PIP is not supported.

22	TX_VOD_PLAYER_PIP_ERROR_TYPE_SEAMLESS_PIP_IS_RUNNING	Seamless PIP feature is running.

AirPlay Status (System Player Only)

Value	Parameter Name	Description
0	TX_VOD_PLAYER_AIRPLAY_STATE_NOT_RUNNING	Not running.
1	TX_VOD_PLAYER_AIRPLAY_STATE_DID_RUNNING	Running.

AirPlay Error Type (System Player Only)

Value	Parameter Name	Description
0	TX_VOD_PLAYER_AIRPLAY_ERROR_TYPE_NONE	No error.
1	TX_VOD_PLAYER_AIRPLAY_ERROR_TYPE_PLAYER_NOT_SUPPORT	Player is not supported.
2	TX_VOD_PLAYER_AIRPLAY_ERROR_TYPE_VIDEO_NOT_SUPPORT	Video is not supported.
10	TX_VOD_PLAYER_AIRPLAY_ERROR_TYPE_PLAYER_INVALID	Player object is unavailable.
11	TX_VOD_PLAYER_AIRPLAY_ERROR_TYPE_PLAYER_STATE	Player status error.

External Subtitle Type

TX_VOD_PLAYER_SUBTITLE_MIME_TYPE

Value	Parameter Name	Description
0	TX_VOD_PLAYER_MIMETYPE_TEXT_SRT	External subtitle in SRT format.
1	TX_VOD_PLAYER_MIMETYPE_TEXT_VTT	External subtitle in VTT format.

Playing Event Parameters

Value	Parameter Name
"CPU_USAGE"	NET_STATUS_CPU_USAGE
"VIDEO_WIDTH"	NET_STATUS_VIDEO_WIDTH
"VIDEO_HEIGHT"	NET_STATUS_VIDEO_HEIGHT
"NET_SPEED"	NET_STATUS_NET_SPEED
"VIDEO_FPS"	NET_STATUS_VIDEO_FPS
"VIDEO_BITRATE"	NET_STATUS_VIDEO_BITRATE
"AUDIO_BITRATE";	NET_STATUS_AUDIO_BITRATE
"VIDEO_CACHE"	NET_STATUS_VIDEO_CACHE
"SERVER_IP"	NET_STATUS_SERVER_IP
"EVT_UTC_TIME"	VOD_PLAY_EVENT_UTC_TIME
"EVT_BLOCK_DURATION"	VOD_PLAY_EVENT_BLOCK_DURATION
"EVT_ERROR_CODE"	VOD_PLAY_EVT_ERROR_CODE
"EVT_TIME"	VOD_PLAY_EVENT_TIME
"EVT_MSG"	VOD_PLAY_EVENT_MSG
"EVT_PARAM1"	VOD_PLAY_EVENT_PARAM1
"EVT_PARAM2"	VOD_PLAY_EVENT_PARAM2
"EVT_GET_MSG"	VOD_PLAY_EVENT_GET_MSG

"EVT_PLAY_COVER_URL"	VOD_PLAY_EVENT_PLAY_COVER_URL
"EVT_PLAY_URL"	VOD_PLAY_EVENT_PLAY_URL
"EVT_PLAY_NAME"	VOD_PLAY_EVENT_PLAY_NAME
"EVT_PLAY_DESCRIPTION"	VOD_PLAY_EVENT_PLAY_DESCRIPTION
"EVT_PLAY_PROGRESS"	VOD_PLAY_EVENT_PLAY_PROGRESS
"EVT_PLAY_DURATION"	VOD_PLAY_EVENT_PLAY_DURATION
"EVT_PLAYABLE_DURATION"	VOD_PLAY_EVENT_PLAYABLE_DURATION
"EVT_IMAGESPRIT_WEBVTTURL"	VOD_PLAY_EVENT_IMAGESPRIT_WEBVTTURL
"EVT_IMAGESPRIT_IMAGEURL_LIST"	VOD_PLAY_EVENT_IMAGESPRIT_IMAGEURL_LIST
"VOD_KEY_VIDEO_CODEC_TYPE"	VOD_KEY_VIDEO_CODEC_TYPE
"VOD_KEY_BACKUP_URL_MEDIA_TYPE"	VOD_KEY_BACKUP_URL_MEDIA_TYPE
"PARAM_MODULE_TYPE"	PLAYER_OPTION_PARAM_MODULE_TYPE
"PARAM_MODULE_CONFIG"	PLAYER_OPTION_PARAM_MODULE_CONFIG
"ENABLE_SENSOR"	PLAYER_OPTION_PARAM_MODULE_VR_ENABLE_SENS(
"FOV"	PLAYER_OPTION_PARAM_MODULE_VR_FOV
"ANGLE_X"	PLAYER_OPTION_PARAM_MODULE_VR_ANGLE_X
"ANGLE_Y"	PLAYER_OPTION_PARAM_MODULE_VR_ANGLE_Y

"ANGLE_RATE"	PLAYER_OPTION_PARAM_MODULE_VR_ANGLE_RATE
"ANGLE_SLOPE_THRESHOLD"	PLAYER_OPTION_PARAM_MODULE_VR_ANGLE_SLOPE_
"EVT_KEY_FRAME_CONTENT_LIST"	VOD_PLAY_EVENT_KEY_FRAME_CONTENT_LIST
"EVT_KEY_FRAME_TIME_LIST"	VOD_PLAY_EVENT_KEY_FRAME_TIME_LIST
"EVT_PLAY_PDT_TIME_MS"	VOD_PLAY_EVENT_PLAY_PDT_TIME_MS
"VOD_KEY_CUSTOM_DATA"	VOD_KEY_CUSTOM_DATA
"EVT_KEY_VIDEO_ROTATION"	VOD_PLAY_EVENT_KEY_VIDEO_ROTATION
"EVT_KEY_SELECT_TRACK_INDEX"	EVT_KEY_SELECT_TRACK_INDEX
"EVT_KEY_SELECT_TRACK_ERROR_CODE"	EVT_KEY_SELECT_TRACK_ERROR_CODE
"VOD_PLAY_BUFFERING_LOADING_TYPE"	VOD_PLAY_BUFFERING_LOADING_TYPE
"MONET_AC_DO_ROTATE"	PLAYER_OPTION_PARAM_MODULE_VR_DO_ROTATE
"EVT_KEY_WATER_MARK_TEXT"	EVT_KEY_WATER_MARK_TEXT
"EVT_KEY_SEI_TYPE"	EVT_KEY_SEI_TYPE
"EVT_KEY_SEI_SIZE"	EVT_KEY_SEI_SIZE

"EVT_KEY_SEI_DATA"	EVT_KEY_SEI_DATA

Player Media Asset Type

Value	Parameter Name	Description
0	MEDIA_TYPE_AUTO	auto type.
1	MEDIA_TYPE_HLS_VOD	adaptive bit rate playback HLS VOD media.
2	MEDIA_TYPE_HLS_LIVE	adaptive bit rate playback HLS live media.
3	MEDIA_TYPE_FILE_VOD	common file VOD media such as MP4.
4	MEDIA_TYPE_DASH_VOD	adaptive bit rate playback DASH VOD media.

MP4 Encryption Level

Value	Parameter Name	Description
0	MP4_ENCRYPTION_LEVEL_NONE	MP4 Unencrypted playback.
1	MP4_ENCRYPTION_LEVEL_L1	L1, MP4 Online encrypted playback.
2	MP4_ENCRYPTION_LEVEL_L2	L2, MP4 Local encrypted playback.

Module Type

TX_VOD_PLAYER_OPTION_PARAM_MODULE_TYPE

Value	Parameter Name	Description
0	PLAYER_OPTION_PARAM_MODULE_TYPE_NONE	Empty type, which means disabling features such as supersampling and VR.
1	PLAYER_OPTION_PARAM_MODULE_TYPE_SR	Supersampling Type
11	PLAYER_OPTION_PARAM_MODULE_TYPE_VR_PANORAMA	VR panorama model, monocular.

12	PLAYER_OPTION_PARAM_MODULE_TYPE_VR_BINOCULAR	VR panorama model, binocular.

Unclassified Variable

Value	Parameter Name	Description
0	PLAYER_SYSTEM_MEDIA_PLAYER	System player.
1	PLAYER_THUMB_PLAYER	Proprietary player, supports software decoding, with better compatibility.
-1	INDEX_AUTO	Adaptive bitrate index flag.
"450"	PLAYER_OPTION_KEY_SUBTITLE_OUTPUT_TYPE	External subtitle output type configuration Key.
"VOD_KEY_BACKUP_URL"	VOD_KEY_BACKUP_URL	Downgrade playback alternate URL Key.
"text/x-subrip"	VOD_PLAY_MIMETYPE_TEXT_SRT	External subtitle in SRT format.
"text/vtt"	VOD_PLAY_MIMETYPE_TEXT_VTT	External subtitle in VTT format.

TXPlayerSubtitleRenderModel

Last updated：2025-05-30 14:57:14

TXPlayerSubtitleRenderModel Overview

Subtitle rendering parameters for on-demand player

Field Details

canvasWidth

canvasWidth and canvasHeight are the dimensions of the subtitle render canvas. The ratio of canvasWidth to canvasHeight must match the aspect ratio of the video; otherwise, the rendered text will be deformed. If not set, the player will use the current video's size as the render canvas size.

```
@property(nonatomic, assign) int canvasWidth;
```

canvasHeight

canvasWidth and canvasHeight define the size of the subtitle render canvas. The ratio between canvasWidth and canvasHeight must align with the video's aspect ratio; otherwise, the rendered text will be distorted. If not specified, the player will use the current video's dimensions as the render canvas size.

```
@property(nonatomic, assign) int canvasHeight;
```

familyName

Font family name. iOS defaults to "Helvetica". A non-empty string is considered set, while an empty string is considered not set.

```
@property(nonatomic, copy) NSString *familyName;
```

fontSize

Font size. If fontSize is set, canvasWidth and canvasHeight must also be set; otherwise, the internal system won't know what size to reference for rendering text. If fontSize is not set, the internal system will use the default font size.

```
@property(nonatomic, assign) float fontSize;
```

fontScale

Font scaling ratio, dedicated for VTT and CSS. Use fontScale to multiply the font-size: em value set in VTT and then adapt to the video width. If fontScale is set, paramFlags must be set to TP_SUBTITLE_PARAM_FLAG_FONT_SCALE. The final font pixel value is calculated as fontScale × VTT em × 16 × canvas width (video width) / default width (491).
fontScale defaults to 1.0. When the video width is 491 pixels and the Chinese font size is set to 16 pixels, the font size in the VTT file is set to 1em (font-size: 1.00em;).

```
@property(nonatomic, assign) float fontScale;
```

fontColor

Text color, ARGB format. If not set, the default is opaque white (0xFFFFFFFF).

```
@property(nonatomic, assign) uint32_t fontColor;
```

outlineWidth

Stroke width. If not set, the internal will use the default stroke width.

```
@property(nonatomic, assign) float outlineWidth;
```

outlineColor

Stroke color, ARGB format. If not set, the default is opaque black (0xFF000000).

```
@property(nonatomic, assign) uint32_t outlineColor;
```

isBondFontStyle

Whether it is bold, the default value is normal font.

```
@property(nonatomic, assign) BOOL isBondFontStyle;
```

lineSpace

Line spacing. If lineSpace is set, canvasWidth and canvasHeight must be set. If not set, the internal will use the default line spacing.

```
@property(nonatomic, assign) float lineSpace;
```

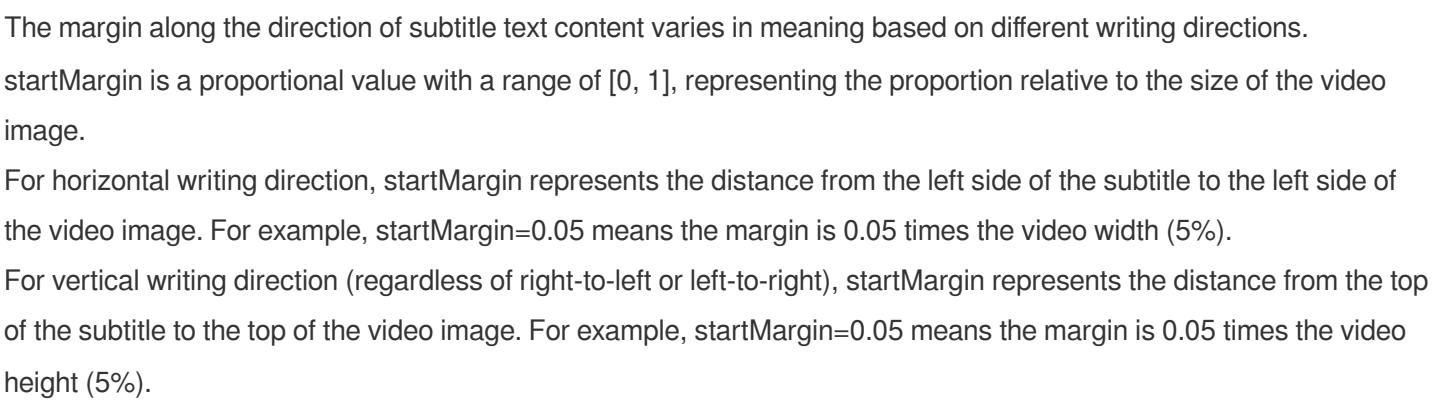
startMargin

startMargin, endMargin, and verticalMargin define the drawing area of subtitles. If not set, the settings in the subtitle file are used. If the subtitle file does not define them either, the default values are used.

Note:

Once set, if the subtitle file defines one or more of the parameters startMargin, endMargin, and yMargin, they will override the corresponding parameters in the subtitle file.

The following diagram illustrates the meaning of these parameters in horizontal writing direction. Use the notes for each parameter to understand them.



The margin along the direction of subtitle text content varies in meaning based on different writing directions. startMargin is a proportional value with a range of [0, 1], representing the proportion relative to the size of the video image.

For horizontal writing direction, startMargin represents the distance from the left side of the subtitle to the left side of the video image. For example, startMargin=0.05 means the margin is 0.05 times the video width (5%).

For vertical writing direction (regardless of right-to-left or left-to-right), startMargin represents the distance from the top of the subtitle to the top of the video image. For example, startMargin=0.05 means the margin is 0.05 times the video height (5%).

```
@property(nonatomic, assign) float startMargin;
```

endMargin

The margin along the direction of subtitle text content varies in meaning based on different writing directions. endMargin is a proportional value with a range of [0, 1], representing the proportion relative to the size of the video image.

For horizontal writing direction, endMargin represents the distance from the right side of the subtitle to the right side of the video image. For example, endMargin=0.05 means the margin is 0.05 times the video width (5%).

For vertical writing direction (regardless of right-to-left or left-to-right), endMargin represents the distance from the bottom of the subtitle to the bottom of the video image. For example, endMargin=0.05 means the margin is 0.05 times the video height (5%).

```
@property(nonatomic, assign) float endMargin;
```

verticalMargin

Margin for vertical subtitle text direction varies based on writing direction. verticalMargin is a proportional value with a range of [0, 1], representing the proportion relative to the video image size.

For horizontal writing direction, yMargin represents the distance from the bottom of the subtitle to the bottom of the video image. For example, yMargin=0.05 means the margin is 0.05 times the video height (5%).

For vertical, right-to-left writing direction, yMargin represents the distance from the right side of the subtitle to the right side of the video image. For example, yMargin=0.05 means the margin is 0.05 times the video width (5%).

For vertical, left-to-right writing direction, yMargin represents the distance from the left side of the subtitle to the left side of the video image. For example, yMargin=0.05 means the margin is 0.05 times the video width (5%).

```
@property(nonatomic, assign) float verticalMargin;
```

TXTrackInfo

Last updated：2025-05-30 14:57:28

TXTrackInfo Overview

Detailed information about the track played by the on-demand player.

TX_VOD_MEDIA_TRACK_TYPE (Track Type)

Value	Parameter Name	Description
0	TX_VOD_MEDIA_TRACK_TYPE_UNKNOW	unknown type.
1	TX_VOD_MEDIA_TRACK_TYPE_VIDEO	video track.
2	TX_VOD_MEDIA_TRACK_TYPE_AUDIO	audio track.
3	TX_VOD_MEDIA_TRACK_TYPE_SUBTITLE	subtitle track.

API Detail

Parameter Name	Type	Description
trackIndex	int	Track index.
trackType	TX_VOD_MEDIA_TRACK_TYPE	track type.
name	NSString	Track name.
isSelected	bool	Whether the current track is selected.
isExclusive	bool	If YES, only one track of this type can be selected at any given moment. If NO, multiple tracks of this type can be selected simultaneously.
isInternal	bool	Whether the current track is an internal primitive track.
getTrackIndex	int	Retrieve track index.

getTrackType	TX_VOD_MEDIA_TRACK_TYPE	Retrieve track type.
getTrackName	NSString	Retrieve track name.
isEqual:	bool	Whether the tracks are the same.

trackType

Retrieve track type.

```
@property(nonatomic, assign) TX_VOD_MEDIA_TRACK_TYPE trackType;
```

trackIndex

Retrieve track index.

```
@property(nonatomic, assign) int trackIndex;
```

name

Retrieve track name.

```
@property(nonatomic, copy) NSString *name;
```

isSelected

Whether the current track is selected.

```
@property(nonatomic, assign) bool isSelected;
```

isExclusive

If YES, only one track of this type can be selected at the same time; if NO, multiple tracks of this type can be selected simultaneously.

```
@property(nonatomic, assign) bool isExclusive;
```

isInternal

Whether the current track is an internal primitive track.

```
@property(nonatomic, assign) bool isInternal;
```

getTrackIndex

Retrieve track index.

```
- (int)getTrackIndex;
```

Return Value

Track index, int.

getTrackType

Retrieve track type.

```
- (TX_VOD_MEDIA_TRACK_TYPE)getTrackType;
```

Return Value

Type	Description
TX_VOD_MEDIA_TRACK_TYPE	Track type.

getTrackName

Retrieve track name.

```
- (NSString *)getTrackName;
```

Return Value

Track name, NSString.

isEqual:

Whether the tracks are the same.

```
- (bool)isEqual:(TXTrackInfo *)trackInfo;
```

Return Value

Whether the tracks are the same, BOOL. YES indicates the same, NO indicates different.

TXVodDef

Last updated：2025-05-30 14:57:41

Common Structure

TXVodSubtitleData

Subtitle text data from VOD player callback.

Parameter Name	Type	Description
subtitleData	NSString	Caption content, empty indicates no caption content.
trackIndex	int64_t	Current subtitle track's trackIndex.
durationMs	int64_t	Subtitle duration in milliseconds. Temporarily empty, do not use.
startPositionMs	int64_t	Subtitle start time in milliseconds. Temporarily empty, do not use.

TXBitrateItem

Last updated：2025-05-30 14:57:54

TXBitrateItem Overview

Video bitrate information.

Field Details

Parameter Name	Type	Description
index	NSInteger	bitrate index, serial number in the m3u8 file
width	NSInteger	width of this video stream
height	NSInteger	height of this video stream.
bitrate	NSInteger	bitrate of this video stream.
bandwidth	int64_t	Bandwidth of this stream.

TXPlayerAuthParams

Last updated：2025-05-30 14:58:08

Introducing the TXPlayerAuthParams API

on-demand player playback media parameters, the Tencent Cloud fileId and playback address can be configured through TXPlayInfoParams.

API Overview

API	Description
appId	Application appId.
fileId	Video file ID.
timeout	Encrypted link timeout timestamp.
exper	Preview duration.
us	Unique identification of request.
sign	Hotlink protection signature.
https	Whether to use HTTPS request.
url	Video file URL.
mediaType	Media asset type.
encryptedMp4Level	MP4 encryption level.
headers	HTTP Header.
preferAudioTrack	Preloaded audio track name.

API Detail

appId

application ID. If the URL is not set, this field is required.

```
@property(nonatomic, assign) int appId;
```

fileId

video file ID. If the URL is not set, this field is required.

```
@property(nonatomic, copy) NSString *fileId;
```

timeout

encrypted link timeout timestamp. Optional, will be converted to a lowercase hexadecimal string, and the CDN server will judge whether the link is valid based on this timestamp.

```
@property(nonatomic, copy) NSString *fileId;
```

exper

Preview duration (in seconds), optional.

```
@property(nonatomic, assign) int exper;
```

us

Unique identification of request, increase link uniqueness.

```
@property(nonatomic, copy) NSString *us;
```

sign

Hotlink protection. If hotlink protection is not used, leave this parameter blank. The hotlink protection parameters (t, us, exper) used by the player API are consistent with the CDN, with the only difference being the calculation method of the sign.
Common hotlink protection signature: sign = md5(KEY + appId + fileId + t + us).
Preview edition hotlink protection signature: sign = md5(KEY + appId + fileId + t + exper + us).

```
@property(nonatomic, copy) NSString *sign;
```

https

Whether to use HTTPS request, defaults to NO.

```
@property(nonatomic, assign) BOOL https;
```

url

video URL.

```
@property(nonatomic, copy) NSString *url;
```

mediaType

Media type. Defaults to MEDIA_TYPE_AUTO.

```
@property(nonatomic, assign) TX_Enum_MediaType mediaType;
```

Parameter Description

Parameter Name	Type	Description
mediaType	TX_Enum_MediaType	Set media asset type. Defaults to auto type.

encryptedMp4Level

Set the encryption level for MP4 playback.

```
@property(nonatomic, assign) TX_Enum_MP4EncryptionLevel encryptedMp4Level;
```

Parameter Description

Parameter Name	Type	Description
encryptedMp4Level	TX_Enum_MP4EncryptionLevel	Set the encryption level for MP4 playback and storage, supported starting from Player Premium version 12.2.

headers

Customize the Http headers carried during online playback by the player.

```
@property(nonatomic, strong) NSDictionary *headers;
```

preferAudioTrack

Set the preferred audio track for playback startup. In pre-download scenarios, configure the preferred download audio track.

Notes:

Note: Player Premium version 12.3 or later is supported.

```
@property(nonatomic, copy) NSString *preferAudioTrack;
```

Parameter Description

Parameter Name	Type	Description
preferAudioTrack	NSString	audio track name.

TXImageSprite

Last updated：2025-05-30 14:58:22

TXImageSprite Overview

On-demand sprite sheet parsing utility class.

API Overview

API	Description
setVTTUrl:imageUrls:	Set the sprite sheet address.
getThumbnail:	Retrieve a thumbnail.

API Detail

setVTTUrl:imageUrls:

Set the sprite sheet address. After configuration, it will start a child thread to download the sprite sheet and parse it.

```
- (void)setVTTUrl:(NSURL *)vttUrl imageUrls:(NSArray<NSURL *> *)images;
```

Parameter Description

Parameter Name	Type	Description
vttUrl	NSURL	sprite sheet web vtt description file download URL.
images	NSArray<NSURL *>	sprite sheet image download URL.

getThumbnail:

Retrieve a thumbnail, return nil if retrieval fails.

```
- (UIImage *)getThumbnail:(GLfloat)time;
```

Parameter Description

Parameter Name	Type	Description

time	GLfloat	Time point, unit: seconds.

Return Value

Thumbnail, UIImage.

TXPlayerDrmBuilder

Last updated：2025-05-30 14:59:08

TXPlayerDrmBuilder Overview

Drm playback information, used in conjunction with `-[TXVodPlayer startPlayDrm:]`.

API Overview

API	Description
initWithDeviceCertificateUrl:licenseUrl:videoUrl:	Construct a Drm playback information object.
deviceCertificateUrl	Set the certificate provider URL.
keyLicenseUrl	Set the decryption Key URL.
playUrl	Set the media playback URL.

API Detail

initWithDeviceCertificateUrl:licenseUrl:videoUrl:

Constructor.

```
- (instancetype)initWithDeviceCertificateUrl:(NSString *)certificateUrl licenseUrl:
```

Parameter Description

Parameter Name	Type	Description
certificateUrl	NSString	Certificate provider URL.
licenseUrl	NSString	Play media URL.
videoUrl	NSString	Set the decryption Key URL.

deviceCertificateUrl

Set the certificate provider URL.

```
@property(nonatomic, strong) NSString *deviceCertificateUrl;
```

Parameter Description

Parameter Name	Type	Description
deviceCertificateUrl	NSString	Certificate provider URL.

keyLicenseUrl

Set the decryption Key URL.

```
@property(nonatomic, strong) NSString *keyLicenseUrl;
```

playUrl

Set the media playback URL.

```
@property(nonatomic, strong) NSString *playUrl;
```

Android TXVodPlayer

Last updated：2025-05-30 15:00:37

Introducing the TXVodPlayer API

TXVodPlayer is a core playback class, primarily responsible for playback, suspension, and playback speed control. For the full capability of the player, please see Player SDK Feature Description.

API Overview

Basic Playback API

API	Description
startVodPlay	Play an HTTP URL-formatted address. Starting from version 10.7, `startPlay` has been changed to `startVodPlay`. You need to call `TXLiveBase#setLicence` to set the License before successful playback; otherwise, playback will fail (resulting in a black screen). The License only needs to be set globally once. Live stream License, short video License, and player License can all be used. If you have not obtained the above License yet, you can click Player License to apply. An official License requires purchase.
startVodPlay	Play in fileId format, pass in TXPlayInfoParams parameter. Starting from version 10.7, startPlay is changed to startVodPlay. You must call {@link TXLiveBase#setLicence} to set the Licence before successful playback; otherwise, playback will fail (black screen). The Licence only needs to be set globally once. Live stream Licence, short video Licence, and player Licence can all be used. If you have not obtained the above Licence yet, you can click Player License to apply. The official license requires purchase.
startPlayDrm	Play DRM-encrypted video.
stopPlay	Stop playback.
isPlaying	Whether it is playing.
pause	Pause playback Stop retrieving streaming data Retain the last frame
resume	Resume playback. Re-fetch streaming data.

seek	Navigate to the specified time point in the video stream, in seconds.
seek	Navigate to the specified time point in the video stream, in seconds, with precision up to 3 decimal places.
getCurrentPlaybackTime	Retrieve the current playback position, in seconds.
getBufferDuration	Retrieve the total duration of the cache, in seconds.
getDuration	Retrieve the total duration, in seconds.
getPlayableDuration	Retrieve the playable duration, in seconds.
getWidth	Get video width.
getHeight	Retrieve video height.
setAutoPlay	Set whether to autoplay after startPlay, default is autoplay.
setStartTime	Set playback start time.
setToken	Encrypt the token for HLS.
getEncryptedPlayKey	Retrieve the encryption key for secure encrypted playback.
setLoop	Set whether to loop playback.
isLoop	Return whether the playback loop status is enabled.
addSubtitleSource	Add external subtitle (only supported in player premium).
getSubtitleTrackInfo	Return subtitle track information list (only supported in player premium).
getAudioTrackInfo	Return audio track information list (only supported in player premium).
selectTrack	Select track (only supported in player premium).
deselectTrack	Cancel track selection (only supported in player premium).
seekToPdtTime	Navigate to a specified PDT (Program Date Time) time point in the video stream, enabling features such as fast-forward, rewind, and progress bar jumping. Currently only supports HLS video format (starting from player premium version 11.6). Parameter unit: milliseconds (ms).

Player Configuration API

API	Description
setConfig	Set player configuration information. For configuration details, please see

	TXVodPlayConfig.
setPlayerView	Set the video rendering TXCloudVideoView for the player.
setPlayerView	Set the video rendering TextureView for the player.
setSurface	Set the video rendering Surface for the player.
setStringOption	Set the player's business parameters, with the parameter format as `<String,Object>` .
setSubtitleStyle	Set subtitle style information. Subtitle styles can be updated after playback (only supported in player premium).
setSubtitleView	Set the subtitle rendering target object View (only supported in player premium).
setAudioNormalization	Set volume equalization and loudness range. Default values can be filled in (related classes or files: Android: TXVodConstants; iOS: TXVodPlayConfig.h). {Turn off: AUDIO_NORMALIZATION_OFF} {On: AUDIO_NORMALIZATION_ON} AUDIO_NORMALIZATION_STANDARD AUDIO_NORMALIZATION_LOW AUDIO_NORMALIZATION_HIGH Custom value can be filled in: ranging from -70 to 0 LUFS.

Video-Related Interfaces

API	Description
enableHardwareDecode	Enable or disable video hardware decoding.
snapshot	Retrieve the current video frame. Note: Since retrieving the current frame image is a time-consuming operation, the screenshot will be returned via an asynchronous callback.
setMirror	Set the image.
setRate	Set the playback speed of on-demand video, default is 1.0.
getBitrateIndex	Return the bitrate index of current playback.
getSupportedBitrates	When the playback address is HLS, return the list of supported bitrates (resolutions).
setBitrateIndex	Set the bitrate index of the currently playing content. Switch resolution seamlessly. Clarity switching may require a short while.
setRenderMode	Set Image Tiling Mode.

setRenderRotation	Set Image Rendering Angle.
setAutoMaxBitrate	Set the switchable highest bitrate for adaptive playback.

Audio-Related Interfaces

API	Description
setMute	Set whether to mute playback.
setAudioPlayoutVolume	Set volume level, range: 0 - 100.
setRequestAudioFocus	Set whether to automatically obtain audio focus. Default is automatic obtain.

Event Notification API

API	Description
setVodSubtitleDataListener	Set video subtitle text data output callback.
setVodListener	Set player event callback.

TRTC Playback-Related APIs

Through the following interfaces, you can push the audio and video stream of a VOD player via TRTC. For more TRTC services, please see the TRTC product overview.

API	Description
attachTRTC	Bind VOD to TRTC service.
detachTRTC	Unbind VOD from TRTC service.
publishVideo	Start pushing video stream.
unpublishVideo	Cancel pushing video stream.
publishAudio	Start pushing audio stream.
unpublishAudio	Cancel pushing audio stream.

API Detail

TXVodPlayer

Create a video-on-demand (VOD) player instance.

```
public TXVodPlayer(Context context)
```

Parameter Description

Parameter Name	Type	Description
context	Context	System

startVodPlay

Start playing through URL.
Starting from version 10.7, you need to pass through XLiveBase#setLicence to set the Licence only after which successful playback can be achieved; otherwise, playback will fail (screen goes black). The global setting only needs to be done once.

```
public int startVodPlay(String playUrl)
```

Parameter Description

Parameter Name	Type	Description
playUrl	String	Playback address.

Return Value

0: Playback success.
Playback failure.

startVodPlay

Start playing through Tencent Cloud fileId.
Starting from version 10.7, you need to pass through XLiveBase#setLicence to set the Licence only after which successful playback can be achieved; otherwise, playback will fail (screen goes black). The global setting only needs to be done once.

```
public void startVodPlay(TXPlayInfoParams playInfoParams)
```

Parameter Description

Parameter Name	Type	Description
playInfoParams	TXPlayInfoParams	System

Return Value

0: Playback success.

Playback failure.

startPlayDrm

Play Drm-encrypted video.

Starting from version 10.7, you need to pass through XLiveBase#setLicence to set the Licence only after which successful playback can be achieved; otherwise, playback will fail (screen goes black). The global setting only needs to be done once.

Notes:

Note: Only supported in player premium.

```
public int startPlayDrm(TXPlayerDrmBuilder playerDrmBuilder)
```

Parameter Description

Parameter Name	Type	Description
playerDrmBuilder	TXPlayerDrmBuilder	Playback information for Drm.

Return Value

0: Playback success.

Non-zero: playback failure.

stopPlay

Stop playback.

```
public int stopPlay(boolean isNeedClearLastImg)
```

Parameter Description

Parameter Name	Type	Description
isNeedClearLastImg	boolean	Whether to clear the last frame. true: Clear the last frame. It is recommended to clear when stopping playback normally. false: Retain the last frame.

isPlaying

Whether it is playing.

```
public boolean isPlaying()
```

pause

Pause playback.

```
public void pause()
```

resume

Resume playback.

```
public void resume()
```

seek

Navigate to the specified time point in the video stream.

```
public void seek(int time)
```

Parameter Description

Parameter Name	Type	Description
time	int	Video stream time point, in seconds.

Return Value

seek

Navigate to the specified time point in the video stream.

```
public void seek(float time)
```

Parameter Description

Parameter Name	Type	Description
time	float	Video stream time point, in seconds, with precision up to 3 decimal places.

seek

Navigate to the specified time point in the video stream.

```
public void seek(float timeInSeconds, boolean isAccurateSeek)
```

Parameter Description

Parameter Name	Type	Description

timeInSeconds	float	Video stream time point, in seconds, with precision up to 3 decimal places.
isAccurateSeek	boolean	Whether it is precise Seek. true: Indicates precise Seek, must find the current time point, which may take longer. false: Indicates imprecise Seek, that is, searching for the previous I-frame.

seekToPdtTime

Navigate to the specified time point in the video stream.

Notes:

Note: Player Premium version 11.6 or later is supported.

```
public void seekToPdtTime(long pdtTimeMs)
```

Parameter Description

Parameter Name	Type	Description
pdtTimeMs	long	Video stream PDT time point, in milliseconds.

getCurrentPlaybackTime

Retrieve the current playback time point, in seconds.

```
public float getCurrentPlaybackTime()
```

getBufferDuration

Retrieve the total duration of the cache, in seconds.

```
public float getBufferDuration()
```

getDuration

Retrieve the total video duration, in seconds.

```
public float getDuration()
```

getPlayableDuration

Retrieve the currently playable video duration, in seconds.

```
public float getPlayableDuration()
```

getWidth

Get video width.

```
public int getWidth()
```

getHeight

Retrieve video height.

```
public int getHeight()
```

setAutoPlay

Set whether to autoplay after startPlay. Default is autoplay.

```
public void setAutoPlay(boolean autoPlay)
```

setStartTime

Set playback start time, in seconds, need to be set before starting playing.

```
public void setStartTime(float pos)
```

Parameter Description

Parameter Name	Type	Description
pos	float	Video stream time point, in seconds, with precision up to 3 decimal places.

setToken

Set encryption HLS token. After setting this value, the player automatically adds voddrm.token before the file name in the URL.

```
public void setToken(String token)
```

getEncryptedPlayKey

Retrieve the encryption key for secure encrypted playback.

```
public static String getEncryptedPlayKey(final String key)
```

setLoop

Set whether to loop playback. Default is non-loop playback.

```
public void setLoop(boolean loop)
```

isLoop

Whether to loop playback.

```
public void setLoop(boolean loop)
```

addSubtitleSource

Add external subtitle.

Note: Only supported in player premium.

```
public void addSubtitleSource(@NonNull String url, @NonNull String name, String mim
```

Parameter Description

Parameter Name	Type	Description
url	String	Subtitle address, supports Http link and local storage absolute path.
name	String	Subtitle name. If adding multiple subtitles, set the subtitle names to different names to distinguish them from other added subtitles; otherwise, it may result in subtitle selection error.
mimeType	String	Subtitle type, only supports VVT and SRT format. VTT format: TXVodConstants#VOD_PLAY_MIMETYPE_TEXT_SRT SRT format: TXVodConstants#VOD_PLAY_MIMETYPE_TEXT_VTT

getSubtitleTrackInfo

Return subtitle track information list.

Note:

Only supported in player premium.

```
public List<TXTrackInfo> getSubtitleTrackInfo()
```

Return Value

List<TXTrackInfo>: subtitle track information list.

getAudioTrackInfo

Return audio track information list.

Note:

Only supported in player premium.

```
public List<TXTrackInfo> getAudioTrackInfo()
```

Return Value

List<TXTrackInfo>: audio track information list.

selectTrack

Select a track.

Note:

Only supported in player premium.

```
public void selectTrack(int trackIndex)
```

Parameter Description

Parameter Name	Type	Description
trackIndex	int	Track index, obtained via TXTrackInfo#getTrackIndex.

deselectTrack

Cancel track selection.

Note:

Only supported in player premium.

```
public void deselectTrack(int trackIndex)
```

Parameter Description

Parameter Name	Type	Description
trackIndex	int	Track index, obtained via TXTrackInfo#getTrackIndex.

setConfig

Set player configuration information. Refer to TXVodPlayConfig.

```
public void setConfig(TXVodPlayConfig config)
```

Parameter Description

setPlayerView

Set the video rendering TXCloudVideoView for the player. It is only valid when set before starting playback.

```
public void setPlayerView(TXCloudVideoView glRootView)
```

Parameter Description

Parameter Name	Type	Description
glRootView	TXCloudVideoView	video rendering View.

setPlayerView

Set the video rendering TextureRenderView for the player, which is only valid before starting playback.

```
public void setPlayerView(TextureRenderView glRootView)
```

Parameter Description

Parameter Name	Type	Description
glRootView	TextureRenderView	video rendering View.

setSurface

Set the video rendering Surface for the player, which is only valid before starting playback.

```
public void setSurface(Surface surface)
```

setStringOption

Set the player's business parameters, with the parameter format as `<String,Object>` .

```
public void setStringOption(String key, Object value)
```

setSubtitleStyle

Set subtitle style information. Supports configuration before playback and allows updating subtitle styles after playback.

Note:

Only supported in player premium.

```
public void setSubtitleStyle(TXSubtitleRenderModel renderModel)
```

Parameter Description

Parameter Name	Type	Description
renderModel	TXSubtitleRenderModel	Subtitle style configuration parameters.

setSubtitleView

Set the subtitle rendering target object View.

Note:

Only supported in player premium.

```
public void setSubtitleView(TXSubtitleView subtitleView)
```

Parameter Description

Parameter Name	Type	Description
subtitleView	TXSubtitleView	The subtitle rendering target object View provided by player SDK.

setAudioNormalization

Set volume balance, loudness range: -70 ~ 0 (LUFS).

Note:

Only supported in player premium.

```
public void setAudioNormalization(float value)
```

Parameter Description

Parameter Name	Type	Description
value	float	Preset value can be filled in (related class or file: Android: TXVodConstants; iOS: TXVodPlayConfig.h) Turn off: AUDIO_NORMALIZATION_OFF On: AUDIO_NORMALIZATION_ON AUDIO_NORMALIZATION_STANDARD AUDIO_NORMALIZATION_LOW AUDIO_NORMALIZATION_HIGH Custom value can be filled in: ranging from -70 to 0 LUFS.

setAutoMaxBitrate

Set the switchable highest bitrate for adaptive playback.

```
public void setAutoMaxBitrate(int autoMaxBitrate)
```

enableHardwareDecode

Enable or disable video hardware decoding. Hardware decoding is enabled by default.

```
public boolean enableHardwareDecode(boolean enable)
```

snapshot

Retrieve the current video frame.

```
public void snapshot(TXLivePlayer.ITXSnapshotListener listener)
```

Parameter Description

Parameter Name	Type	Description
listener	public interface ITXSnapshotListener { void onSnapshot(Bitmap bmp); }	Callback interface class for screenshots.

setMirror

Set mirror video.

```
public void setMirror(boolean mirror)
```

setRate

Set the playback speed of on-demand video, default is 1.0.

```
public void setRate(float rate)
```

getBitrateIndex

Return the bitrate index of current playback.

```
public TXVodPlayer(Context context)
```

getSupportedBitrates

When the playback address is HLS, return the list of supported bitrates (resolutions).

```
public ArrayList<TXBitrateItem> getSupportedBitrates
```

Return Value

ArrayList<TXBitrateItem>: Bitrate list.

setBitrateIndex

Set the bitrate index of the currently playing content. Tencent Cloud supports multi-bitrate HLS fragment alignment to ensure the best experience. When switching resolution seamlessly, clarity switching may require a short while.

```
public void setBitrateIndex(int index) {
```

Parameter Description

Parameter Name	Type	Description
index	int	Bitrate index. index == -1 indicates enabling HLS adaptive bitrate streaming. index > 0 means manually switching to the corresponding clarity bitrate. The index value can be obtained through the API TXVodPlayer#getSupportedBitrates.

setRenderMode

Set the player's Image Tiling Mode.

```
public void setRenderMode(int mode)
```

Parameter Description

Parameter Name	Type	Description
mode	int	Image Tiling Mode, valid values: TXVodConstants#RENDER_MODE_FULL_FILL_SCREEN: Fill the video footage to the entire screen, scale the image proportionally to cover the whole screen, and crop the excess part. In this mode, there is no black border on the screen. TXVodConstants#RENDER_MODE_ADJUST_RESOLUTION: Adapt the video footage to the screen, scale the image proportionally, ensure the scaled width and height do not exceed the display area, display it centered, and may leave a black border.

setRenderRotation

Set the image rendering angle of the player.


```
public void setRenderRotation(int rotation)
```

Parameter Description

Parameter Name	Type	Description
context	Context	Image rendering angle, valid values: TXVodConstants#RENDER_ROTATION_PORTRAIT: portrait TXVodConstants#RENDER_ROTATION_LANDSCAPE: Rotate 90 degrees clockwise.

setMute

Set whether to mute playback. Default is unmuted playback.

```
public void setMute(boolean mute)
```

setAudioPlayoutVolume

Set volume level, range: 0 - 100.

```
public void setAudioPlayoutVolume(int volume)
```

setRequestAudioFocus

Set whether to automatically obtain audio focus. Default is automatic obtain.

false can be passed in, then the business manages audio focus itself.

```
public boolean setRequestAudioFocus(boolean requestFocus)
```

setVodSubtitleDataListener

Set video subtitle text data output callback.

```
public void setVodSubtitleDataListener(ITXVodPlayListener.ITXVodSubtitleDataListene
```

Parameter Description

Parameter Name	Type	Description
listener	ITXVodPlayListener.ITXVodSubtitleDataListener	Subtitle text data callback API.

setVodListener

Create a video-on-demand (VOD) player instance.

```
public void setVodListener(ITXVodPlayListener listener)
```

Parameter Description

Parameter Name	Type	Description
listener	ITXVodPlayListener	Player event callback API.

attachTRTC

Bind VOD to TRTC service.

```
public void attachTRTC(Object trtcCloud)
```

detachTRTC

Unbind VOD from TRTC service.

```
public void detachTRTC()
```

publishVideo

Start pushing video stream.

```
public void publishVideo()
```

unpublishVideo

Cancel pushing video stream.

```
public void unpublishVideo()
```

publishAudio

Start pushing audio stream.

```
public void publishAudio()
```

unpublishAudio

Cancel pushing audio stream.

```
public void unpublishAudio()
```

ITXVodPlayListener

Last updated：2025-05-30 15:01:18

ITXVodPlayListener API Introduction

Playback event and network event callback monitoring API for on-demand player

Callback API Overview

API	Description
onPlayEvent	Playback event notification.
onNetStatus	Playback process network status event callback.

Callback API Detail

onPlayEvent

Playback event notifications, including start playback, first frame event, Loading event, playback progress, end playback, and other events.

```
public void onPlayEvent(final TXVodPlayer player, final int event, final Bundle par
```

Parameter Description

Parameter Name	Type	Description
player	TXVodPlayer	Current player object.
event	int	Player event.
param	Bundle	Playback event carried parameters, saved in (Key, Value) format, where Key can refer to the event parameters in TXVodConstants.

onNetStatus

Playback process network status event callback.

```
public void onNetStatus(final TXVodPlayer player, final Bundle status)
```

Parameter Description

Parameter Name	Type	Description
player	TXVodPlayer	Current player object.
status	Bundle	Playback process network status parameters, format: (Key, Value).

ITXVodSubtitleDataListener

Last updated：2025-05-30 15:01:32

ITXVodSubtitleDataListener API Introduction

Player subtitle text callback listener

Callback API Overview

API	Description
onSubtitleData	Subtitle text callback

Callback API Details

onSubtitleData

Playback event notification, including start playback, first frame event, Loading event, playback progress, end playback, and other events.

```
public void onSubtitleData(TXVodDef.TXVodSubtitleData subtitleData)
```

Parameter Description

Parameter Name	Type	Description
subtitleData	TXVodSubtitleData	Subtitle text data. For detailed field values, see TXVodSubtitleData.

Player Config
TXPlayerGlobalSetting

Last updated：2025-05-30 15:01:58

TXPlayerGlobalSetting API Introduction

Global configuration for on-demand player.

API Overview

API	Description
setCacheFolderPath	Set the Cache directory for the playback engine.
setMaxCacheSize	Set the maximum cache size for the playback engine.
setDrmProvisionEnv	Set the Drm certificate provider environment.
setPlayCGIHosts	Set the list of domain addresses for Tencent Cloud PlayCGI hosts.

API Detail

setCacheFolderPath

Set the Cache directory for the playback engine. After configuration, the player and preloading will preferentially read from and store in this directory.

```
public static void setCacheFolderPath(String path)
```

Parameter Description

Parameter Name	Type	Description
path	String	Cache directory, Sdcard absolute path, null means caching is disabled.

setMaxCacheSize

Set the maximum cache size of the playback engine. After configuration, it will automatically clean up files in the Cache directory based on the specified value. Unit: MB.

```
public static void setMaxCacheSize(int sizeMB)
```

Parameter Description

Parameter Name	Type	Description
sizeMB	int	Maximum cache size, unit: MB.

setDrmProvisionEnv

Set the Drm certificate provider environment (Note: Supported starting from version 11.2).

```
public static void setDrmProvisionEnv(DrmProvisionEnv env)
```

Parameter Description

Parameter Name	Type	Description
env	DrmProvisionEnv	Set the Drm certificate provider environment (Note: Supported starting from version 11.2). Optional values: TXPlayerGlobalSetting.DrmProvisionEnv#DRM_PROVISION_ENV_COM, representing the use of google COM domain name certificate provider. TXPlayerGlobalSetting.DrmProvisionEnv#DRM_PROVISION_ENV_CN, representing the use of google CN domain name certificate provider.

setPlayCGIHosts

Set the list of domain addresses for Tencent Cloud PlayCGI hosts. When the request to the built-in domain fails, the set backup domain name will be enabled.

```
public static void setPlayCGIHosts(List<String> hosts)
```

Parameter Description

Parameter Name	Type	Description
hosts	List<String>	Domain address list, domain name format: playvideo.qcloud.com.

TXVodPlayConfig

Last updated：2025-05-30 15:02:13

TXVodPlayConfig API Introduction

on-demand player playback configuration, need to be set before playback.

API Overview

API	Description
setConnectRetryCount	Set the player's reconnection attempts in abnormal scenarios.
setTimeout	Set the player connection timeout period.
setPlayerType	Set the player type.
setHeaders	Set the Http header.
setEnableAccurateSeek	Set whether to seek accurately.
setAutoRotate	Set whether to automatically rotate the angle when playing MP4.
setSmoothSwitchBitrate	Set whether to smoothly switch between multiple bitrates for HLS.
setCacheMp4ExtName	Set the cache for MP4 filename extension.
setProgressInterval	Set interval for progress callbacks.
setMaxBufferSize	Set the maximum playback buffer size for the player.
setMaxPreloadSize	Set the maximum preload buffer size.
setEnableRenderProcess	Set whether the player is allowed to load post-processing services.
setPreferredResolution	Set the optimal HLS stream for startup playback.
setEncryptedMp4Level	Set MP4 encrypted playback.

setMediaType	Set the media asset type for player playback.
setExtInfo	Set the player's expanded parameters.
setPreferredAudioTrack	Set the preferred audio track for startup playback.

API Detail

setConnectRetryCount

Set the player's reconnection attempts in abnormal scenarios.
When the SDK is disconnected due to an exception from the server, it will attempt to reconnect to the server. Use this function to set the reconnection attempts of the SDK, with a default value of 3.

```
public void setConnectRetryCount(int count)
```

Parameter Description

Parameter Name	Type	Description
count	int	Reconnection attempts in playback exception scenarios.

setTimeout

Set the player connection timeout period, with a default value of 10 seconds.

```
public void setTimeout(int timeout)
```

Parameter Description

Parameter Name	Type	Description
interval	int	Connection timeout, in seconds, default value is 10 seconds.

setCacheFolderPath

Set the video-on-demand cache directory.
Note: This API is deprecated. Please use `TXPlayerGlobalSetting#setCacheFolderPath` for global configuration.

```
public void setCacheFolderPath(String folderPath)
```

Parameter Description

Parameter Name	Type	Description

folderPath	String	cache path.

setMaxCacheItems

Set the number of cached files.

Note: This API is deprecated. Please use TXPlayerGlobalSetting#setMaxCacheSize for global configuration.

```
public void setMaxCacheItems(int maxCacheItems)
```

Parameter Description

Parameter Name	Type	Description
maxCacheItems	int	Maximum cache entries.

setPlayerType

Set player type, default is Tencent Cloud proprietary player.

```
public void setPlayerType(int playerType)
```

Parameter Description

Parameter Name	Type	Description
playerType	int	Player type, valid values: TXVodConstants#PLAYER_SYSTEM_MEDIA_PLAYER: Android system player. TXVodConstants#PLAYER_THUMB_PLAYER: Tencent Cloud proprietary player, default value.

setHeaders

Customize configuration for the player to play with Http headers carried during the online process.

```
public void setHeaders(Map<String, String> headers)
```

Parameter Description

Parameter Name	Type	Description
headers	Map<String, String>	Custom Http header content.

setEnableAccurateSeek

Set whether to seek accurately. Default true.

```
public void setEnableAccurateSeek(boolean accurateSeek)
```

Parameter Description

Parameter Name	Type	Description
accurateSeek	boolean	Set whether to seek accurately.

setAutoRotate

When playing an MP4 file, if set to YES, it will automatically rotate according to the rotation angle in the file. The rotation angle can be obtained in the PLAY_EVT_CHANGE_ROTATION event, with a default value of YES.

```
public void setAutoRotate(boolean autoRotate)
```

Parameter Description

Parameter Name	Type	Description
autoRotate	boolean	Set whether to automatically rotate the angle during playback.

setSmoothSwitchBitrate

Set whether to smoothly switch between multiple bitrates for HLS, default false.

```
public void setSmoothSwitchBitrate(boolean smoothSwitchBitrate)
```

Parameter Description

Parameter Name	Type	Description
smoothSwitchBitrate	boolean	Whether to smoothly switch between multiple bitrates for HLS.

setCacheMp4ExtName

Set the cache for mp4 filename extension. The default is mp4.

```
public void setCacheMp4ExtName(String cacheMp4ExtName)
```

Parameter Description

Parameter Name	Type	Description
cacheMp4ExtName	boolean	Filename extension.

setProgressInterval

Set the interval for progress callbacks, with a default value of 0.5 seconds per callback.

```
public void setProgressInterval(int intervalMs)
```

Parameter Description

Parameter Name	Type	Description
intervalMs	int	Interval time, in milliseconds.

setMaxBufferSize

Set the maximum playback buffer size, in MB.

```
public void setMaxBufferSize(float maxBufferSize)
```

Parameter Description

Parameter Name	Type	Description
maxBufferSize	float	Playback buffer size

setMaxPreloadSize

Set the maximum preload buffer size, in MB.

```
public void setMaxPreloadSize(float maxPreloadSize)
```

Parameter Description

Parameter Name	Type	Description
maxPreloadSize	float	Preload size.

setFirstStartPlayBufferTime

Set the data duration for the player's initial preload, in ms, with a default value of 100ms.

Note: This API is deprecated. Please use #setMaxBufferSize or #setMaxPreloadSize to set the buffer size.

```
public void setFirstStartPlayBufferTime(int milliseconds)
```

Parameter Description

Parameter Name	Type	Description
milliseconds	int	Duration size.

setEnableRenderProcess

Set whether the player is allowed to load post-processing services, off by default.

```
public void setEnableRenderProcess(boolean enableRenderProcess)
```

Parameter Description

Parameter Name	Type	Description
enableRenderProcess	boolean	Set whether loading post-rendering post-processing services is allowed.

setPreferredResolution

When playing HLS with multiple streams, the player selects the optimal stream for startup playback based on the set preferredResolution, which is the product of width and height (width * height).
Settings are only valid before playback starts.

```
public void setPreferredResolution(long preferredResolution)
```

Parameter Description

Parameter Name	Type	Description
preferredResolution	long	Product of video width and height (width * height).

setEncryptedMp4Level

Set MP4 encrypted playback, not encrypted (default).

```
public void setEncryptedMp4Level(int level)
```

Parameter Description

Parameter Name	Type	Description
level	int	Set MP4 playback and storage encryption levels, supported starting from Player Premium 12.2, currently supports: TXVodConstants#MP4_ENCRYPTION_LEVEL_NONE: Unencrypted playback, supported by default. TXVodConstants#MP4_ENCRYPTION_LEVEL_L2: MP4 local encrypted playback.

setMediaType

Set the media asset type for player playback, default is AUTO type.

```
public void setMediaType(int mediaType)
```

Parameter Description

Parameter Name	Type	Description
mediaType	int	Set the media asset type, default is AUTO type. Optional values: TXVodConstants#MEDIA_TYPE_AUTO, AUTO type (default value, adaptive bit rate playback not supported). TXVodConstants#MEDIA_TYPE_HLS_VOD, HLS VOD media. TXVodConstants#MEDIA_TYPE_HLS_LIVE, HLS live media. TXVodConstants#MEDIA_TYPE_FILE_VOD, common file VOD media such as MP4 (supported starting from version 11.2). TXVodConstants#MEDIA_TYPE_DASH_VOD, DASH VOD media (supported starting from version 11.2)

setExtInfo

Set the player's expanded parameters.

```
public void setExtInfo(Map<String, Object> map)
```

Parameter Description

Parameter Name	Type	Description
map	Map	Expanded parameters

setPreferredAudioTrack

Set the preferred audio track for startup playback, supported starting from Player Premium version 12.3.

```
public void setPreferredAudioTrack(String audioTrackName)
```

Parameter Description

Parameter Name	Type	Description
audioTrackName	String	audio track name.

Media Download TXVodDownloadManager

Last updated：2025-05-30 15:02:39

TXVodDownloadManager API Introduction

On-demand video player download API class
Video download supports downloading MP4 and HLS videos, corresponds to nested HLS videos, requires specifying preferred resolution (preferredResolution).

Class Constant

Parameter Name	Type	Value	Description
TXVodDownloadManager#DOWNLOAD_SUCCESS	int	0	Download success.
TXVodDownloadManager#DOWNLOAD_AUTH_FAILED	int	-5001	File ID download authentication failure.
TXVodDownloadManager#DOWNLOAD_NO_FILE	int	-5003	File does not exist for download.
TXVodDownloadManager#DOWNLOAD_FORMAT_ERROR	int	-5004	Unsupported download format.
TXVodDownloadManager#DOWNLOAD_DISCONNECT	int	-5005	Network error.
TXVodDownloadManager#DOWNLOAD_HLS_KEY_ERROR	int	-5006	Failed to retrieve HLS decryption Key.
TXVodDownloadManager#DOWNLOAD_PATH_ERROR	int	-5007	Download directory access failure.
TXVodDownloadManager#DOWNLOAD_403FORBIDDEN	int	-5008	Signature expired or invalid request.

API Overview

API	Description
getInstance	Retrieve the TXVodDownloadManager instance object in singleton mode.
setHeaders	Set the HTTP request header for downloading.
setListener	Set the download callback method, which must be configured before downloading.
startDownloadUrl	Start the download using the URL method.
startDownload	Start the download using the fileId method.
startDownloadDrm	Start the download using the Drm method.
stopDownload	Stop downloading, stopped successfully when ITXVodDownloadListener.onDownloadStop callback is triggered.
deleteDownloadMediaInfo	Delete download information.
getDownloadMediaInfoList	Retrieve download list information for all users, time-consuming API, do not call on the main thread.
getDownloadMediaInfo	Retrieve download information, time-consuming API, do not call on the main thread.
getDownloadMediaInfo	Retrieve download information, time-consuming API, do not call on the main thread.

API Detail

getInstance

Retrieve the TXVodDownloadManager instance object in singleton mode.

```
public static TXVodDownloadManager getInstance()
```

setHeaders

Set the HTTP request header for downloading.

```
public void setHeaders(Map<String, String> headers)
```

setListener

Set the download callback method, which must be configured before downloading.

```
public void setListener(ITXVodDownloadListener listener)
```

Parameter Description

Parameter Name	Type	Description
listener	ITXVodDownloadListener	Download listening status callback.

startDownloadUrl

Start the download using the URL method.

Set the cache directory of the playback engine before starting the download.

`TXPlayerGlobalSetting#setCacheFolderPath` .

```
public TXVodDownloadMediaInfo startDownloadUrl(String url, long preferredResolution
```

Parameter Description

Parameter Name	Type	Description
url	String	Download address, mandatory parameter; otherwise, the download will fail.
preferredResolution	long	Download preference resolution, multiple resolution URLs are mandatory parameters, with values being the width * height of the preferred resolution (e.g., for 720p, input 921600 = 1280*720), and -1 for single resolution.
userName	String	Account name, optional parameter; defaults to "default" if not provided.

Return Value

Video download information TXVodDownloadMediaInfo.

startDownloadUrl

Start the download using the URL method.

Abandoned, recommended for use: startDownloadUrl(String, long, String)

```
public TXVodDownloadMediaInfo startDownloadUrl(String url, String userName)
```

startDownloadUrl

Start the download using the URL method.

Abandoned, recommended for use: startDownloadUrl(String, long, String)

```
public TXVodDownloadMediaInfo startDownloadUrl(String url)
```

startDownload

Start the download using the Tencent Cloud Video fileId method.

```
public TXVodDownloadMediaInfo startDownload(final TXVodDownloadDataSource dataSourc
```

Parameter Description

Parameter Name	Type	Description
dataSource	TXVodDownloadDataSource	Download a resource object.

Return Value

Video download information TXVodDownloadMediaInfo.

startDownloadDrm

Start the download using the Drm method.

```
public TXVodDownloadMediaInfo startDownloadDrm(final TXPlayerDrmBuilder drmBuilder,
```

Parameter Description

Parameter Name	Type	Description
drmBuilder	TXPlayerDrmBuilder	Construct a Drm resource.
preferredResolution	long	Download preference resolution. Multiple resolution URLs are required parameters, with values being the width * height of the preferred resolution (e.g., for 720p, input 921600 = 1280*720). For single resolution, input -1.
userName	String	Account name, optional parameter; defaults to "default" if not provided.

Return Value

Video download information TXVodDownloadMediaInfo.

stopDownload

Stop downloading, stopped successfully when ITXVodDownloadListener.onDownloadStop callback is triggered.

```
public void stopDownload(TXVodDownloadMediaInfo downloadMediaInfo)
```

Parameter Description

Parameter Name	Type	Description
downloadMediaInfo	TXVodDownloadMediaInfo	Video download information.

deleteDownloadFile

Delete downloaded files.

Abandoned, recommended for use: deleteDownloadMediaInfo.

```
public boolean deleteDownloadFile(String playPath)
```

Parameter Description

Parameter Name	Type	Description
playPath	String	File path.

deleteDownloadMediaInfo

Delete download information.

```
public boolean deleteDownloadMediaInfo(TXVodDownloadMediaInfo downloadMediaInfo)
```

Parameter Description

Parameter Name	Type	Description
downloadMediaInfo	TXVodDownloadMediaInfo	Video download information.

getDownloadMediaInfoList

Retrieve download list information for all users.

```
public List<TXVodDownloadMediaInfo> getDownloadMediaInfoList()
```

Return Value

Video download information list: List<TXVodDownloadMediaInfo>.

getDownloadMediaInfo

Retrieve download information.

Call this API and ensure that the download task parameters are created via `TXVodDownloadDataSource(int, String, int, String, String)` beforehand.

Refer to `TXVodDownloadMediaInfo#getDataSource and TXVodDownloadDataSource#getUserName` .

```
public TXVodDownloadMediaInfo getDownloadMediaInfo(int appId, String fileId, int qu
```

Parameter Description

Parameter Name	Type	Description
appId	int	Tencent Cloud Video on Demand (VOD) application appId.
fileId	String	Tencent Cloud Video on Demand (VOD) video fileId.
qualityId	int	Video QUALITY Id, refer to the constant TXVodDownloadDataSource#QUALITY_240P.
userName	String	Account name, must match the account name passed in during download. If not passed in during download, pass in an empty string "" here.

Return Value

Video download information TXVodDownloadMediaInfo.

getDownloadMediaInfo

Retrieve download information. Call this API and ensure that the download is started by calling startDownloadUrl(String, long, String) beforehand.

```
public TXVodDownloadMediaInfo getDownloadMediaInfo(String url, long preferredResolu
```

Parameter Description

Parameter Name	Type	Description
url	String	Download address, mandatory parameter; otherwise, the download will fail.
preferredResolution	long	Download preference resolution. Multiple resolution URLs are required parameters, with values being the width * height of the preferred resolution (e.g., for 720p, input 921600 = 1280*720). For single resolution, input -1.
userName	String	Account name, optional parameter; defaults to "default" if not provided.

Return Value

Video download information TXVodDownloadMediaInfo.

getDownloadMediaInfo

Retrieve URL download information. Abandoned, recommended for use: getDownloadMediaInfo(String, long, String).

```
public TXVodDownloadMediaInfo getDownloadMediaInfo(String url)
```

getDownloadMediaInfo

Retrieve URL download information. Abandoned, recommended for use: getDownloadMediaInfo(String, long, String).

```
public TXVodDownloadMediaInfo getDownloadMediaInfo(int appId, String fileId, int qu
```

ITXVodDownloadListener

Last updated：2025-05-30 15:07:00

ITXVodDownloadListener API Introduction

On-demand player download callback listener API

Callback API Overview

API	Description
onDownloadStart	Download started.
onDownloadProgress	Download progress updated.
onDownloadStop	Download stopped.
onDownloadFinish	Download completed.
onDownloadError	Error encountered during download.
hlsKeyVerify	Download HLS, encounter an encrypted file, provide the decryption key for external verification.

Callback API Details

onDownloadStart

Download started.

```
void onDownloadStart(TXVodDownloadMediaInfo mediaInfo)
```

Parameter Description

Parameter Name	Type	Description
mediaInfo	TXVodDownloadMediaInfo	Video download information.

onDownloadProgress

Download progress updated.

```
void onDownloadProgress(TXVodDownloadMediaInfo mediaInfo)
```

Parameter Description

Parameter Name	Type	Description
mediaInfo	TXVodDownloadMediaInfo	Video download information.

onDownloadStop

Download stopped. This callback is received when calling the TXVodDownloadManager#stopDownload method.

```
void onDownloadStop(TXVodDownloadMediaInfo mediaInfo)
```

Parameter Description

Parameter Name	Type	Description
mediaInfo	TXVodDownloadMediaInfo	Video download information.

onDownloadFinish

Download completed.

```
void onDownloadFinish(TXVodDownloadMediaInfo mediaInfo)
```

Parameter Description

Parameter Name	Type	Description
mediaInfo	TXVodDownloadMediaInfo	Video download information.

onDownloadError

Error encountered during download.

```
void onDownloadError(TXVodDownloadMediaInfo mediaInfo, int error, String reason)
```

Parameter Description

Parameter Name	Type	Description
mediaInfo	TXVodDownloadMediaInfo	Video download information.
error	int	Download error code, refer to Download Error Codes.

reason	String	Download error message.

hlsKeyVerify

Download HLS, encounter an encrypted file, provide the decryption key for external verification.

TXVodDownloadDataSource.

Deprecated interface, integrate spacetime implementation only need to.

```
int hlsKeyVerify(TXVodDownloadMediaInfo mediaInfo, String url, byte[] receive)
```

Parameter Description

Parameter Name	Type	Description
mediaInfo	TXVodDownloadMediaInfo	Video download information.
url	String	video download URL.
receive	byte[]	receive server return value.

Return Value:

0: Verification passed, continue downloading.

Other values: Verification failed, throw download error.

TXVodDownloadDataSource

Last updated：2025-05-30 15:07:15

TXVodDownloadDataSource Overview

On-demand download resource object.

Class Constant

Parameter Name	Type	Value	Description
TXVodDownloadDataSource#QUALITY_OD	int	0	Original video quality.
TXVodDownloadDataSource#QUALITY_240P	int	240	Smooth 240P.
TXVodDownloadDataSource#QUALITY_360P	int	360	Smooth 360P.
TXVodDownloadDataSource#QUALITY_480P	int	480	Standard definition 480P.
TXVodDownloadDataSource#QUALITY_540P	int	540	Standard definition 540P.
TXVodDownloadDataSource#QUALITY_720P	int	720	HD 720P.
TXVodDownloadDataSource#QUALITY_1080P	int	1080	Full HD 1080P.

API Detail

TXVodDownloadDataSource

Construct a download resource object for Tencent Cloud Video fileId download.

```
public TXVodDownloadDataSource(int appId, String fileId, int quality, String pSign,
```

Parameter Description

Parameter Name	Type	Description
appId	int	Tencent Cloud Video on Demand (VOD) application appId.
fileId	String	Tencent Cloud Video on Demand (VOD) video fileId.

qualityId	int	Video QUALITY Id, refer to the TXVodDownloadDataSource#QUALITY_240P constant for details.
pSign	String	video playback signature
userName	String	account name, must be consistent with the account name passed in during download. If not passed in during download, pass in the empty character "" here.

TXVodDownloadDataSource

Construct a download resource object for Tencent Cloud Video fileId V2 version download.
Abandoned, recommended for use: TXVodDownloadDataSource(int appId, String fileId, int quality, String pSign, String userName).

```
public TXVodDownloadDataSource(TXPlayerAuthBuilder authBuilder, int quality)
```

TXVodDownloadDataSource

Construct a download resource object for Tencent Cloud Video fileId V2 version download.
Abandoned, recommended for use: TXVodDownloadDataSource(int appId, String fileId, int quality, String pSign, String userName).

```
public TXVodDownloadDataSource(TXPlayerAuthBuilder authBuilder, String templateName
```

setToken

Set this value, and the player will automatically add voddrm.token.<Token> before the file name in the URL.

```
public void setToken(String token)
```

setQuality

Set the video QUALITY Id.

```
public void setQuality(int quality)
```

Parameter Description

Parameter Name	Type	Description
quality	int	Video QUALITY Id, refer to the TXVodDownloadDataSource#QUALITY_240P constant for details.

getAppId

Retrieve the passed-in appId.

```
public int getAppId()
```

getFileId

Retrieve the passed-in fileId.

```
public String getFileId()
```

getPSign

Retrieve the passed-in download signature.

```
public String getPSign()
```

getQuality

Retrieve the passed-in quality.

```
public int getQuality()
```

getUserName

Retrieve the passed-in userName, default "default".

```
public String getUserName()
```

getToken

Retrieve the passed-in token.

```
public String getToken()
```

TXVodDownloadMediaInfo

Last updated：2025-05-30 15:07:30

Downloading TXVod Media Information Overview

Download on-demand media asset description.

Class Constant

Parameter Name	Type	Value	Description
TXVodDownloadMediaInfo#STATE_INIT	int	0	Download initial state.
TXVodDownloadMediaInfo#STATE_START	int	1	Download started.
TXVodDownloadMediaInfo#STATE_STOP	int	2	Download stopped.
TXVodDownloadMediaInfo#STATE_ERROR	int	3	Download error.
TXVodDownloadMediaInfo#STATE_FINISH	int	4	Download completed.

API Overview

API	Description
getDataSource	Retrieve the passed-in download source media asset information when downloading with Tencent Cloud Video fileId.
getDuration	Get the total duration of the video, in milliseconds.
getPlayableDuration	Get the playable duration of the downloaded content, in milliseconds.
getSize	Retrieve the download file total size, in Byte, only valid for fileId download source. Note: The total size refers to the size of the original file uploaded to the Tencent Cloud Video on Demand (VOD) console. The substream size after adaptive dynamic streaming cannot be accessed temporarily.
getDownloadSize	Retrieve the downloaded file size, in Byte, only valid for fileId download source.
getProgress	Retrieve the current download progress.

getPlayPath	Retrieve the playback path of the currently downloaded resource, which can be transmitted to TXVodPlayer for playback.
getDownloadState	Retrieve the download status.
isDownloadFinished	Determine whether the download is complete.
getSpeed	Retrieve the download speed, in KByte per second. (Supported starting from version 10.9.)
isResourceBroken	Check if the downloaded video resource is damaged. Return true if it is deleted after download or other similar cases. (Supported starting from version 11.0.)
getTaskId	Retrieve task id, unique representation of the download task.
getUrl	Retrieve the actual download address.
getUserName	Retrieve the download account name.
getPreferredResolution	Retrieve the preferred download resolution.

API Detail

getDataSource

Retrieve the passed-in download source media asset information when downloading with Tencent Cloud Video fileId.

```
public TXVodDownloadDataSource getDataSource()
```

Return Value

Download resource object information:TXVodDownloadDataSource

getDuration

Get the total duration of the video, in milliseconds.

```
public int getDuration()
```

getPlayableDuration

Get the playable duration of the downloaded content, in milliseconds.

```
public TXVodDownloadDataSource(TXPlayerAuthBuilder authBuilder, String templateName
```

getSize

Retrieve the download file total size, in Byte, only valid for Tencent Cloud Video fileId download source.
Note: The total size refers to the size of the original file uploaded to the Tencent Cloud Video on Demand (VOD) console. The substream size after adaptive dynamic streaming cannot be accessed temporarily.

```
public long getSize()
```

getDownloadSize

Retrieve the downloaded file size, in Byte, only valid for Tencent Cloud Video fileId download source.

```
public long getDownloadSize()
```

getProgress

Retrieve the current download progress.

```
public float getProgress()
```

getPlayPath

Retrieve the playback path of the currently downloaded resource, which can be transmitted to TXVodPlayer for playback.

```
public String getPlayPath()
```

getDownloadState

Retrieve the download status.

```
public int getDownloadState()
```

isDownloadFinished

Determine whether the download is complete.

```
public boolean isDownloadFinished()
```

getSpeed

Retrieve the download speed, in KByte per second. (Supported starting from version 10.9.)

```
public int getSpeed()
```

getTaskId

Retrieve task id, unique representation of the download task.

```
public int getTaskId()
```

isResourceBroken

Check if the downloaded video resource is damaged. Return true if it is deleted after download or other similar cases. (Supported starting from version 11.0.)

```
public boolean isResourceBroken()
```

getUrl

Retrieve the actual download address.

```
public String getUrl()
```

getUserName

Retrieve the download account name.

```
public String getUserName()
```

getPreferredResolution

Retrieve the preferred download resolution.

```
public long getPreferredResolution()
```

TXVodPreloadManager

Last updated：2025-05-30 15:07:48

Introducing the TXVodPreloadManager API

On-demand player preload API class
No need to create a player instance. Pre-download partial video content to speed up startup time when using the player, delivering a better playback experience.

API Overview

API	Description
getInstance	Retrieve the TXVodPreloadManager instance object in singleton mode.
startPreload:URL	Start preloading through URL.
startPreload:TXPlayInfoParams	Start preloading through fileId or URL. Preferentially use this API.
stopPreload	Stop preloading.

API Detail

getInstance

Retrieve the TXVodPreloadManager instance object in singleton mode.

```
public static TXVodPreloadManager getInstance(Context context)
```

startPreload:URL

Start preloading through URL.
Before starting preloading, set the cache directory `TXPlayerGlobalSetting#setCacheFolderPath` and cache size `TXPlayerGlobalSetting#setMaxCacheSize` for the playback engine. This setting is a global configuration and must be consistent with the player; otherwise, it can lead to playback cache failure.

```
public int startPreload(final String url, final float preloadSizeMB, final long pre
```

Parameter Description

Parameter Name	Type	Description
url	String	Preloaded video URL.
preloadSizeMB	float	Preload size, unit: MB.
preferredResolution	long	Expected download resolution, the product of video width and height (width * height). Pass -1 when multiple resolutions are not supported or no specification is required.
listener	ITXVodPreloadListener	Preloading listening status callback.

Return Value

task ID, use this task ID to stop preloading `TXVodPreloadManager#stopPreload`.

If the return value is -1, it means this task ID is invalid.

startPreload:TXPlayInfoParams

Start preloading, support through Tencent Cloud fileId and video URL.

If TXPlayInfoParams#getUrl is not null, preferentially start video URL preloading, at this point support calling in the mainline.

If TXPlayInfoParams#getFileId is not null, start video fileId preloading, at this point does not support calling in the mainline.

Note:

1. Preloading is a time-consuming operation. Do not call it on the main thread; calling it on the main thread will throw an illegal call exception.
2. Set the cache directory `TXPlayerGlobalSetting#setCacheFolderPath` and cache size `TXPlayerGlobalSetting#setMaxCacheSize` of the playback engine before starting preloading. This setting is a global configuration and must be consistent with the player; otherwise, it can lead to playback cache failure.

```
public int startPreload(final TXPlayInfoParams playInfoParams, final float preloadS
        final ITXVodFilePreloadListener listener)
```

Parameter Description

Parameter Name	Type	Description
playInfoParams	TXPlayInfoParams	Download information. Set the http request headers for preloading using TXPlayInfoParams#setHeaders, and set the preferred audio track name for preloading using TXPlayInfoParams#setPreferAudioTrack.

preloadSizeMB	float	Preload size, unit: MB.
preferredResolution	long	Expected download resolution, the product of video width and height (width * height). Pass -1 when multiple resolutions are not supported or no specification is required.
listener	ITXVodFilePreloadListener	Preloading listening status callback.

Return Value

task ID, use this task ID to stop preloading `TXVodPreloadManager#stopPreload`.

If the return value is -1, it means this task ID is invalid.

stopPreload

Stop preloading.

```
public void stopPreload(int taskID)
```

Parameter Description

Parameter Name	Type	Description
taskID	int	task ID. The ID is obtained from the return value of `TXVodPreloadManager#startPreload`.

ITXVodPreloadListener

Last updated：2025-05-30 15:08:02

Introducing the ITXVodPreloadListener API

Pre-download URL state callback listener API

Callback API Overview

API	Description
onComplete	Video preloading completed.
onError	Video preloading error.

Callback API Details

onComplete

Preloading completed with successful callback.

```
void onComplete(int taskID, String url);
```

Parameter Description

Parameter Name	Type	Description
taskID	int	Pre-download task ID.
url	String	Pre-download task URL.

onError

Pre-download failure callback.

```
void onError(int taskID, String url, int code, String msg)
```

Parameter Description

Parameter Name	Type	Description

taskID	int	Pre-download task ID.
url	String	Pre-download task URL.
code	int	Error code.
msg	String	Error information.

ITXVodFilePreloadListener

Last updated：2025-05-30 15:08:18

Introducing the ITXVodFilePreloadListener API

Tencent Cloud Video fileId and URL preload status callback listener API

Callback API Overview

API	Description
onStart	Video preloading started. For fileId preloading, callback before starting preloading after link replacement succeeds.
onComplete	Video preloading completed.
onError	Video preloading error.

Callback API Details

onStart

Preloading completed with successful callback.

```
public void onStart(int taskID, String fileId, String url, Bundle bundle)
```

Parameter Description

Parameter Name	Type	Description
taskID	int	Pre-download task ID.
fileId	int	Pre-downloaded video fileId.
url	String	Pre-download task URL, which is the video URL after link replacement, can be used for subsequent playback.
bundle	Bundle	Additional information carried by preloading.

onComplete

Preloading completed with successful callback.

```
void onComplete(int taskID, String url);
```

Parameter Description

Parameter Name	Type	Description
taskID	int	Pre-download task ID.
url	String	Pre-download task URL.

onError

Pre-download failure callback.

```
void onError(int taskID, String url, int code, String msg)
```

Parameter Description

Parameter Name	Type	Description
taskID	int	Pre-download task ID.
url	String	Pre-download task URL.
code	int	Error code.
msg	String	Error information.

Type Definition
TXVodConstants

Last updated：2025-05-30 15:08:44

TXVodConstants API Introduction

Constant class used in VOD player

Image Tiling Mode

Value	Parameter Name	Description
0	RENDER_MODE_FULL_FILL_SCREEN	Video footage fills the screen.
1	RENDER_MODE_ADJUST_RESOLUTION	Video footage adapts to the screen.

Image Rendering Angle

Value	Parameter Name	Description
0	RENDER_ROTATION_PORTRAIT	portrait.
270	RENDER_ROTATION_LANDSCAPE	Rotate 90 degrees clockwise.

Playing Event List

Value	Parameter Name	Description
2002	VOD_PLAY_EVT_HIT_CACHE	Play start cache hit
2003	VOD_PLAY_EVT_RCV_FIRST_I_FRAME	Network reception receives the first video packet (IDR).
2004	VOD_PLAY_EVT_PLAY_BEGIN	Video playback start.
2005	VOD_PLAY_EVT_PLAY_PROGRESS	Video playback progress.

2006	VOD_PLAY_EVT_PLAY_END	Video playback end.
6001	VOD_PLAY_EVT_LOOP_ONCE_COMPLETE	Loop one round playback ends.
2007	VOD_PLAY_EVT_PLAY_LOADING	Video playback Loading.
2008	VOD_PLAY_EVT_START_VIDEO_DECODER	Decoder starts up.
2009	VOD_PLAY_EVT_CHANGE_RESOLUTION	Video resolution changed.
2010	VOD_PLAY_EVT_GET_PLAYINFO_SUCC	Successfully retrieve VOD file information.
2011	VOD_PLAY_EVT_CHANGE_ROTATION	Video rotation information.
2013	VOD_PLAY_EVT_VOD_PLAY_PREPARED	Video loading complete.
2014	VOD_PLAY_EVT_VOD_LOADING_END	loading complete.
2017	VOD_PLAY_EVT_FIRST_VIDEO_PACKET	Received the first frame of data (supported starting from version 12.0).
2019	VOD_PLAY_EVT_SEEK_COMPLETE	Seek completed (supported starting from version 10.3).
2020	VOD_PLAY_EVT_SELECT_TRACK_COMPLETE	Track switch completed.
2026	VOD_PLAY_EVT_RCV_FIRST_AUDIO_FRAME	First audio playback.
2103	VOD_PLAY_WARNING_RECONNECT	Network disconnection, auto reconnection started.
2030	VOD_PLAY_EVT_VIDEO_SEI	Video sei information event.
2031	VOD_PLAY_EVT_HEVC_DOWNGRADE_PLAYBACK	HEVC downgrade playback.
-2301	VOD_PLAY_ERR_NET_DISCONNECT	Network disconnection, multiple reconnection attempts failed.
-2303	VOD_PLAY_ERR_FILE_NOT_FOUND	File does not exist.
-2304	VOD_PLAY_ERR_HEVC_DECODE_FAIL	HEVC decoding failed.
-2305	VOD_PLAY_ERR_HLS_KEY	HLS decryption key acquisition failed.
-2306	VOD_PLAY_ERR_GET_PLAYINFO_FAIL	Failed to retrieve VOD file information.
2106	VOD_PLAY_WARNING_HW_ACCELERATION_FAIL	Hardware decoding startup failed, use software decoding.

-5	VOD_PLAY_ERR_INVALID_LICENCE	Invalid license, playback failure. Before calling startVodPlay, you must configure the license using TXLiveBase#setLicence for successful playback; otherwise, playback will fail (black screen). The license only needs to be set globally once. Live stream publishing license, Short Video License, and player license can all be used. If you haven't obtained the above licenses yet, you can click Player License to apply. An official license requires purchase.
-6004	VOD_PLAY_ERR_SYSTEM_PLAY_FAIL	System player playback error.
-6006	VOD_PLAY_ERR_DECODE_VIDEO_FAIL	Video decoding error, video format not supported.
-6007	VOD_PLAY_ERR_DECODE_AUDIO_FAIL	Audio decoding error, audio format not supported.
-6008	VOD_PLAY_ERR_DECODE_SUBTITLE_FAIL	Subtitle decoding error.
-6009	VOD_PLAY_ERR_RENDER_FAIL	Video rendering error.
-6010	VOD_PLAY_ERR_PROCESS_VIDEO_FAIL	Video post-processing error.
-6101	VOD_PLAY_ERR_DRM	DRM playback failure

Playing Event Parameters

Value	Parameter Name	Description
"CPU_USAGE"	NET_STATUS_CPU_USAGE	Current instantaneous CPU utilization.
"VIDEO_WIDTH"	NET_STATUS_VIDEO_WIDTH	Video resolution - Width.
"VIDEO_HEIGHT"	NET_STATUS_VIDEO_HEIGHT	Video resolution - Height.
"NET_SPEED"	NET_STATUS_NET_SPEED	Current network data reception speed, unit: KBps.

"VIDEO_FPS"	NET_STATUS_VIDEO_FPS	Current streaming media video frame rate.
"VIDEO_BITRATE"	NET_STATUS_VIDEO_BITRATE	Current streaming media video bitrate, unit: bps.
"AUDIO_BITRATE";	NET_STATUS_AUDIO_BITRATE	Current streaming media audio bitrate, unit: bps.
"VIDEO_CACHE"	NET_STATUS_VIDEO_CACHE	Buffer (jitterbuffer) size, current buffer length is 0, indicating buffering is imminent, unit: KBps.
"SERVER_IP"	NET_STATUS_SERVER_IP	Connected server IP.
"EVT_UTC_TIME"	EVT_UTC_TIME	UTC time.
"EVT_TIME"	EVT_TIME	Event occurrence time.
"EVT_MSG"	EVT_DESCRIPTION	Event description.
"EVT_PARAM1"	EVT_PARAM1	Event parameter 1.
"EVT_PARAM2"	EVT_PARAM2	Event parameter 2.
"EVT_PLAY_COVER_URL"	EVT_PLAY_COVER_URL	Video cover.
"EVT_PLAY_URL"	EVT_PLAY_URL	Video address.
"EVT_PLAY_NAME"	EVT_PLAY_NAME	Video name.
"EVT_PLAY_DESCRIPTION"	EVT_PLAY_DESCRIPTION	Video description.
"EVT_PLAY_PROGRESS_MS"	EVT_PLAY_PROGRESS_MS	Playback progress (ms).
"EVT_PLAY_DURATION_MS"	EVT_PLAY_DURATION_MS	Playable duration (ms).
"EVT_PLAY_PROGRESS"	EVT_PLAY_PROGRESS	Playback progress.
"EVT_PLAY_DURATION"	EVT_PLAY_DURATION	Playable duration.
"EVT_PLAYABLE_DURATION_MS"	EVT_PLAYABLE_DURATION_MS	On-demand playable duration (ms).

"EVT_PLAYABLE_RATE"	EVT_PLAYABLE_RATE	Playback speed.
"EVT_PLAYABLE_DURATION"	EVT_PLAYABLE_DURATION	On-demand playable duration.
"EVT_IMAGESPRIT_WEBVTTURL"	EVT_IMAGESPRIT_WEBVTTURL	sprite sheet web vtt description file download URL.
"EVT_IMAGESPRIT_IMAGEURL_LIST"	EVT_IMAGESPRIT_IMAGEURL_LIST	sprite sheet image download URL.
"EVT_DRM_TYPE"	EVT_DRM_TYPE	encryption type.
"EVT_CODEC_TYPE"	EVT_CODEC_TYPE	video codec type.
"EVT_KEY_FRAME_CONTENT_LIST"	EVT_KEY_FRAME_CONTENT_LIST	video keyframe description.
"EVT_KEY_FRAME_TIME_LIST"	EVT_KEY_FRAME_TIME_LIST	keyframe time.
"EVT_PLAY_PDT_TIME_MS"	EVT_PLAY_PDT_TIME_MS	Playback PDT time (ms).
"EVT_KEY_VIDEO_ROTATION"	EVT_KEY_VIDEO_ROTATION	MP4 video rotation angle.
"EVT_KEY_WATER_MARK_TEXT"	EVT_KEY_WATER_MARK_TEXT	ghost watermark text (supported starting from version 11.5).

Playing Media Asset Type

Value	Parameter Name	Description
0	MEDIA_TYPE_AUTO	AUTO type.
1	MEDIA_TYPE_HLS_VOD	Adaptive bitrate playback of HLS VOD media.
2	MEDIA_TYPE_HLS_LIVE	Adaptive bitrate playback of HLS live media.
3	MEDIA_TYPE_FILE_VOD	Playback of common file VOD media such as MP4.
4	MEDIA_TYPE_DASH_VOD	Adaptive bitrate playback of DASH VOD media.

MP4 Encryption Level

Value	Parameter Name	Description
0	MP4_ENCRYPTION_LEVEL_NONE	MP4 Unencrypted playback.
2	MP4_ENCRYPTION_LEVEL_L2	L2, MP4 Local encrypted playback.

Unclassified Variable

Value	Parameter Name	Description
0	PLAYER_SYSTEM_MEDIA_PLAYER	System player.
1	PLAYER_THUMB_PLAYER	Proprietary player, supports software decoding, with better compatibility.
-1	INDEX_AUTO	Adaptive bitrate index flag.
"450"	PLAYER_OPTION_KEY_SUBTITLE_OUTPUT_TYPE	External subtitle output type configuration Key
"backup_url"	VOD_KEY_BACKUP_URL	Downgrade playback alternate URL Key.
"mimetype"	VOD_KEY_MIMETYPE	Playback resource Mimetype Key.
"text/x-subrip"	VOD_PLAY_MIMETYPE_TEXT_SRT	External

		subtitle in SRT format.
"text/vtt"	VOD_PLAY_MIMETYPE_TEXT_VTT	External subtitle in VTT format.
"EVT_KEY_WATER_MARK_TEXT"	EVT_KEY_WATER_MARK_TEXT	ghost watermark text (supported starting from version 11.5).

TXPlayInfoParams

Last updated：2025-05-30 15:08:59

TXPlayInfoParams API Introduction

VOD player plays media parameters, and you can configure Tencent Cloud fileId and url playback through TXPlayInfoParams.

API Overview

API	Description
TXPlayInfoParams:fileId	Create a media instance that plays via Tencent Cloud VOD fileId.
TXPlayInfoParams:url	Create a media instance that plays through URL.
setMediaType	Set the media asset type for player playback.
setHeaders	Set the Http header.
setEncryptedMp4Level	Set MP4 encrypted playback.
setPreferAudioTrack	Set the preferred audio track for playback startup.

API Details

TXPlayInfoParams:fileId

Create a media instance that plays via Tencent Cloud VOD fileId.

```
public TXPlayInfoParams(int appId, String fileId, String pSign)
```

Parameter Description

Parameter Name	Type	Description
appId	int	Tencent Cloud Video on Demand (VOD) application appId.
fileId	String	Tencent Cloud Video on Demand (VOD) resource fileId.

pSign	String	Playback signature

TXPlayInfoParams:url

Create a media instance that plays through url.

```
public TXPlayInfoParams(String url)
```

Parameter Description

Parameter Name	Type	Description
url	String	Play the resource address.

setMediaType

Set the media type.

```
public void setMediaType(int mediaType)
```

Parameter Description

Parameter Name	Type	Description
mediaType	int	Set the media asset type. The default is AUTO type. Optional values: TXVodConstants#MEDIA_TYPE_AUTO, AUTO type (default value, adaptive bit rate playback not supported). TXVodConstants#MEDIA_TYPE_HLS_VOD, HLS VOD media. TXVodConstants#MEDIA_TYPE_HLS_LIVE, HLS live media. TXVodConstants#MEDIA_TYPE_FILE_VOD, common file VOD media such as MP4 (supported starting from version 11.2). TXVodConstants#MEDIA_TYPE_DASH_VOD, DASH VOD media (supported starting from version 11.2).

setHeaders

Customize the player configuration to play the Http header carried during the online process.

```
public void setHeaders(Map<String, String> headers)
```

Parameter Description

Parameter Name	Type	Description

headers	Map<String, String>	Custom Http header content.

setEncryptedMp4Level

Set MP4 encrypted playback, not encrypted (default).

```
public void setEncryptedMp4Level(int level)
```

Parameter Description

Parameter Name	Type	Description
level	int	Set MP4 playback and storage encryption levels, supported starting from Player Premium 12.2, currently supports: TXVodConstants#MP4_ENCRYPTION_LEVEL_NONE: No encryption playback, supported by default. TXVodConstants#MP4_ENCRYPTION_LEVEL_L2: MP4 local encryption playback.

setPreferredAudioTrack

Set the preferred audio track for playback startup. Supported starting from Player Premium version 12.3.

In the pre-download scenario, configure to preferentially download the audio track.

```
public void setPreferredAudioTrack(String audioTrackName)
```

Parameter Description

Parameter Name	Type	Description
audioTrackName	String	audio track name.

TXVodDef

Last updated：2025-05-30 15:09:14

Common Structure

TXVodSubtitleData

Subtitle text data from VOD player callback.

Parameter Name	Type	Description
subtitleData	String	Subtitle content, empty indicates no subtitle content.
trackIndex	long	Current subtitle track's trackIndex.
durationMs	long	Subtitle duration in milliseconds. Temporarily empty, do not use.
startPositionMs	long	Subtitle start time in milliseconds. Temporarily empty, do not use.

TXTrackInfo

Last updated：2025-05-30 15:09:27

TXTrackInfo Overview

Detailed information of the track played by the on-demand player.

API Detail

Parameter Name	Type	Description
trackIndex	int	Track index.
trackType	int	track type. Valid values: TXTrackInfo#TX_VOD_MEDIA_TRACK_TYPE_VIDEO: Video track. TXTrackInfo#TX_VOD_MEDIA_TRACK_TYPE_AUDIO: Audio track. TXTrackInfo#TX_VOD_MEDIA_TRACK_TYPE_SUBTITLE: Subtitle track.
name	String	Track name.
isSelected	boolean	Whether the current track is selected.
isExclusive	boolean	If it is true, only one track of this type can be selected at any given moment. If it is false, multiple tracks of this type can be selected simultaneously.
isInternal	boolean	Whether the current track is an internal primitive track.

getTrackIndex

Retrieve the track index.

```
public int getTrackIndex()
```

getTrackType

Retrieve the track type.

```
public int getTrackType()
```

getName

Retrieve the track name.

```
public int getName()
```

TXSubtitleRenderModel

Last updated：2025-05-30 15:09:40

TXSubtitleRenderModel Overview

VOD player subtitle style rendering parameters.

Field Details

fontColor

Text color, in ARGB format. If not set, defaults to opaque white (0xFFFFFFFF).

```
public int fontColor
```

fontSize

Font size. If fontSize is set, canvasWidth and canvasHeight must be set; otherwise, the internal system does not know what size to reference for font rendering. If fontSize is not set, the internal system will use the default font size.

```
public float fontSize
```

familyName

Font family name. On Android, the default is "Roboto". If the string is not null, it is considered set; if it is null, it is considered not set.

```
public String familyName
```

canvasWidth

canvasWidth and canvasHeight are the dimensions of the subtitle render canvas. The aspect ratio of canvasWidth and canvasHeight must match the aspect ratio of the video; otherwise, the rendered text will be deformed. If not set, the player will use the size of the current video as the render canvas size.

```
public int canvasWidth
```

canvasHeight

canvasWidth and canvasHeight define the size of the subtitle render canvas. Their aspect ratio must match the video's aspect ratio; otherwise, rendered text will be distorted. If not specified, the player will use the current video's dimensions as the render canvas size.

```
public int canvasHeight
```

isBondFontStyle

Whether it is bold text, with the default value being normal font.

```
public boolean isBondFontStyle
```

outlineWidth

stroke width, if not set, the internal will use the default stroke width.

```
public float outlineWidth
```

outlineColor

Stroke color, in ARGB format. If not set, defaults to opaque black (0xFF000000).

```
public int outlineColor
```

lineSpace

Line spacing: if lineSpace is set, canvasWidth and canvasHeight must be set; if not set, the internal will use the default line spacing.

```
public float lineSpace
```

startMargin

startMargin, endMargin, and verticalMargin define the drawing area for subtitles. If not set, use the settings in the subtitle file; if the subtitle file does not define them, use the default.
Once startMargin, endMargin, and yMargin are set, and the subtitle file defines one or more of these parameters, they will override the corresponding parameters in the subtitle file. The following diagram illustrates the meaning of these parameters in horizontal writing direction. Use the notes for each parameter to understand.

The margin along the direction of subtitle text content varies in meaning based on different writing orientations. startMargin is a proportional value with a range of [0, 1], representing the proportion relative to the video image size.

For horizontal writing direction, startMargin represents the distance from the left side of the subtitle to the left side of the video image. For example, startMargin=0.05 means the margin is 0.05 times the video width (5%).
For vertical writing direction (regardless of right-to-left or left-to-right), startMargin represents the distance from the top of the subtitle to the top of the video image. For example, startMargin=0.05 means the margin is 0.05 times the video height (5%).

```
public float startMargin
```

endMargin

Margin along the direction of subtitle text content

```
public float endMargin
```

verticalMargin

Margin in the vertical subtitle text direction

```
public float verticalMargin
```

TXBitrateItem

Last updated：2025-05-30 15:09:54

TXBitrateItem Overview

Video bitrate information.

Field Details

Parameter Name	Type	Description
index	int	bitrate index, serial number in the m3u8 file.
width	int	The width of this video stream
height	int	The height of this video stream.
bitrate	int	The bitrate of this video stream.

TXPlayerDrmBuilder

Last updated：2025-05-30 15:10:06

TXPlayerDrmBuilder Overview

Drm playback information, used in conjunction with TXVodPlayer#startPlayDrm.

API Overview

API	Description
TXPlayerDrmBuilder	Construct a Drm playback information object.
setDeviceCertificateUrl	Set the certificate provider URL.
setKeyLicenseUrl	Set the decryption Key URL.
setPlayUrl	Set the media playback URL.

API Detail

TXPlayerDrmBuilder

Constructor.

```
public TXPlayerDrmBuilder(String licenseUrl, String playUrl)
```

Parameter Description

Parameter Name	Type	Description
licenseUrl	String	Play media URL.
playUrl	String	Decryption Key URL.

setDeviceCertificateUrl

Set the certificate provider URL.

```
public TXPlayerDrmBuilder setDeviceCertificateUrl(String deviceCertificateUrl)
```

Parameter Description

Parameter Name	Type	Description
deviceCertificateUrl	String	Certificate provider URL. If it is Widevine, this field can be left blank and will follow the default process.

setKeyLicenseUrl

Set the decryption Key URL.

```
public TXPlayerDrmBuilder setKeyLicenseUrl(String keyLicenseUrl)
```

setPlayUrl

Set the media playback URL.

```
public TXPlayerDrmBuilder setPlayUrl(String playUrl)
```

TXImageSprite

Last updated：2025-05-30 15:10:20

TXImageSprite Overview

On-demand sprite sheet parsing tool class.

API Overview

API	Description
TXImageSprite	Constructor.
setVTTUrlAndImageUrls	Set sprite sheet address.
getThumbnail	Retrieve a thumbnail.
release	Release resources, call after usage is complete, otherwise it can lead to memory leak.

API Detail

TXImageSprite

Constructor.

```
public TXImageSprite(Context context)
```

setVTTUrlAndImageUrls

Set the sprite sheet address. After configuration, it will start a child thread to download the sprite sheet and parse it.

```
public void setVTTUrlAndImageUrls(String vttUrl, List<String> imagesUrl)
```

Parameter Description

Parameter Name	Type	Description
vttUrl	String	sprite sheet web vtt description file download URL.

imagesUrl	List<String>	sprite sheet image download URL.

getThumbnail

Retrieve a thumbnail, return null if retrieval fails.

```
public Bitmap getThumbnail(float time)
```

Parameter Description

Parameter Name	Type	Description
time	float	Time point, unit: seconds.

release

Release resources, call after usage is complete, otherwise it can lead to memory leak.

```
public void release()
```

Flutter

Last updated：2025-06-23 11:43:05

`SuperPlayerPlugin` Class

setGlobalLicense

Description

This API is used to set the license.

After you apply for and get a license, you can use the following API to initialize it. We recommend you call this API during application start. Video playback will fail if the license is not set.

API

```
static Future<void> setGlobalLicense(String licenceUrl, String licenceKey) async;
```

Parameter description

Parameter	Type	Description
licenceUrl	String	The license URL
licenceKey	String	The license key

Return values

Unlimited

createVodPlayer

Description

This API is used to create a VOD player instance at the native layer. If you use `TXVodPlayerController` , as it already integrates an instance, you don't need to create another.

API

```
static Future<int?> createVodPlayer() async;
```

Parameter description

Unlimited

Return values

Returned Value	Type	Description
playerId	int	The player ID

createLivePlayer

Description

This API is used to create a live player instance at the native layer. If you use `TXVodPlayerController`, as it already integrates an instance, you don't need to create another.

API

```
static Future<int?> createLivePlayer() async;
```

Parameter description

Unlimited

Return values

Returned Value	Type	Description
playerId	int	The player ID

setConsoleEnabled

Description

This API is used to enable/disable player native log output.

API

```
static Future<int?> setConsoleEnabled() async;
```

Parameter description

Parameter	Type	Description
enabled	bool	Enables/Disables player log

Return values

Unlimited

releasePlayer

Description

This API is used to release the player resources.

API

```
static Future<int?> releasePlayer(int? playerId) async;
```

Parameter description

Unlimited

Return values

Unlimited

setGlobalMaxCacheSize

Description

This API is used to set the maximum cache size of the player engine. After setting, the backend will clear files in the cache directory automatically according to the set value.

API

```
static Future<void> setGlobalMaxCacheSize(int size) async;
```

Parameter description

Parameter	Type	Description
size	int	The maximum cache size in MB

Return values

Unlimited

setGlobalCacheFolderPath

Description

This API is used to set the cache path, which is set to the application's sandbox directory by default. You only need to pass in the relative cache directory instead of the entire absolute path to the parameter.

API

```
static Future<bool> setGlobalCacheFolderPath(String postfixPath) async;
```

Parameter description

Parameter	Type	Description
postfixPath	String	The cache path, which is set to the application's sandbox directory by default. You only need to pass in the relative cache directory instead of the entire absolute path to `postfixPath`. On Android, videos will be cached to the `Android/data/your-pkg-name/files/testCache` directory on the SD card. On iOS, videos will be cached to the `Documents/testCache` directory in the sandbox.

Return values

Unlimited

setLogLevel

Description

This API is used to set the log output level.

API

```
static Future<void> setLogLevel(int logLevel) async;
```

Parameter description

Parameter	Type	Description
logLevel	int	`0`: Logs at all levels; `1`: DEBUG, INFO, WARNING, ERROR, and FATAL logs; `2`: INFO, WARNING, ERROR, and FATAL logs; `3`: WARNING, ERROR, and FATAL logs; `4`: ERROR and FATAL logs; `5`: Only FATAL logs; `6`: No SDK logs.

Return values

Unlimited

setBrightness

Description

This API is used to set the brightness. It is applicable only to the current application.

API

```
static Future<void> setBrightness(double brightness) async;
```

Parameter description

Parameter	Type	Description
brightness	double	The brightness level. Value range: 0.0–1.0

Return values

Unlimited

restorePageBrightness

Description

This API is used to reset the UI brightness. It is applicable only to the current application.

API

```
static Future<void> restorePageBrightness() async;
```

Parameter description

Unlimited

Return values

Unlimited

getBrightness

Description

This API is used to get the brightness level of the current UI.

API

```
static Future<double> getBrightness() async;
```

Parameter description

Unlimited

Return values

Parameter	Type	Description
brightness	double	The brightness level. Value range: 0.0–1.0

setSystemVolume

Description

This API is used to set the volume level of the current system.

API

```
static Future<void> setSystemVolume(double volume) async;
```

Parameter description

Parameter	Type	Description
volume	double	The volume level. Value range: 0.0–1.0

Return values

Unlimited

getSystemVolume

Description

This API is used to set the volume level of the current system.

API

```
static Future<double> getSystemVolume() async;
```

Parameter description

Unlimited

Returned value description

Parameter	Type	Description
volume	double	The volume level. Value range: 0.0–1.0

abandonAudioFocus

Description

This API is used to release the audio focus. It is applicable only to Android.

API

```
static Future<double> abandonAudioFocus() async;
```

Parameter description

Unlimited

Return values

Unlimited

requestAudioFocus

Description

This API is used to request the audio focus. It is applicable only to Android.

API

```
static Future<void> requestAudioFocus() async ;
```

Parameter description

Unlimited

Return values

Unlimited

isDeviceSupportPip

Description

This API is used to check whether the current device supports the picture-in-picture (PiP) mode.

API

```
static Future<int> isDeviceSupportPip() async;
```

Parameter description

Unlimited

Returned value description

Parameter	Type	Description
isDeviceSupportPip	int	`0` : The PiP mode can be enabled; `-101` : The Android version is too early; `-102` : The PiP permission is disabled or the device doesn't support PiP; `-103` : The current UI has been terminated.

getLiteAVSDKVersion

Description

This API is used to get the version number of the current native-layer player SDK.

API

```
static Future<String?> getLiteAVSDKVersion() async;
```

Parameter description

Unlimited

Returned value description

Parameter	Type	Description
sdkVersion	String	The version of the current player SDK.

startVideoOrientationService

Description

Start monitoring the device's rotation direction. Once enabled, if the device's auto-rotate feature is turned on, the player will automatically rotate the video direction based on the current device orientation.

This interface currently only applies to the Android platform. The iOS platform will automatically enable this feature.

Note

Before calling this interface, please be sure to inform the user of the privacy risks.

API

```
static Future<bool> startVideoOrientationService() async
```

Parameter description

Unlimited

Returned value description

Parameter	Type	Description
result	bool	true means the feature was successfully enabled, while false means it failed to enable. Possible reasons for failure include premature activation before the context is

		initialized or failure to obtain the sensor.

registerSysBrightness

Description

Enable or disable the monitoring of system brightness. If enabled, when the system brightness changes, it will change the current window brightness and callback the brightness to the Flutter layer. This interface needs to be used in conjunction with setBrightness and onExtraEventBroadcast.

API

```
static Future<void>registerSysBrightness(bool isRegister) async
```

Parameter description

Parameter	Type	Description
isRegister	bool	true：Turn on listening. false：Turns off monitoring.

Returned value description

Unlimited

setUserId

Description

Set the userId, generally used for console data tracking.

API

```
static Future<void> setUserId(String userId) async
```

Parameter description

Parameter	Type	Description
userId	String	userId

Returned value description

Unlimited

setSDKListener

Description

Set up SDK monitoring, currently supports callback listening for license status settings.

API

```
void setSDKListener({FTXLicenceLoadedListener? licenceLoadedListener})
```

Parameter description

Parameter	Type	Description
licenceLoadedListener	FTXLicenceLoadedListener	The parameters include result and reason. A result of 0 indicates that the license verification was successful, while the result contains the verification message.

Returned value description

Unlimited

setLicenseFlexibleValid

Description

Enable flexible license verification for the player. Once enabled, the first two playback verifications will be automatically passed after the player is launched for the first time.

API

```
static Future<void> setLicenseFlexibleValid(bool enabled) async
```

Parameter description

Parameter	Type	Description
enabled	bool	Whether to enable flexible verification.

Returned value description

Unlimited

setDrmProvisionEnv

Description

Configure the DRM certificate environment.

API

```
static Future<void> setDrmProvisionEnv(TXDrmProvisionEnv env) async
```

Parameter description

Parameter	Type	Description
env	TXDrmProvisionEnv	DRM_PROVISION_ENV_COM: COM domain certificate provider.

		DRM_PROVISION_ENV_CN: CN domain certificate provider.

Returned value description

Unlimited

`TXVodPlayerController` Class

initialize

Description

This API is used to initialize the controller and request assignment of shared textures.

Note:

Versions 12.3.1 and later no longer require this call.

API

```
Future<void> initialize({bool? onlyAudio}) async;
```

Parameter description

Parameter	Type	Description
onlyAudio	bool	Whether the player is an audio-only player. This parameter is optional.

Returned value description

Unlimited

startVodPlay

Note

Starting from v10.7, `startPlay` is replaced by `startVodPlay`, and playback will succeed only after you use `{@link SuperPlayerPlugin#setGlobalLicense}` to set the license; otherwise, playback will fail (black screen occurs). The license needs to be set only once globally. You can use the license for CSS, UGSV, or video playback. If you have no such licenses, you can quickly apply for a trial license or purchase an official license.

Description

This API is used to play back a video via URL.

API

```
Future<bool> startVodPlay(String url) async;
```

Parameter description

Parameter	Type	Description
url	String	The URL of the video to be played back.

Returned value description

Parameter	Type	Description
result	bool	Whether creation succeeded.

startVodPlayWithParams

Note

Starting from v10.7, `startPlay` is replaced by `startVodPlay`, and playback will succeed only after you use `{@link SuperPlayerPlugin#setGlobalLicense}` to set the license; otherwise, playback will fail (black screen occurs). The license needs to be set only once globally. You can use the license for CSS, UGSV, or video playback. If you have no such licenses, you can quickly apply for a trial license or purchase an official license.

Description

This API is used to play back a video via `fileId`.

API

```
Future<void> startVodPlayWithParams(TXPlayInfoParams params) async;
```

Parameter description

Parameter	Type	Description
appId	int	The application's `appId`, which is required.
fileId	String	The file ID, which is required.
sign	String	The hotlink protection signature. For more information, see Overview.
url	String	Video url, either this field or fileId can be filled.

Returned value description

Unlimited

pause

Description

This API is used to pause a video during playback.

API

```
Future<void> pause() async;
```

Parameter description

Unlimited

Returned value description

Unlimited

resume

Description

This API is used to resume the playback of a paused video.

API

```
Future<void> resume() async;
```

Parameter description

Unlimited

Returned value description

Unlimited

stop

Description

This API is used to stop a video during played back.

API

```
Future<bool> stop({bool isNeedClear = false}) async;
```

Parameter description

Parameter	Type	Description
isNeedClear	bool	Whether to clear the last-frame image.

Returned value description

Parameter	Type	Description
result	bool	Whether stop succeeded.

setIsAutoPlay

Description

This API is used to set whether to automatically play back the video after calling `startVodPlay` to load the video URL.

API

```
Future<void> setIsAutoPlay({bool? isAutoPlay}) async;
```

Parameter description

Parameter	Type	Description
isAutoPlay	bool	Whether to play back the video automatically.

Returned value description

Unlimited

isPlaying

Description

This API is used to query whether the player is currently playing a video.

API

```
Future<bool> isPlaying() async;
```

Parameter description

Unlimited

Returned value description

Parameter	Type	Description
isPlaying	bool	Whether playback is ongoing.

setMute

Description

This API is used to set whether to mute the current playback.

API

```
Future<void> setMute(bool mute) async;
```

Parameter description

Parameter	Type	Description
mute	bool	Whether to mute the playback.

Returned value description

Unlimited

setLoop

Description

This API is used to specify whether to loop the video after the video playback ends.

API

```
Future<void> setLoop(bool loop) async;
```

Parameter description

Parameter	Type	Description
loop	bool	Whether to loop the video.

Returned value description

Unlimited

seek

Description

This API is used to adjust the playback progress to the specified time.

API

```
_controller.seek(progress);
```

Parameter description

Parameter	Type	Description
progress	double	Target playback time in seconds.

Returned value description

Unlimited

setRate

Description

This API is used to set the playback rate.

API

```
Future<void> setRate(double rate) async;
```

Parameter description

Parameter	Type	Description
rate	double	Video playback rate. Default value: 1.0

Returned value description

Unlimited

getSupportedBitrates

Description

This API is used to get the bitrates supported by the video being played back.

API

```
Future<List?> getSupportedBitrates() async;
```

Parameter description

Unlimited

Returned value description

Returned Value	Type	Description
index	int	Bitrate number
width	int	The video width for the bitrate
height	int	Video height for the bitrate
bitrate	int	Bitrate value

getBitrateIndex

Description

This API is used to get the set bitrate number.

API

```
Future<int> getBitrateIndex() async;
```

Parameter description

Unlimited

Returned value description

Returned Value	Type	Description
index	int	Bitrate number

setBitrateIndex

Description

This API is used to set the current bitrate by bitrate number.

API

```
Future<void> setBitrateIndex(int index) async;
```

Parameter description

Returned Value	Type	Description
index	int	The bitrate number. `-1` indicates to enable adaptive bitrate streaming.

Returned value description

Unlimited

setStartTime

Description

This API is used to specify the playback start time.

API

```
Future<void> setStartTime(double startTime) async;
```

Parameter description

Returned Value	Type	Description
startTime	double	The playback start time in seconds.

Returned value description

Unlimited

setAudioPlayoutVolume

Description

This API is used to set the video volume level.

API

```
Future<void> setAudioPlayoutVolume(int volume) async;
```

Parameter description

Parameter	Type	Description

volume	int	Video volume level. Value range: 0–100

Returned value description

Unlimited

setRequestAudioFocus

Description

This API is used to set the audio focus. It is applicable only to Android.

API

```
Future<bool> setRequestAudioFocus(bool focus) async;
```

Parameter description

Parameter	Type	Description
focus	bool	Whether to set the audio focus.

Returned value description

Parameter	Type	Description
result	bool	Whether the focus is set successfully.

setConfig

Description

This API is used to configure the player.

API

```
Future<void> setConfig(FTXVodPlayConfig config) async ;
```

Parameter description

Parameter	Type	Description
config	FTXVodPlayConfig	For more information, see the `FTXVodPlayConfig` class.

Returned value description

Unlimited

getCurrentPlaybackTime

Description

This API is used to get the current playback time in seconds.

API

```
Future<double> getCurrentPlaybackTime() async;
```

Parameter description

Unlimited

Returned value description

Parameter	Type	Description
playbackTime	double	The current playback time in seconds.

getBufferDuration

Description

This API is used to get the currently buffered video duration in seconds.

API

```
Future<double> getBufferDuration();
```

Parameter description

Unlimited

Returned value description

Parameter	Type	Description
playbackTime	double	The currently buffered video duration in seconds.

getPlayableDuration

Description

This API is used to get the playable duration of the video being played back in seconds.

API

```
Future<double> getPlayableDuration() async;
```

Parameter description

Unlimited

Returned value description

Parameter	Type	Description
playableDuration	double	The currently playable video duration in seconds.

getWidth

Description

This API is used to get the width of the video being played back.

API

```
Future<int> getWidth() async;
```

Parameter description

Unlimited

Returned value description

Parameter	Type	Description
width	int	The current video width

getHeight

Description

This API is used to get the height of the video being played back.

API

```
Future<int> getHeight() async;
```

Parameter description

Unlimited

Returned value description

Parameter	Type	Description
height	int	The current video height.

setToken

Description

This API is used to set the token for HLS encryption. After the token is set, the player will automatically add `voddrm.token` before the filename in the URL.

API

```
Future<void> setToken(String? token) async;
```

Parameter description

Parameter	Type	Description
token	String	The token for video playback.

Returned value description

Unlimited

isLoop

Description

This API is used to get the current playback loop status of the player.

API

```
Future<bool> isLoop() async;
```

Parameter description

Unlimited

Returned value description

Parameter	Type	Description
isLoop	bool	Whether the player is in loop status.

enableHardwareDecode

Description

This API is used to enable/disable playback based on hardware decoding. After the value is set, it will not take effect until the video playback is restarted.

API

```
Future<bool> enableHardwareDecode(bool enable);
```

Parameter description

Parameter	Type	Description
enable	bool	Whether to enable hardware decoding.

Returned value description

Parameter	Type	Description
result	bool	The hardware/software decoding setting result.

dispose

Description

This API is used to terminate the controller. After it is called, all notification events will be terminated, and the player will be released.

API

```
Future<void> dispose() async;
```

Parameter description

Unlimited

Returned value description

Unlimited

getDuration

Description

This API is used to get the total video duration.

API

```
Future<double> getDuration() async;
```

Parameter description

Unlimited

Returned value description

Parameter	Type	Description
duration	double	The total video duration in seconds.

enterPictureInPictureMode

Description

This API is used to enter the PiP mode.

API

```
Future<int> enterPictureInPictureMode({String? backIconForAndroid, String? playIcon
```

Parameter description

The parameters are applicable only to Android.

Parameter	Type	Description
backIcon	String	The seek backward icon, which can be up to 1 MB in size as limited by Android. It is optional, and if it is not set, the system icon will be used.
playIcon	String	The playback icon, which can be up to 1 MB in size as limited by Android. It is optional, and if it is not set, the system icon will be used.
pauseIcon	String	The pause icon, which can be up to 1 MB in size as limited by Android. It is optional, and if it is not set, the system icon will be used.

forwardIcon	String	The fast forward icon, which can be up to 1 MB in size as limited by Android. It is optional, and if it is not set, the system icon will be used.

Returned value description

Parameter	Code	Description
NO_ERROR	0	Started successfully with no errors.
ERROR_PIP_LOWER_VERSION	-101	The Android version is too early and doesn't support the PiP mode.
ERROR_PIP_DENIED_PERMISSION	-102	The PiP mode permission wasn't enabled, or the current device doesn't support PiP.
ERROR_PIP_ACTIVITY_DESTROYED	-103	The current UI was terminated.
ERROR_IOS_PIP_DEVICE_NOT_SUPPORT	-104	The device model or system version doesn't support PiP (only supported on iPadOS 9+ and iOS 14+). This error is applicable only to iOS.
ERROR_IOS_PIP_PLAYER_NOT_SUPPORT	-105	The player doesn't support PiP. This error is applicable only to iOS.
ERROR_IOS_PIP_VIDEO_NOT_SUPPORT	-106	The video doesn't support PiP. This error is applicable only to iOS.
ERROR_IOS_PIP_IS_NOT_POSSIBLE	-107	The PiP controller was unavailable. This error is applicable only to iOS.
ERROR_IOS_PIP_FROM_SYSTEM	-108	The PiP controller reported an error. This error is applicable only to iOS.
ERROR_IOS_PIP_PLAYER_NOT_EXIST	-109	The player object doesn’t exist. This error is applicable only to iOS.
ERROR_IOS_PIP_IS_RUNNING	-110	The PiP feature was running. This error is applicable only to iOS.
ERROR_IOS_PIP_NOT_RUNNING	-111	The PiP feature didn't start. This error is applicable only to iOS.
ERROR_IOS_PIP_START_TIME_OUT	-112	PIP launch timeout, applicable only to iOS.
ERROR_PIP_AUTH_DENIED	-201	Insufficient permissions, currently only occurs in live streaming Picture-in-Picture, applicable to iOS.

ERROR_PIP_CAN_NOT_ENTER	-120	PIP error, currently cannot enter PIP mode.

initImageSprite

Description

Initialize video sprite image

API

```
Future<void> initImageSprite(String? vvtUrl, List<String>? imageUrls) async;
```

Parameter description

Parameter	Type	Description
vvtUrl	String	Sprite Image web VTT description file download URL.
imageUrls	List<String>	Sprite image download URL.

Returned value description

Unlimited

getImageSprite

Description

Get the loaded sprite Image.

API

```
Future<Uint8List?> getImageSprite(double time) async;
```

Parameter description

Parameter	Type	Description
time	double	Time point, in seconds.

Returned value description

Parameter	Type	Description
thumb	Uint8List	Sprite Image

exitPictureInPictureMode

Description

Exit picture-in-picture mode if the player is currently in picture-in-picture mode.

API

```
Future<void> exitPictureInPictureMode() async;
```

Parameter description

Unlimited

Returned value description

Unlimited

addSubtitleSource

Description

Add external subtitles.

Note: This feature requires Player Premium version 11.7 to be supported.

API

```
Future<void> addSubtitleSource(String url, String name, {String? mimeType}) async;
```

Parameter description

Parameter	Type	Description
url	String	subtitle url
name	String	subtitle name
mimeType	String	Subtitle type, supports SRT (TXVodPlayEvent.VOD_PLAY_MIMETYPE_TEXT_SRT) and VVT (TXVodPlayEvent.VOD_PLAY_MIMETYPE_TEXT_VTT) formats

getSubtitleTrackInfo

Description

Returns the subtitle track information list.

Note: This feature requires Player Premium version 11.7 to be supported.

API

```
Future<List<TXTrackInfo>> getSubtitleTrackInfo() async;
```

Parameter description

TXTrackInfo class：

Parameter	Type	Description
trackType	int	Track type. The values are: Video track: TX_VOD_MEDIA_TRACK_TYPE_VIDEO = 1

		Audio track: TX_VOD_MEDIA_TRACK_TYPE_AUDIO = 2 Subtitle track: TX_VOD_MEDIA_TRACK_TYPE_SUBTITLE = 3
trackIndex	int	track index
name	String	track name
isSelected	bool	Whether the current track is selected
isExclusive	bool	If it is true, only one track of this type can be selected at each time. If it is false, multiple tracks of this type can be selected at the same time.
isInternal	bool	Whether the current track is an internal original track.

getAudioTrackInfo

Description

Returns the subtitle track information list.

Note: This feature requires Player Premium version 11.7 to be supported.

API

```
Future<List<TXTrackInfo>> getAudioTrackInfo() async;
```

Parameter description

Reference TXTrackInfo class

selectTrack

Description

Select track.

Note: This feature requires Player Premium version 11.7 to be supported.

API

```
Future<void> selectTrack(int trackIndex) async;
```

Parameter description

Parameter	Type	Description
trackIndex	int	Track index, trackIndex track index, obtained through the trackIndex of [TXTrackInfo].

deselectTrack

Description

Deselect the track.

Note: This feature requires Player Premium version 11.7 to be supported.

API

```
Future<void> deselectTrack(int trackIndex) async;
```

Parameter description

Parameter	Type	Description
trackIndex	int	Track index, trackIndex track index, obtained through the trackIndex of [TXTrackInfo].

setStringOption

Description

Set extended parameters.

API

```
Future<void> setStringOption(String key, Object value) async
```

Parameter description

Parameter	Type	Description
key	String	Extended parameter key-value pairs.
value	Object	Extended parameter values.

setPlayerView

Description

Bind Video Rendering Texture.

API

```
Future<void>setPlayerView(int renderViewId) async
```

Parameter description

Parameter	Type	Description
renderViewId	int	The viewId returned by the onRenderViewCreatedListener callback of TXPlayerVideo.

setRenderMode

Description

Set the screen tiling mode.

API

```
Future<void> setRenderMode(FTXPlayerRenderMode renderMode) async
```

Parameter description

Parameter	Type	Description
renderMode	FTXPlayerRenderMode	Set the screen tiling mode: one is `FTXPlayerRenderMode.ADJUST_RESOLUTION` that prioritizes displaying the complete video frame while maintaining the aspect ratio, and the other is `FTXPlayerRenderMode.FULL_FILL_CONTAINER` that fills the container while preserving the aspect ratio.

reDraw

Description

Force a screen redraw, Android only.

API

```
Future<void> reDraw() async
```

`FTXVodPlayConfig` Class

Attribute configuration description

Parameter	Type	Description
connectRetryCount	int	The number of player reconnections. If the SDK is disconnected from the server due to an exception, the SDK will attempt to reconnect to the server.
connectRetryInterval	int	The interval between two player reconnections. If the SDK is disconnected from the server due to an exception, the SDK will attempt to reconnect to the server.
timeout	int	Player connection timeout period

playerType	int	Player type. Valid values: 0: VOD; 1: live streaming; 2: live stream replay.
headers	Map	Custom HTTP headers
enableAccurateSeek	bool	Whether to enable accurate seek. Default value: `true`.
autoRotate	bool	If it is set to `true`, the MP4 file will be automatically rotated according to the rotation angle set in the file, which can be obtained from the `PLAY_EVT_CHANGE_ROTATION` event. Default value: `true`.
smoothSwitchBitrate	bool	Whether to enable smooth multi-bitrate HLS stream switch. If it is set to `false` (default), multi-bitrate URLs are opened faster. If it is set to `true`, the bitrate can be switched smoothly when IDR frames are aligned.
cacheMp4ExtName	String	The cached MP4 filename extension. Default value: `mp4`.
progressInterval	int	The progress callback interval in milliseconds. If it is not set, the SDK will call back the progress once every 0.5 seconds.
maxBufferSize	int	The maximum size of playback buffer in MB. The setting will affect `playableDuration`. The greater the value, the more the data that is buffered in advance.
maxPreloadSize	int	Maximum preload buffer size in MB
firstStartPlayBufferTime	int	Duration of the video data that needs to be loaded during the first buffering in ms. Default value: 100 ms
nextStartPlayBufferTime	int	Minimum buffered data size to stop buffering (secondary buffering for insufficient buffered data or progress bar drag buffering caused by `seek`) in ms. Default value: 250 ms
overlayKey	String	The HLS security enhancement encryption and decryption key
overlayIv	String	The HLS security enhancement encryption and decryption IV
extInfoMap	Map	Some special configuration items
enableRenderProcess	bool	Whether to allow the postrendering and postproduction feature, which is enabled by default. If the super-resolution plugin exists after it is enabled, the plugin will be loaded by default
preferredResolution	int	Resolution of the video used for playback preferably. `preferredResolution = width * height`
mediaType	int	Set the media type, with AUTO as the default value. Optional values are:

		TXVodConstants#MEDIA_TYPE_AUTO, AUTO type (default value, adaptive bit rate playback is not supported for now); TXVodConstants#MEDIA_TYPE_HLS_VOD, HLS VOD media; TXVodConstants#MEDIA_TYPE_HLS_LIVE, HLS live media; TXVodConstants#MEDIA_TYPE_HLS_VOD, MP4 and other common file VOD media (supported starting from version 11.7); TXVodConstants#MEDIA_TYPE_DASH_VOD, DASH VOD media (supported starting from version 11.7);

`TXLivePlayerController` Class

initialize

Description

This API is used to initialize the controller and request assignment of shared textures.

Note:

Versions 12.3.1 and later no longer require this call.

API

```
Future<void> initialize({bool? onlyAudio}) async;
```

Parameter description

Parameter	Type	Description
onlyAudio	bool	Whether the player is an audio-only player. This parameter is optional.

Returned value description

Unlimited

startLivePlay

Note

Starting from v10.7, `startPlay` is replaced by `startLivePlay`, and playback will succeed only after you use `{@link SuperPlayerPlugin#setGlobalLicense}` to set the license; otherwise, playback will fail (black screen occurs). The license needs to be set only once globally. You can use the license for CSS, UGSV, or video playback. If you have no such licenses, you can quickly apply for a trial license or purchase an official license.

Description

This API is used to play back a video via URL.

API

```
Future<bool> play(String url) async;
```

Parameter description

Parameter	Type	Description
url	String	The URL of the video to be played back.

Returned value description

Parameter	Type	Description
result	bool	Whether creation succeeded.

pause

Description

This API is used to pause a video during playback.

API

```
Future<void> pause() async;
```

Parameter description

Unlimited

Returned value description

Unlimited

resume

Description

This API is used to resume the playback of a paused video.

API

```
Future<void> resume() async;
```

Parameter description

Unlimited

Returned value description

Unlimited

stop

Description

This API is used to stop a video during played back.

API

```
Future<bool> stop({bool isNeedClear = false}) async;
```

Parameter description

Parameter	Type	Description
isNeedClear	bool	Whether to clear the last-frame image.

Returned value description

Parameter	Type	Description
result	bool	Whether stop succeeded.

isPlaying

Description

This API is used to query whether the player is currently playing a video.

API

```
Future<bool> isPlaying() async;
```

Parameter description

Unlimited

Returned value description

Parameter	Type	Description
isPlaying	bool	Whether playback is ongoing.

setMute

Description

This API is used to set whether to mute the current playback.

API

```
Future<void> setMute(bool mute) async;
```

Parameter description

Parameter	Type	Description
mute	bool	Whether to mute the playback.

Returned value description

Unlimited

setVolume

Description

This API is used to set the video volume level.

API

```
Future<void> setVolume(int volume);
```

Parameter description

Parameter	Type	Description
volume	int	Video volume level. Value range: 0–100

Returned value description

Unlimited

setLiveMode

Description

This API is used to set the live streaming mode.

API

```
Future<void> setLiveMode(TXPlayerLiveMode mode) async;
```

Parameter description

Parameter	Type	Description
mode	int	Live streaming mode, which can be set to auto, expedited, or smooth mode.

Returned value description

Unlimited

setAppID

Description

This API is used to set the `appID` for cloud-based control.

API

```
Future<void> setAppID(int appId) async;
```

Parameter description

Parameter	Type	Description
appId	int	The application ID.

Returned value description

Unlimited

enableHardwareDecode

Description

This API is used to enable/disable playback based on hardware decoding. After the value is set, it will not take effect until the video playback is restarted.

API

```
Future<bool> enableHardwareDecode(bool enable);
```

Parameter description

Parameter	Type	Description
enable	bool	Whether to enable hardware decoding.

Returned value description

Parameter	Type	Description
result	bool	The hardware/software decoding setting result.

enterPictureInPictureMode

Description

This API is used to enter the PiP mode. It is applicable only to Android. Currently, live streaming on iOS doesn't support the PiP mode.

API

```
Future<int> enterPictureInPictureMode({String? backIconForAndroid, String? playIcon
```

Parameter description

The parameters are applicable only to Android.

Parameter	Type	Description
backIcon	String	The seek backward icon, which can be up to 1 MB in size as limited by Android. It is optional, and if it is not set, the system icon will be used.
playIcon	String	The playback icon, which can be up to 1 MB in size as limited by Android. It is optional, and if it is not set, the system icon will be used.
pauseIcon	String	The pause icon, which can be up to 1 MB in size as limited by Android. It is optional,

		and if it is not set, the system icon will be used.
forwardIcon	String	The fast forward icon, which can be up to 1 MB in size as limited by Android. It is optional, and if it is not set, the system icon will be used.

Returned value description

Parameter	Code	Description
NO_ERROR	0	Started successfully with no errors.
ERROR_PIP_LOWER_VERSION	-101	The Android version is too early and doesn't support the PiP mode.
ERROR_PIP_DENIED_PERMISSION	-102	The PiP mode permission wasn't enabled, or the current device doesn't support PiP.
ERROR_PIP_ACTIVITY_DESTROYED	-103	The current UI was terminated.
ERROR_IOS_PIP_DEVICE_NOT_SUPPORT	-104	The device model or system version doesn't support PiP (only supported on iPadOS 9+ and iOS 14+). This error is applicable only to iOS.
ERROR_IOS_PIP_PLAYER_NOT_SUPPORT	-105	The player doesn't support PiP. This error is applicable only to iOS.
ERROR_IOS_PIP_VIDEO_NOT_SUPPORT	-106	The video doesn't support PiP. This error is applicable only to iOS.
ERROR_IOS_PIP_IS_NOT_POSSIBLE	-107	The PiP controller was unavailable. This error is applicable only to iOS.
ERROR_IOS_PIP_FROM_SYSTEM	-108	The PiP controller reported an error. This error is applicable only to iOS.
ERROR_IOS_PIP_PLAYER_NOT_EXIST	-109	The player object doesn’t exist. This error is applicable only to iOS.
ERROR_IOS_PIP_IS_RUNNING	-110	The PiP feature was running. This error is applicable only to iOS.
ERROR_IOS_PIP_NOT_RUNNING	-111	The PiP feature didn't start. This error is applicable only to iOS.

dispose

Description

This API is used to terminate the controller. After it is called, all notification events will be terminated, and the player will be released.

API

```
Future<void> dispose() async;
```

Parameter description

Unlimited

Returned value description

Unlimited

switchStream

Description

This API is used to switch the stream played back.

API

```
Future<int> switchStream(String url) async;
```

Parameter description

Parameter	Type	Description
url	String	The video source to be switched to.

Returned value description

Parameter	Type	Description
result	int	The switch result.

enableReceiveSeiMessage

Description

Enable reception of SEI messages.

API

```
Future<int> enableReceiveSeiMessage(bool isEnabled, int payloadType) async
```

Parameter description

Parameter	Type	Description
isEnabled	bool	Whether to enable reception of SEI messages, default is off.

payloadType	int	Specify the payloadType for receiving SEI messages, supporting 5, 242, 243. Please ensure it matches the payloadType of the sender.

Returned value description

Parameter	Type	Description
result	int	The result of enabling.

showDebugView

Description

Display debug information layer.

API

```
Future<void> showDebugView(bool isShow) async
```

Parameter description

Parameter	Type	Description
isShow	bool	Whether to display the debug information layer.

setProperty

Description

Call the advanced API interface of V2TXLivePlayer.

API

```
Future<int> setProperty(String key, Object value) async
```

Parameter description

Parameter	Type	Description
key	String	Extended parameter key-value pairs.
value	Object	Extended parameter values.

Returned value description

Parameter	Type	Description
result	int	The result of the setting.

getSupportedBitrate

Description

Get stream information.

API

```
Future<List<FSteamInfo>> getSupportedBitrate() async
```

setCacheParams

Description

Set the minimum and maximum time (in seconds) for automatic adjustment of the player's cache.

API

```
Future<int> setCacheParams(double minTime, double maxTime) async
```

Parameter description

Parameter	Type	Description
minTime	double	Minimum time for automatic cache adjustment.
maxTime	double	Maximum time for automatic cache adjustment.

Returned value description

Parameter	Type	Description
result	int	The result of the setting.

setPlayerView

Description

Bind Video Rendering Texture.

API

```
Future<void>setPlayerView(int renderViewId) async
```

Parameter description

Parameter	Type	Description
renderViewId	int	The viewId returned by the onRenderViewCreatedListener callback of TXPlayerVideo.

setRenderMode

Description

Set the screen tiling mode.

API

```
Future<void> setRenderMode(FTXPlayerRenderMode renderMode) async
```

Parameter description

Parameter	Type	Description
renderMode	FTXPlayerRenderMode	Set the screen tiling mode: one is `FTXPlayerRenderMode.ADJUST_RESOLUTION` that prioritizes displaying the complete video frame while maintaining the aspect ratio, and the other is `FTXPlayerRenderMode.FULL_FILL_CONTAINER` that fills the container while preserving the aspect ratio.

`FTXLivePlayConfig` Class

Attribute configuration description

Parameter	Type	Description
maxAutoAdjustCacheTime	double	The maximum time for automatic cache time adjustment in seconds. The value must be greater than `0` . Default value: `5` .
minAutoAdjustCacheTime	double	The minimum time for automatic cache time adjustment in seconds. The value must be greater than `0` . Default value: `1` .
connectRetryCount	int	The default number of times the SDK tries reconnecting when the player is disconnected. Value range: 1–10. Default value: `3`
connectRetryInterval	int	The interval between network reconnections in seconds. Value range: 3–30. Default value: `3`

`TXVodDownloadController` Class

startPreLoad

Description

Start preloading. Before starting preloading, please set the cache directory `[SuperPlayerPlugin.setGlobalCacheFolderPath]` and cache size `[SuperPlayerPlugin.setGlobalMaxCacheSize]` of the playback engine. This setting is a global configuration and needs to be consistent with the player to avoid invalidating the playback cache.

API

```
Future<int> startPreLoad(
final String playUrl,
final int preloadSizeMB,
final int preferredResolution, {
FTXPredownloadOnCompleteListener? onCompleteListener,
    FTXPredownloadOnErrorListener? onErrorListener,
}) async
```

```
Future<void> startPreload(TXPlayInfoParams txPlayInfoParams,
    final int preloadSizeMB,
    final int preferredResolution, {
      FTXPredownloadOnCompleteListener? onCompleteListener,
      FTXPredownloadOnErrorListener? onErrorListener,
      FTXPredownloadOnStartListener? onStartListener,
    }) async
```

Parameter description

Parameter	Type	Description
playUrl	String	The URL to be preloaded
preloadSizeMB	int	Size of preloaded data (unit: MB).
preferredResolution	int	Expected resolution, with the value in the format of height x width. For example, 720x1080. If multiple resolutions are not supported or not specified, pass -1.
onCompleteListener	FTXPredownloadOnCompleteListener?	Preloading successful callback, global.
onErrorListener	FTXPredownloadOnErrorListener	Preloading failed callback, global.

TXPlayInfoParams:

Parameter	Type	Description
appId	int	The application's appId, which is required.
fileId	String	The file ID, which is required.

url	String	Only one of the video url and fileId needs to be filled in. If both are filled in, the url takes priority.
sign	String	Hotlink protection signature, see Hotlink protection product documentation.

Returned value description

Parameter	Type	Description
taskId	int	task ID

stopPreLoad

Description

Stop preloading.

API

```
Future<void> stopPreLoad(final int taskId) async
```

Parameter description

Parameter	Type	Description
taskId	int	task ID

Returned value description

Unlimited

startDownload

Description

Start downloading the video.

API

```
Future<void> startDownload(TXVodDownloadMediaInfo mediaInfo) async
```

Parameter description

Parameter	Type	Description
mediaInfo	TXVodDownloadMediaInfo	Download task information.

TXVodDownloadMediaInfo

Parameter	Type	Description

playPath	String?	Cache address. This value is obtained when accessing the video cache and does not need to be assigned when starting the download.
progress	double?	Cache progress. This value is obtained when accessing the video cache and does not need to be assigned when starting the download.
downloadState	int?	Cache status. This value is obtained when accessing the video cache and does not need to be assigned when starting the download.
userName	String?	Download account name used to distinguish downloads from different accounts. Pass an empty string to use "default".
duration	int?	Total duration of cached video. The unit is milliseconds on Android and seconds on iOS. This value is obtained when accessing the video cache and does not need to be assigned when starting the download.
playableDuration	int?	Cached duration of the video. The unit is milliseconds on Android and seconds on iOS. This value is obtained when accessing the video cache and does not need to be assigned when starting the download.
size	int?	Total file size, in bytes. This value is obtained when accessing the video cache and does not need to be assigned when starting the download.
downloadSize	int?	Downloaded file size, in bytes. This value is obtained when accessing the video cache and does not need to be assigned when starting the download.
url	String?	The URL of the video to be downloaded. This is a required field for downloading URLs. Nesting m3u8 and mp4 downloads is not supported.
dataSource	TXVodDownloadDataSource?	File ID information of the video to be downloaded. Either the URL or this parameter can be used, but at least one is required.
speed	int?	Download speed, in units of KBytes/second.
isResourceBroken	bool?	Whether the resource is corrupted, e.g. the resource has been deleted.

TXVodDownloadDataSource

Parameter	Type	Description
appId	int?	The App ID corresponding to the downloaded file. This is a required field.
fileId	String?	The ID of the downloaded file. This is a required field.
pSign	String?	Encryption signature. This is a required field for encrypted videos.
quality	int?	The ID of the video quality. This is a required field.
token	String?	Encryption token.
userName	String?	Download account name used to distinguish downloads from different accounts. Pass an empty string to use "default".

Returned value description

Unlimited

stopDownload

Description

Stop downloading.

API

```
Future<void> stopDownload(TXVodDownloadMediaInfo mediaInfo) async
```

Parameter description

Parameter	Type	Description
mediaInfo	TXVodDownloadMediaInfo	Task information.

Returned value description

Unlimited

setDownloadHeaders

Description

Set the request header for the download task.

API

```
Future<void> setDownloadHeaders(Map<String, String> headers) async
```

Parameter descriptions

Parameter	Type	Description
headers	Map<String, String>	request heade information

Returned value description

Unlimited

getDownloadList

Description

Get all download tasks, including completed, in-progress, and failed tasks.

API

```
Future<List<TXVodDownloadMediaInfo>> getDownloadList() async
```

Parameter descriptions

Unlimited

Returned value description

Parameter	Type	Description
mediaInfoList	List<TXVodDownloadMediaInfo>	Task list. Different users' downloads can be distinguished by comparing the user name.

getDownloadInfo

Description

Get download task information.

API

```
Future<TXVodDownloadMediaInfo> getDownloadInfo(TXVodDownloadMediaInfo mediaInfo) as
```

Parameter descriptions

Parameter	Type	Description
mediaInfo	TXVodDownloadMediaInfo	task information

Returned value description

Parameter	Type	Description
mediaInfo	TXVodDownloadMediaInfo	Cached task details.

setDownloadObserver

Description

Get download task information.

API

```
void setDownloadObserver(FTXDownloadOnStateChangeListener downloadOnStateChangeList
```

Parameter descriptions

Parameter	Type	Description
downloadOnStateChangeListener	FTXDownloadOnStateChangeListener	Task download status callback.
downloadOnErrorListener	FTXDownloadOnErrorListener	Task download error callback.

Returned value description

Unlimited

deleteDownloadMediaInfo

Description

Delete downloaded video.

API

```
Future<bool> deleteDownloadMediaInfo(TXVodDownloadMediaInfo mediaInfo) async
```

Parameter descriptions

Parameter	Type	Description
mediaInfo	TXVodDownloadMediaInfo	Cached task details.

Returned value description

Parameter	Type	Description
result	bool	Deletion result.

Player Adapter
Player Adapter for iOS

Last updated：2022-10-17 11:16:01

Player Adapter for iOS is a player plugin provided by VOD for customers who want to use a third-party or proprietary player to connect to Tencent Cloud PaaS resources. It is generally used by customers who strongly need to customize player features.

SDK Download

The Player Adapter SDK and demo for iOS can be downloaded here.

Integration Guide

Environment requirements

To configure the support for HTTP requests, you need to set `App Transport Security Settings->Allow Arbitrary Loads` to `YES` in the `info.plist` file of your project.

Component dependency

Add the `GCDWebServer` component dependency.

```
pod "GCDWebServer", "~> 3.0"
```

GCDWebServer is a lightweight HTTP server based on GCD and can be used for OS X and iOS. This library also implements extended features such as web-based file upload and WebDAV server.

Using the Player

Declare the variables and then create an instance. The main class of the player is `TXCPlayerAdapter` , and videos can be played back after it is created.

A `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 and find it. After clicking it, you can view relevant parameters in the video details on the right.

```
NSInteger appId; //// `appid` can be applied for in Tencent Cloud VOD
NSString *fileId;
// `psign` is a player signature. For more information on the signature and how to
NSString *pSign = self.pSignTextView.text;

TXCPlayerAdapter *adapter = [TXCPlayerAdapter shareAdapterWithAppId:appId];
```

Request the video information and play back the video:

```
id<ITXCPlayerAssistorProtocol> assistor = [TXCPlayerAdapter createPlayerAssistorWit
[assistor requestVideoInfo:^(id<ITXCPlayerAssistorProtocol> response, NSError *erro
    if (error){
        NSLog(@"create player assistor error : %@",error);
        [self.view makeToast:error.description duration:5.0 position:CSToastPositio
        return;
    }
    [weakSelf avplayerPlay:response]; // Play back the video
}];



- (void)avplayerPlay:(id<ITXCPlayerAssistorProtocol>)response
{
    AVPlayerViewController *playerVC = [[AVPlayerViewController alloc] init];
    self.playerVC = playerVC;
    TXCStreamingInfo *info = response.getStreamingInfo;
    AVPlayer *player = [[AVPlayer alloc] initWithURL:[NSURL URLWithString:info.play
    playerVC.player = player;
    playerVC.title = response.getVideoBasicInfo.name;
    [self.navigationController pushViewController:playerVC animated:YES];

    [player addObserver:self forKeyPath:@"status" options:NSKeyValueObservingOption
}
```

Terminate the player after use:

```
[TXCPlayerAdapter destroy];
```

SDK API Description

Initializing Adapter

This API is used to initialize an Adapter singleton.

API

```
+ (instancetype)shareAdapterWithAppId:(NSUInteger)appId;
```

Parameter description

appId: Enter the `appid` (if a subapplication is used, enter the `subappid`). |

Terminating Adapter

This API is used to terminate Adapter. It can be called after the program exits.

API

```
+ (void)destroy;
```

Creating the Player auxiliary class

An auxiliary class of the player can be used to get the playback `fileId` and process DRM encryption APIs.

API

```
+ (id<ITXCPlayerAssistorProtocol>)createPlayerAssistorWithFileId:(NSString
*)fileId
                                                          pSign:(NSString
*)pSign;
```

Parameter description

Parameter	Type	Description
fileId	String	The `fileId` of the video to be played.
pSign	String	The player signature.

Requesting video playback information

This API is used to request the stream information of the video to be played back from the Tencent Cloud VOD server.

API

```
- (void)requestVideoInfo:(ITXCRequestVideoInfoCallback)completion;
```

Parameter description

Parameter	Type	Description
completion	ITXCRequestVideoInfoCallback	Async callback function.

Terminating the Player auxiliary class

This API is used to terminate an auxiliary class. You can call it when exiting the player or switching to the next video for playback.

API

```
+ (void)destroyPlayerAssistor:(id<ITXCPlayerAssistorProtocol>)assistor;
```

Getting the basic video information

This API is used to get the video information and will take effect only after `id<ITXCPlayerAssistorProtocol>.requestVideoInfo` is called back.

API

```
- (TXCVideoBasicInfo *)getVideoBasicInfo;
```

Parameter description

The parameters of `TXCVideoBasicInfo` are as follows:

Parameter	Type	Description
name	String	Video name
size	Int	Video size in bytes
duration	Float	Video duration in seconds
description	String	Video description
coverUrl	String	Video cover

Getting video stream information

This API is used to get the video stream information list and will take effect only after `id<ITXCPlayerAssistorProtocol>.requestVideoInfo` is called back.

API

```
- (TXCStreamingInfo *)getStreamingInfo;
```

Parameter description

The parameters of `TXCStreamingInfo` are as follows:

Parameter	Type	Description
playUrl	String	Playback URL
subStreams	List	The adaptive bitrate substream information of the SubStreamInfo type.

The parameters of `TXCSubStreamInfo` are as follows:

Parameter	Type	Description
type	String	Substream type. Valid values: `video`
width	Int	The substream video width in px.
height	Int	The substream video height in px.
resolutionName	String	The name of the substream video displayed in the player.

Getting keyframe timestamp information

This API is used to get the video keyframe timestamp information and will take effect only after `id<ITXCPlayerAssistorProtocol>.requestVideoInfo` is called back.

API

```
- (NSArray<TXCKeyFrameDescInfo *> *)getKeyFrameDescInfos;
```

Parameter description

The parameters of `TXCKeyFrameDescInfo` are as follows:

Parameter	Type	Description
timeOffset	Float	1.1
content	String	"Beginning now..."

Getting thumbnail information

This API is used to get the thumbnail information and will take effect only after `id<ITXCPlayerAssistorProtocol>.requestVideoInfo` is called back.

API

```
- (TXCImageSpriteInfo *)getImageSpriteInfo;
```

Parameter description

The parameters of `TCXImageSpriteInfo` are as follows:

Parameter	Type	Description
imageUrls	List	Array of thumbnail download URLs of `String` type.
webVttUrl	String	Thumbnail VTT file download URL.

Player Adapter for Android

Last updated：2022-10-17 11:16:01

Player Adapter for Android is a player plugin provided by VOD for customers who want to use a third-party or proprietary player to connect to Tencent Cloud PaaS resources. It is generally used by customers who strongly need to customize player features.

SDK Download

The Player Adapter SDK and demo for Android can be downloaded here.

Integration Guide

Integrating the SDK

Integrate the SDK, copy `TXCPlayerAdapter-release-1.0.0.aar` to the `libs` directory, and add dependencies:

```
implementation(name:'TXCPlayerAdapter-release-1.0.0', ext:'aar')
```

Add the script for obfuscation:

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

Using the Player

Declare the variables and then create an instance. The main class of the player is `ITXCPlayerAssistor`, and videos can be played back after it is created.

A `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 and find it. After clicking it, you can view relevant parameters in the video details on the right.

```
// `psign` is a player signature. For more information on the signature and how to
private String mFileId, mPSign;
ITXCPlayerAssistor mPlayerAssistor = TXCPlayerAdapter.createPlayerAssistor(mFileId,
```

Initialization:

```
// Initialize the component
TXCPlayerAdapter.init(appId); // `appid` can be applied for in Tencent Cloud VOD
TXCPlayerAdapter.setLogEnable(true); // Enable log

mSuperPlayerView = findViewById(R.id.sv_videoplayer);
mPlayerAssistor = TXCPlayerAdapter.createPlayerAssistor(mFileId, mPSign);
```

Request the video information and play back the video:

```
mPlayerAssistor.requestVideoInfo(new ITXCRequestVideoInfoCallback() {

    @Override
    public void onError(int errCode, String msg) {
        Log.d(TAG, "onError msg = " + msg);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                Toast.makeText(VideoActivity.this, "onError msg = " + msg, Toast.LE
            }
        });
    }

    @Override
    public void onSuccess() {
        Log.d(TAG, "onSuccess");
        TXCStreamingInfo streamingInfo = mPlayerAssistor.getStreamingInfo();
        Log.d(TAG, "streamingInfo = " + streamingInfo);
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                if (mPlayerAssistor.getStreamingInfo() != null) {
                    // Play back the video
                    mSuperPlayerView.play(mPlayerAssistor.getStreamingInfo().playUr
                } else {
                    Toast.makeText(VideoActivity.this, "streamInfo = null", Toast.L
                }
            }
        });
    }
});
```

Terminate the player after use

```
TXCPlayerAdapter.destroy();
```

SDK API Description

Initializing TXCPlayerAdatper

This API is used to initialize Adapter each time.

API

```
TXCPlayerAdapter.init(String appId);
```

Parameter description

appId: Enter the `appid` (if a subapplication is used, enter the `subappid`). |

Terminating TXCPlayerAdatper

This API is used to terminate Adapter. It can be called after the program exits.

API

```
TXCPlayerAdapter.destroy();
```

Creating the Player auxiliary class

An auxiliary class of the player can be used to get the playback `fileId` and process DRM encryption APIs.

API

```
ITXCPlayerAssistor playerAssistor = TXCPlayerAdapter.createPlayerAssistor(String fi
```

Parameter description

Parameter	Type	Description
fileId	String	The `fileId` of the video to be played. back.
pSign	String	The player signature.

Terminating the Player auxiliary class

This API is used to terminate an auxiliary class. You can call it when exiting the player or switching to the next video for playback.

API

```
TXCPlayerAdapter.destroyPlayerAssistor(ITXCPlayerAssistor assistor);
```

Requesting video playback information

This API is used to request the stream information of the video to be played back from the Tencent Cloud VOD server.

API

```
playerAssistor.requestVideoInfo(ITXCRequestVideoInfoCallback callback);
```

Parameter description

Parameter	Type	Description
callback	ITXCRequestVideoInfoCallback	Async callback function.

Getting the basic video information

This API is used to get the video information and will take effect only after `playerAssistor.requestPlayInfo` is called back.

API

```
TXCVideoBasicInfo playerAssistor.getVideoBasicInfo();
```

Parameter description

The parameters of `TXCVideoBasicInfo` are as follows:

Parameter	Type	Description
name	String	The video name.
duration	Float	The video duration in seconds.
description	String	The video description.
coverUrl	String	The video thumbnail.

Getting video stream information

This API is used to get the video stream information list and will take effect only after `playerAssistor.requestPlayInfo` is called back.

API

```
TXCStreamingInfo playerAssistor.getStreamimgInfo();
```

Parameter description

TXCStreamingInfo

Parameter	Type	Description
playUrl	String	The playback URL.
subStreams	List	The adaptive bitrate substream information of the SubStreamInfo type.

The parameters of `SubStreamInfo` are as follows:

Parameter	Type	Description
type	String	Substream type. Valid values: `video`
width	Int	The substream video width in px.
height	Int	The substream video height in px.
resolutionName	String	The specification name of the substream video displayed in the player.

Getting keyframe timestamp information

This API is used to get the video keyframe timestamp information and will take effect only after `playerAssistor.requestPlayInfo` is called back.

API

```
List<TXCKeyFrameDescInfo> playerAssistor.getKeyFrameDescInfo();
```

Parameter description

The parameters of `TXCKeyFrameDescInfo` are as follows:

Parameter	Type	Description
timeOffset	Float	1.1
content	String	"Beginning now..."

Getting thumbnail information

This API is used to get the thumbnail information and will take effect only after `playerAssistor.requestPlayInfo` is called back.

API

```
TXCImageSpriteInfo playerAssistor.getImageSpriteInfo();
```

Parameter description

The parameters of `TCXImageSpriteInfo` are as follows:

Parameter	Type	Description
imageUrls	List	Array of thumbnail download URLs of `String` type.
webVttUrl	String	Thumbnail VTT file download URL.

Player Adapter for Web

Last updated：2022-10-17 11:42:21

This document describes the Player Adapter for web, which helps you quickly integrate third-party players with VOD capabilities through flexible APIs to implement video playback. It supports the acquisition of basic video information, video stream information, and keyframe and thumbnail information as well as private encryption. This document is intended for developers with a basic knowledge of JavaScript.

SDK Integration

Player Adapter for web supports **integration via CDN** and **npm**.

Integration via CDN

Import the initialization script to the page where videos need to be played back. The script will expose TcAdapter variables globally.

```
<script src="https://cloudcache.tencentcs.com/qcloud/video/dist/tcadapter.1.0.0.min
```

Integration via npm

```
// npm install
npm install tcadapter --save

// import TcAdapter
import TcAdapter from 'tcadapter';
```

Placing Player Container

Add the container to the page where the player needs to be displayed. TcAdapter only needs to carry the container for video playback. The playback styles and custom features can be implemented by third-party players or yourself.

```
<video id="player-container-id">
</video>
```

SDK Use Instructions

Checking the development environment

Check whether the current environment supports TcAdapter.

```
TcAdapter.isSupported();
```

Initializing Adapter

This API is used to initialize Adapter and create an Adapter instance. The initialization process will request the video file information from the Tencent Cloud VOD server.

API

```
const adapter = new TcAdapter('player-container-id', {
  fileID: string,
  appID: string,
  psign: string,
  hlsConfig: {}
}, callback);
```

Parameter description

Parameter	Type	Description
appID	String	The `APPID` of the VOD account.
fileID	String	The `fileId` of the video to be played back.
psign	String	The player signature.
hlsConfig	HlsConfig	HLS settings. Any parameters supported by `hls.js` can be used.
callback	TcAdapterCallBack	Callback for initialization completion. Basic video information can be obtained after this method is used.

Note:

The underlying layer of TcAdapter is implemented based on `hls.js`, and it can receive any parameters supported by `hls.js` through `HlsConfig` to fine-tune playback.

Getting the basic video information

This API is used to get the video information and will take effect only after initialization.

API

```
VideoBasicInfo adapter.getVideoBasicInfo();
```

Parameter description

The parameters of `VideoBasicInfo` are as follows:

Parameter	Type	Description
name	String	The video name.
duration	Float	Video duration in seconds.
description	String	The video description.
coverUrl	String	The video thumbnail.

Getting video stream information

API

```
List<StreamingOutput> adapter.getStreamimgOutputList();
```

Parameter description

The parameters of `StreamingOutput` are as follows:

Parameter	Type	Description
drmType	String	The adaptive bitstream protection type. Valid values: `plain` (no encryption), `simpleAES` (HLS common encryption).
playUrl	String	Playback URL
subStreams	List	The adaptive bitrate substream information of the SubStreamInfo type.

The parameters of `SubStreamInfo` are as follows:

Parameter	Type	Description
type	String	Substream type. Valid values: `video`
width	Int	The substream video width in px.
height	Int	The substream video height in px.
resolutionName	String	The name of the substream video displayed in the player.

Getting keyframe timestamp information

API

```
List<KeyFrameDescInfo> adapter.getKeyFrameDescInfo();
```

Parameter description

The parameters of `KeyFrameDescInfo` are as follows:

Parameter	Type	Description
timeOffset	Float	1.1
content	String	"Beginning now..."

Getting thumbnail information

API

```
ImageSpriteInfo adapter.getImageSpriteInfo();
```

Parameter description

The parameters of `ImageSpriteInfo` are as follows:

Parameter	Type	Description
imageUrls	List	Array of thumbnail download URLs of `String` type.
webVttUrl	String	Thumbnail VTT file download URL.

Listening on events

The player can perform event listening through the objects returned by the initialization, for example:

```
const adapter = TcAdapter('player-container-id', options);
adapter.on(TcAdapter.TcAdapterEvents.Error, function(error) {
  // do something
});
```

Here, `type` is the event type. Supported events include HLS' native events and the following events. Event names can be accessed from `TcAdapter.TcAdapterEvents` :

Name	Description
LOADEDMETADATA	The corresponding video information was obtained through `playcgi` . Relevant video information can be obtained through this event callback.
HLSREADY	An HLS instance was created. At this point, you can call the various attributes and methods of the HLS instance object.
ERROR	This event will be triggered when an error occurs. You can view the specific failure cause according to the callback parameters.

Getting an HLS instance

The underlying layer of Adapter is implemented based on HLS.js. You can access an HLS instance and its attributes and methods through an Adapter instance to achieve fine control over the playback process.

```
adapter.on('hlsready', () => {
  const hls = adapter.hls;
  // ...
})
```

Note:

For more information, see hls.js.

Samples

Sample 1. Using TcAdapter in React

For the specific sample, see tcplayer/tcadapter-combine-video.

```
import { useEffect, useRef } from 'react';
import TcAdapter from 'tcadapter';

function App() {
  if (!TcAdapter.isSupported()) {
    throw new Error('current environment can not support TcAdapter');
  }

  const videoRef = useRef(null);
  useEffect(() => {
    const adapter = new TcAdapter(videoRef.current, {
      appID: '1500002611',
      fileID: '5285890813738446783',
      psign: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTUwMDAwMjYxMSwiZmls
      hlsConfig: {},
    }, () => {
      console.log('basicInfo', adapter.getVideoBasicInfo());
    });

    adapter.on(TcAdapter.TcAdapterEvents.HLSREADY, () => {
      const hls = adapter.hls;
            // ...
    })
  }, []);


  const play = () => {
```

```
    videoRef.current.play();
  }

  return (
    <div>
      <div>
        <video id="player" ref={ videoRef }></video>
      </div>
      <button onClick={play}>play</button>
    </div>
  );
}

export default App;
```

Sample 2. Combining TcAdapter with Video.js

For the specific sample, see tcplayer/tcadapter-combine-videojs.

```
// 1. Video.js playback over HLS uses `@videojs/http-streaming`, so we have develop

// src/js/index.js
import videojs from './video';
import '@videojs/http-streaming';
import './tech/tcadapter'; // Add logic
export default videojs;


// src/js/tech/tcadapter.js
import videojs from '../video.js';
import TcAdapter from 'tcadapter';

class Adapter {
  constructor(source, tech, options) {
    const el = tech.el();
    // Get parameters and initialize the instance
    const adapter = new TcAdapter(el, {
      appID: '1500002611',
      fileID: '5285890813738446783',
      psign: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTUwMDAwMjYxMSwiZmls
      hlsConfig: {},
    });
    adapter.on(TcAdapter.TcAdapterEvents.LEVEL_LOADED, this.onLevelLoaded.bind(this
  }

  dispose() {
```

```
    this.hls.destroy();
  }

  onLevelLoaded(event) {
    this._duration = event.data.details.live ? Infinity : event.data.details.totald
  }

}

let hlsTypeRE = /^application\\/(x-mpegURL|vnd\\.apple\\.mpegURL)$/i;
let hlsExtRE = /\\.m3u8/i;

let HlsSourceHandler = {
  name: 'hlsSourceHandler',
  canHandleSource: function (source) {
    // skip hls fairplay, need to use Safari resolve it.
    if (source.skipHlsJs || (source.keySystems && source.keySystems['com.apple.fps.
      return '';
    } else if (hlsTypeRE.test(source.type)) {
      return 'probably';
    } else if (hlsExtRE.test(source.src)) {
      return 'maybe';
    } else {
      return '';
    }
  },

  handleSource: function (source, tech, options) {
    if (tech.hlsProvider) {
      tech.hlsProvider.dispose();
      tech.hlsProvider = null;
    } else {
      // After automatic loading is disabled for HLS, you need to manually load res
      if (options.hlsConfig && options.hlsConfig.autoStartLoad === false) {
        tech.on('play', function () {
          if (!this.player().hasStarted()) {
            this.hlsProvider.hls.startLoad();
          }
        });
      }
    }
    tech.hlsProvider = new Adapter(source, tech, options);
    return tech.hlsProvider;
  },
  canPlayType: function (type) {
    if (hlsTypeRE.test(type)) {
      return 'probably';
```

```
    }
    return '';
  }
};

function mountHlsProvider(enforce) {
  if (TcAdapter && TcAdapter.isSupported() || !!enforce) {
    try {
      let html5Tech = videojs.getTech && videojs.getTech('Html5');
      if (html5Tech) {
        html5Tech.registerSourceHandler(HlsSourceHandler, 0);
      }
    } catch (e) {
      console.error('hls.js init failed');
    }
  } else {
    // TcAdapter is not imported, MSE is not available, or X5 kernel is disabled
  }
}
mountHlsProvider();
export default Adapter;
```

Player SDK Policy

Privacy Policy

Last updated：2025-04-25 16:33:52

For details, see PRIVACY POLICY MODULE.