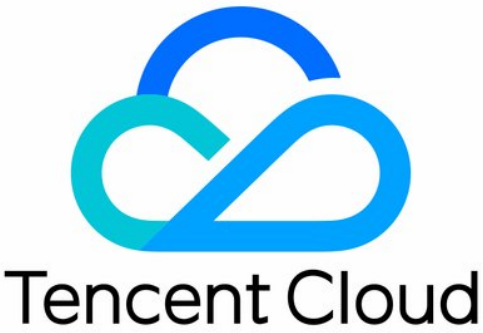
Video on Demand

Development Guide

Product Documentation



Contents

Development Guide
Media Upload
Overview

Last updated：2024-10-22 16:39:01

Media upload refers to upload of media files such as videos, audios, and thumbnail images to VOD so that they can be processed or distributed.

Upload Methods

VOD supports the following upload methods:

Local upload through console

Use the VOD console to upload local media files to VOD. This method is fast and easy. You can use this method if you are managing a small number of media files.

Pull through console

Use the VOD console to pull data from URLs. VOD will pull data offline from the URLs you specify.

Upload from server

Upload media files stored on your backend server to VOD. This method is suitable for automated and systematic production environments. VOD provides the following server upload SDKs for different programming languages:

Java SDK

C# SDK

PHP SDK

Python SDK

Node.js SDK

Golang SDK

Upload from client

Upload videos from the client to VOD. This method is suitable for UGC and PGC applications. VOD provides the following client upload SDKs:

Android upload SDK

iOS upload SDK

Web upload SDK

API upload

Use VOD's server-side API to pull media from URLs. VOD will pull data offline from the URLs you specify. This method is suitable for the migration of a large number of media files or automated data migration.

Live recording

If you use the live recording feature of CSS, the live streams recorded will be saved to VOD for archiving, editing, and replay.

Storage Regions

Supported regions

VOD has storage nodes around the globe. Your media files are uploaded to one of these storage nodes. Currently, VOD supports the following storage regions:

Region	Value
Beijing	ap-beijing
Shanghai	ap-shanghai
Guangzhou	ap-guangzhou
Chongqing	ap-chongqing
Tianjin	ap-beijing-1
Nanjing	ap-nanjing
Chengdu	ap-chengdu
Hong Kong (China)	ap-hongkong
Taipei (China)	ap-taipei
Singapore	ap-singapore
Jakarta (Indonesia)	ap-jakarta
Seoul (Korea)	ap-seoul
Bangkok (Thailand)	ap-bangkok
Tokyo (Japan)	ap-tokyo
Silicon Valley (West US)	na-siliconvalley
Virginia (East US)	na-ashburn
São Paulo (Brazil)	sa-saopaulo

Frankfurt (Germany)	eu-frankfurt

Enabling storage regions

A big advantage of having multiple storage regions is that it can improve upload performance (upload speed and success rate). This is because upload performance is affected by the distance between the upload source and the storage node. The shorter the distance, the better.
After you activate the VOD service, the **Singapore** storage region will be enabled for you automatically. You can enable other storage regions based on your actual needs. For detailed directions, see Upload Storage Settings. **Once enabled, a storage region cannot be disabled.**

Default storage region

You can have only one default region. If you have enabled only one storage region (i.e., Singapore), this region will be the default region. If you have enabled multiple storage regions, you can specify the default region in the console. For detailed directions, see Configuring Storage Regions.
The default region is given a higher priority than the others in certain scenarios. For details, see the explanation below.

Selecting a storage region

When you upload media files to VOD, by default, VOD will select a storage region automatically. You can also specify a region in your upload request.
Automatic selection:
If you have only one storage region, i.e., Singapore, all media files will be uploaded to this region.
If you have enabled multiple storage regions, VOD will select a region as follows:

Upload Method	Region Selection Policy
Local upload through console	The storage region closest to the upload source.
Pull through console	The default storage region.
Upload from server	The storage region closest to the upload source.
Upload from client	The storage region closest to the upload source.
Pull through API	The default storage region.
Live recording	The storage region closest to the live streaming source.

You can also use the methods below to specify a storage region:

Upload Method	Region Designation Method
Local upload through console	Not supported

Pull through console	Not supported
Upload from server	Java SDK C# SDK PHP SDK Python SDK Node.js SDK Go SDK
Upload from client	Using an upload signature parameter
Pull through API	Using the `StorageRegion` parameter of the API
Live recording	Not supported

Features and Limits

Media types

VOD supports the following file formats:
Video: WMV, RM, MOV, MPEG, MP4, 3GP, FLV, AVI, RMVB, TS, ASF, MPG, WEBM, MKV, M3U8, WM, ASX, RAM, MPE, VOB, DAT, MP4V, M4V, F4V, MXF, QT, and OGG.
Audio: MP3, M4A, FLAC, OGG, WAV, RA, AAC, and AMR.
Thumbnail: JPG, JPEG, PNG, GIF, BMP, TIFF, AI, CDR, EPS, and TIF.

Event notification

VOD can send you a notification after a media file is uploaded. For more information on how event notifications work and how to configure them, see Event Notification and Callback Settings.
The event notification types of different upload methods are as below:

Upload Method	Event Notification Type
Local upload through console Upload from server Upload from client Live recording	NewFileUpload
Pull through console API pull	PullComplete

Additional features

VOD offers other features related to media upload, including media management, video processing, notifications, and upload control.

Media management

Thumbnail: You can upload an image together with a video. This image will be used as the video's thumbnail in VOD.
Expiration time: You can specify the expiration time of a media file. After the file expires, VOD will automatically delete it and its associated files (transcoding outputs and screenshots).
Categorization: You can specify the category for a media file you upload.
Support for the above features by different upload methods are as follows:

Feature	Local upload through console	Pull through console	Upload from server	Upload from client	API pull	Live recording
Thumbnail	Not supported	Not supported	Java SDK C# SDK PHP SDK Python SDK Node.js SDK Go SDK	Web SDK Android SDK iOS SDK	The CoverUrl parameter of the PullUpload API.	Not supported
Expiration time	Not supported	Not supported	The ExpireTime parameter of the Java SDK upload API. The ExpireTime parameter of the C# SDK upload API. The ExpireTime parameter of the PHP SDK upload API. The ExpireTime parameter of the Python SDK upload API.	Not supported	The ExpireTime parameter of the PullUpload API	Live recording configuration.

			The ExpireTime parameter of the Node.js SDK upload API. The ExpireTime parameter of the Go SDK upload API.			
Categorization	Specifying the category	Not supported	The ClassId parameter of the Java SDK upload API. The ClassId parameter of the C# SDK upload API. The ClassId parameter of the PHP SDK upload API. The ClassId parameter of the Python SDK upload API. The ClassId parameter of the Node.js SDK upload API. The ClassId parameter of the Go SDK upload API.	The upload signature parameter classId.	The ClassId parameter of the PullUpload API.	Not supported

Video processing and notifications

Automatic video processing: You can specify a task flow for an uploaded media file. After upload, VOD will automatically execute this task flow. Common processing tasks include capturing the first video frame as the

thumbnail, transcoding, and content moderation.
Pass-through field for video processing: If automatic video processing is enabled, after the video is processed, VOD will pass through this field when sending the video processing notification.
Pass-through field for upload: After upload is completed, VOD will pass through this field when sending the upload notification.
Support for the above features by different upload methods are as follows:

Feature	Local Upload Through Console	Pull Through Console	Upload from Server	Upload from Client	Pull Through API	Live Reco
Automatic video processing	Auto-processing after upload	Not supported	Java SDK C# SDK PHP SDK Python SDK Node.js SDK Go SDK	The upload signature parameter procedure.	The procedure parameter of the PullUpload API	Not suppo
Pass-through field for video processing	Not supported	Not supported	Not supported	The upload signature parameter sessionContext.	The SessionContext parameter of the PullUpload API	Not suppo
Pass-through field for upload	Not supported	Not supported	The SourceContext parameter of the Java SDK upload API. TheSourceContext parameter of the C# SDK upload API. The SourceContext parameter of the PHP SDK upload API. The SourceContext parameter of the Python SDK upload API.	The upload signature parameter sourceContext.	Not supported	Not suppo

			The SourceContext parameter of the Node.js SDK upload API. The SourceContext parameter of the Go SDK upload API.			

Upload control

Checkpoint restart: If an upload is interrupted due to a disconnection, closing of the browser, or other reasons, when the user resumes the upload, VOD can start from where the user left off.

Pausing/Resuming upload: You can pause or resume an upload.

Canceling upload: You can cancel an upload.

Getting the upload progress: You can get the percentage of data that has already been uploaded.

Multipart upload: A media file can be segmented into multiple parts and uploaded separately. Under poor network conditions, this can reduce interruptions. In case of high-bandwidth connections, uploading multiple parts at the same time makes better use of bandwidth resources.

Support for the above features by different upload methods are as follows:

Feature	Local upload through console	Pull through console	Upload from server	Upload from client	API pull	Live recording
Checkpoint restart	Not supported	N/A	Not supported	Web SDK Android SDK iOS SDK	N/A	N/A
Pausing/Resuming upload	Not supported	N/A	Not supported	Web SDK Android SDK iOS SDK	N/A	N/A
Canceling upload	Refresh or close	N/A	Not supported	Web SDK	N/A	StopLiveRecord

	the webpage			Android SDK iOS SDK		
Getting the upload progress	Progress shown by default	Not supported	Not supported	Web SDK Android SDK iOS SDK	Not supported	N/A
Multipart upload	Enabled	N/A	Java SDK C# SDK PHP SDK Python SDK Node.js SDK Go SDK	Enabled by default for the web SDK Enabled by default for the Android SDK Enabled by default for the iOS SDK	N/A	N/A

Limits

The limits on media file size are as below:

Upload Method	Maximum File Size
Local upload through console Upload from client - web SDK	60 GB
Upload from server Pull through console Pull through API	48.82 TB (50,000 GB)
Upload from client - Android SDK Upload from client - iOS SDK	10 GB

Live recording	48.82 TB (50,000 GB) for MP4 and FLV No limit for HLS Other limits depend on your live recording settings.

There isn’t a limit on the number of files that can be uploaded.

Upload from Server Guide

Last updated：2022-05-26 12:34:32

Overview

Video upload from server refers to uploading videos to the VOD platform by the application backend. This document describes how to upload videos by using server APIs.

Prerequisites

1. Activate the service

Activate the VOD service.

2. Get TencentCloud API key

Get the security credentials (i.e., `SecretId` and `SecretKey`) required to call the server API in the following steps:
1. Log in to the console and select **Products** > **Cloud Access Management** > **API Key Management** to enter the "API Key Management" page.
2. Get the TencentCloud API key. If you have not created a key, click **Create Key** to create a pair of `SecretId` and `SecretKey`.

Directions

1. Initiate upload

An upload can be initiated through the SDK or API.

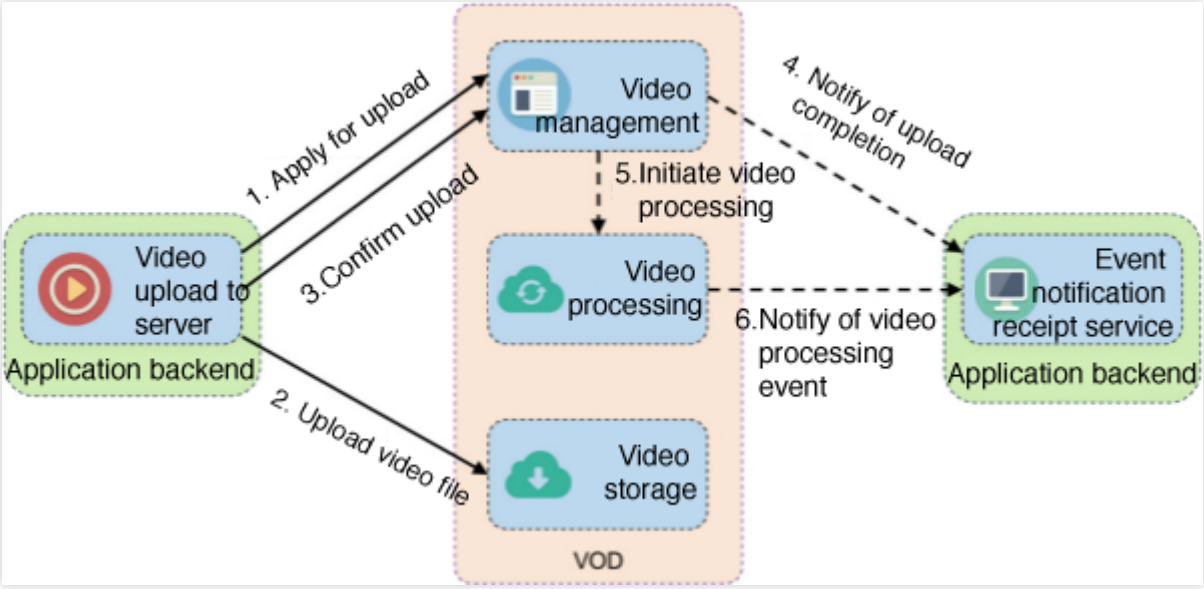
Initiating upload through SDK

To facilitate the upload feature in your development environment, VOD provides SDKs for different programming languages. The SDK for each language comes with a corresponding demo. For more information, please see:
SDK for PHP
SDK for Java
SDK for Python
SDK for Go

SDK for C#

Initiating upload through API

If the upload SDK provided by VOD does not apply to the programming language used by your application backend, the application backend needs to call VOD server APIs for video upload (this method is more complicated and not recommended). The business flow of API-based upload is as follows:



API-based upload requires you to implement steps such as applying for upload and uploading file on your own, which is not so convenient as SDK-based upload. Plus, you need to develop multipart upload logic for uploading large files. For more information, please see:

Server API - ApplyUpload

Server APIs - Uploading File

Server API - CommitUpload

Advanced features

Specify a task flow during upload

If you want to automatically initiate a video processing task flow such as transcoding and screencapturing upon video upload completion, you can specify the `Procedure` parameter when calling the server API ApplyUpload, and the parameter value should the name of the desired task flow template. VOD supports creating task flow templates and naming them. When initiating a task flow, you can use the task flow template name to indicate the desired task. All the SDKs provided by VOD for different programming languages support specifying the task flow parameter. For more information, please see:

SDK for PHP

SDK for Java
SDK for Python
SDK for Go
SDK for C#

Specify a storage region during upload

The storage region provided by VOD is "Singapore" by default. If you want to store files in another region, you need to activate it in the console. For more information, please see Upload Storage Settings. After the settings are made, the storage region can be specified by the `StorageRegion` parameter when the server API ApplyUpload is called, and the parameter value should be a region abbreviation. All the SDKs provided by VOD for different programming languages support specifying the storage region during upload. For more information, please see:

SDK for PHP
SDK for Java
SDK for Python
SDK for Go
SDK for C#

2. Event notification

After a video upload is completed, VOD will initiate an event notification - video upload completion to the application backend, through which the application backend can become aware of the video upload event. To receive event notifications, you need to go to Console - Callback Settings to enable event notification. Event Notification - Video Upload Completion mainly contains the following information:

`FileId` and URL of the uploaded video.

VOD supports specifying passthrough fields during video upload, which will be sent to the application backend upon event completion. The following fields are in the event notification:

`SourceType` : this field is always `ServerUpload` , indicating that the upload originates from a server.

`SourceContext` : this is a custom passthrough field specified by the application backend during signature distribution, which corresponds to the `sourceContext` parameter in the signature.

VOD supports automatic video processing upon video upload completion. If a video processing task flow is specified during upload, the task ID will also be included in the event notification content, i.e., the `data.procedureTaskId` field.

For more information, please see:

Task Management and Event Notification
Event Notification - Video Upload Completion

Uploading Files

Last updated：2020-03-12 16:46:02

API Description

For the API used to upload a small file (below 5 MB), please see Simple File Upload.
For the API used to upload a large file (above 5 MB), please see Initializing Multipart Upload, Uploading Parts One by One, and Ending Multipart Upload.

Feature Description

1. Upload media (and cover) files.
2. For how to upload from a client using an API, please see Overview of Upload from Client.

Via SDK

It is recommended to use the encapsulated SDK to call the API.

Via API

For usage, please see the documents in the API links above. The syntax of each API is as follows:

```
PUT <ObjectName> HTTP/1.1
Host: <BucketName>-<APPID>.cos.<Region>.myqcloud.com
Date: GMT Date
Authorization: Auth String
```

The following variables in the syntax take the values in the return result of the ApplyUpload API:
`<ObjectName>` is **MediaStoragePath** (or **CoverStoragePath** for a cover file).
`<BucketName>-<APPID>` is **StorageBucket**.
`<Region>` is **StorageRegion**.
For API requests, note the following:
For the `Authorization` signature, use `SecretId` and `SecretKey` in **TempCertificate** in the return result of the ApplyUpload API. For the calculation method, please see Request Signature.

Pass in the **x-cos-security-token** field (identifying the security token used in the request) in the HTTP header or `form-data` of the POST request packet, and assign it the value of the `Token` field in **TempCertificate**.

SDK for Java

Last updated：2022-06-24 15:52:33

VOD provides an SDK for Java for uploading videos from a server. For more information on the upload process, please see Guide.

Integration Methods

Importing Maven dependency

Add the VOD SDK dependency in the pom.xml file of your project.

```
<dependency>
    <groupId>com.qcloud</groupId>
    <artifactId>vod_api</artifactId>
    <version>2.1.4</version>
</dependency>
```

Importing jar packages

If Maven is not used for dependency management in your project, you can directly download the required jar packages and import them into the project:

jar File	Description
vod_api-2.1.4.jar	VOD SDK.
jackson-annotations-2.9.0.jar,jackson-core-2.9.7.jar,jackson-databind-2.9.7.jar,gson-2.2.4.jar	Open-source JSON libraries.
cos_api-5.4.10.jar	COS SDK.
tencentcloud-sdk-java-3.1.2.jar	TencentCloud API SDK.
commons-codec-1.10.jar,commons-logging-1.2.jar,log4j-1.2.17.jar,slf4j-api-1.7.21.jar,slf4j-log4j12-1.7.21.jar	Open-source log libraries.
httpclient-4.5.3.jar,httpcore-4.4.6.jar,okhttp-2.5.0.jar,okio-1.6.0.jar	Open-source HTTP processing libraries.
joda-time-2.9.9.jar	Open-source time processing library.
jaxb-api-2.3.0.jar	Open-source XML processing

	library.
bcprov-jdk15on-1.59.jar	Open-source encryption processing library.

Download the jar packages associated with the SDK for Java here and import them into your project.

Simple Upload

Initializing upload client object

Initialize a `VodUploadClient` instance with a TencentCloud API key.

```
VodUploadClient client = new VodUploadClient("your secretId", "your
secretKey");
```

Constructing upload request object

Set the local media upload path.

```
VodUploadRequest request = new VodUploadRequest();
request.setMediaFilePath("/data/videos/Wildlife.wmv");
```

Calling upload method

Call the upload method and pass in the access point region and upload request.

```
try {
    VodUploadResponse response = client.upload("ap-guangzhou", request);
    logger.info("Upload FileId = {}", response.getFileId());
} catch (Exception e) {
    // The business team performs troubleshooting
    logger.error("Upload Err", e);
}
```

Note:

The upload method automatically selects simple upload or multipart upload based on the file size, eliminating your need to take care of every step in multipart upload.

Advanced Features

Uploading cover

```
VodUploadClient client = new VodUploadClient("your secretId", "your
secretKey");
VodUploadRequest request = new VodUploadRequest();
request.setMediaFilePath("/data/videos/Wildlife.wmv");
request.setCoverFilePath("/data/videos/Wildlife.jpg");
try {
    VodUploadResponse response = client.upload("ap-guangzhou", request);
    logger.info("Upload FileId = {}", response.getFileId());
} catch (Exception e) {
    // The business team performs troubleshooting
    logger.error("Upload Err", e);
}
```

Specifying task flow

First, create a task flow template and name it. When initiating the task flow, you can set the `Procedure` parameter with the task flow template name, and the task flow will be executed automatically upon upload success.

```
VodUploadClient client = new VodUploadClient("your secretId", "your
secretKey");
VodUploadRequest request = new VodUploadRequest();
request.setMediaFilePath("/data/videos/Wildlife.wmv");
request.setProcedure("Your Procedure Name");
try {
    VodUploadResponse response = client.upload("ap-guangzhou", request);
    logger.info("Upload FileId = {}", response.getFileId());
} catch (Exception e) {
    // The business team performs troubleshooting
    logger.error("Upload Err", e);
}
```

Uploading to subapplication

Pass in a subapplication ID. After the upload is successful, the resource will belong only to the specified subapplication.

```
VodUploadClient client = new VodUploadClient("your secretId", "your
secretKey");
VodUploadRequest request = new VodUploadRequest();
request.setMediaFilePath("/data/videos/Wildlife.wmv");
request.setSubAppId(101);
try {
    VodUploadResponse response = client.upload("ap-guangzhou", request);
    logger.info("Upload FileId = {}", response.getFileId());
} catch (Exception e) {
```

```
    // The business team performs troubleshooting
    logger.error("Upload Err", e);
}
```

Specifying storage region

In the console, confirm that the target storage region has been activated. If not, you can do so as instructed in Upload Storage Settings and then set the abbreviation of the storage region through the `StorageRegion` attribute.

```
VodUploadClient client = new VodUploadClient("your secretId", "your
secretKey");
VodUploadRequest request = new VodUploadRequest();
request.setMediaFilePath("/data/videos/Wildlife.wmv");
request.setStorageRegion("ap-chongqing");
try {
    VodUploadResponse response = client.upload("ap-guangzhou", request);
    logger.info("Upload FileId = {}", response.getFileId());
} catch (Exception e) {
    // The business team performs troubleshooting
    logger.error("Upload Err", e);
}
```

Specifying the number of concurrent parts

The number of concurrent parts is applicable to uploading a large file in multiple parts simultaneously. The advantage of multipart upload lies in that a large file can be uploaded quickly. The SDK automatically selects simple upload or multipart upload based on the file size, eliminating your need to take care of every step in multipart upload. The number of concurrent parts of the file is specified by the `ConcurrentUploadNumber` parameter.

```
VodUploadClient client = new VodUploadClient("your secretId", "your
secretKey");
VodUploadRequest request = new VodUploadRequest();
request.setMediaFilePath("/data/videos/Wildlife.wmv");
request.setConcurrentUploadNumber(5);
try {
    VodUploadResponse response = client.upload("ap-guangzhou", request);
    logger.info("Upload FileId = {}", response.getFileId());
} catch (Exception e) {
    // The business team performs troubleshooting
    logger.error("Upload Err", e);
}
```

Uploading with temporary credentials

Pass in the relevant key information of the temporary credentials to use the temporary credentials for authentication and upload.

```
VodUploadClient client = new VodUploadClient("Credentials TmpSecretId",
"Credentials TmpSecretKey", "Credentials Token");
VodUploadRequest request = new VodUploadRequest();
request.setMediaFilePath("/data/videos/Wildlife.wmv");
try {
    VodUploadResponse response = client.upload("ap-guangzhou", request);
    logger.info("Upload FileId = {}", response.getFileId());
} catch (Exception e) {
    // The business team performs troubleshooting
    logger.error("Upload Err", e);
}
```

Setting upload proxy

Set an upload proxy, and then the protocol and data involved will be processed by the proxy. In this way, you can use the proxy to upload files to Tencent Cloud over your organization's private network.

```
VodUploadClient client = new VodUploadClient("your secretId", "your
secretKey");
VodUploadRequest request = new VodUploadRequest();
request.setMediaFilePath("/data/videos/Wildlife.wmv");
HttpProfile httpProfile = new HttpProfile();
httpProfile.setProxyHost("your proxy ip");
httpProfile.setProxyPort(8080); //your proxy port
client.setHttpProfile(httpProfile);
try {
    VodUploadResponse response = client.upload("ap-guangzhou", request);
    logger.info("Upload FileId = {}", response.getFileId());
} catch (Exception e) {
    // The business team performs troubleshooting
    logger.error("Upload Err", e);
}
```

Uploading adaptive bitstream file

The adaptive bitstream formats supported by this SDK for upload include HLS and DASH, and the media files referenced by the `manifest` (M3U8 or MPD) must be relative paths (i.e., URLs and absolute paths cannot be used) and be located in the same-level directory or subdirectory of `manifest` (i.e., `../` cannot be used). When calling the SDK's upload APIs, enter the `manifest` path as the `MediaFilePath` parameter, and the SDK will parse the list of related media files and upload them together.

```
VodUploadClient client = new VodUploadClient("your secretId", "your
secretKey");
VodUploadRequest request = new VodUploadRequest();
request.setMediaFilePath("/data/videos/prog_index.m3u8");
try {
    VodUploadResponse response = client.upload("ap-guangzhou", request);
    logger.info("Upload FileId = {}", response.getFileId());
} catch (Exception e) {
    // The business team performs troubleshooting
    logger.error("Upload Err", e);
}
```

API Description

Upload client class `VodUploadClient`

Attribute Name	Attribute Description	Type	Required
secretId	TencentCloud API key ID.	String	Yes
secretKey	TencentCloud API key.	String	Yes

Upload request class `VodUploadRequest`

Attribute Name	Attribute Description	Type	Required
MediaFilePath	Path of the media file to be uploaded, which must be a local path and does not support URLs.	String	Yes
SubAppId	ID of subapplication in VOD. If you need to access a resource in a subapplication, enter the subapplication ID in this field; otherwise, leave it empty.	Integer	No
MediaType	Type of the media file to be uploaded. For the valid values, please see Overview of media upload. If the `MediaFilePath` path contains a file extension, this parameter can be left empty.	String	No
MediaName	Name of the media file after being uploaded. If this parameter is left empty, the filename in `MediaFilePath` will be used by default.	String	No
CoverFilePath	Path of the cover file to be uploaded, which must be a local path and does not support URLs.	String	No

CoverType	Type of the cover file to be uploaded. For the valid values, please see Overview of media upload. If the `CoverFilePath` path contains a file extension, this parameter can be left empty.	String	No
Procedure	Name of the task flow to be automatically executed after upload is completed. This parameter is specified when the task flow is created through the API or console. For more information, please see Task Flow.	String	No
ExpireTime	Expiration time of media file in ISO 8601 format. For more information, please see the notes on ISO date format.	String	No
ClassId	Category ID, which is used to categorize the media for management. A category can be created, and its ID can be obtained by using the CreateClass API.	Integer	No
SourceContext	Source context of up to 250 characters, which is used to pass through the user request information and will be returned by the upload callback API.	String	No
StorageRegion	Storage region, which specifies the region where to store the file. This field should be filled in with a region abbreviation.	String	No
ConcurrentUploadNumber	Number of concurrent parts, which is valid when a large file is uploaded in multiple parts.	Integer	No

Upload response class `VodUploadResponse`

Attribute Name	Attribute Description	Type
FileId	Unique ID of media file.	String
MediaUrl	Media playback address.	String
CoverUrl	Media cover address.	String
RequestId	Unique ID of request. Each request returns a unique ID. The `RequestId` is required to troubleshoot issues.	String

Upload method `VodUploadClient.upload(String region, VodUploadRequest request)`

Parameter	Description	Type	Required

Name			
region	Access point region, i.e., the region where to request a VOD server. This is different from the storage region. For more information, please see the list of supported regions.	String	Yes
request	Upload request.	VodUploadRequest	Yes

Error Codes

Status Code	Description
InternalError	Internal error.
InvalidParameter.ExpireTime	Incorrect parameter value: expiration time.
InvalidParameterValue.CoverType	Incorrect parameter value: cover type.
InvalidParameterValue.MediaType	Incorrect parameter value: media type.
InvalidParameterValue.SubAppId	Incorrect parameter value: subapplication ID.
InvalidParameterValue.VodSessionKey	Incorrect parameter value: VOD session.
ResourceNotFound	The resource does not exist.

SDK for C#

Last updated：2022-08-03 11:06:04

VOD provides an SDK for C# for uploading videos from a server. For more information on the upload process, please see Upload Videos from Server.

Integration Steps

Installing via NuGet

1. Install on the command line:

```
dotnet add package VodSDK --version 1.0.1
```

2. Search for VodSDK in Visual Studio's NuGet package manager and install it.

Installing using source package

If your project does not have NuGet, you can directly download the source code and import it into the project:

Access from GitHub

Click here to download the SDK for C#

Download the latest code, decompress and install it in your project's working directory, and open it with Visual Studio 2017 for compiling. As this SDK also depends on external packages, the following SDKs need to be installed too:

TencentCloud API SDK

COS SDK

Simple Upload

Initializing upload client object

Initialize a `VodUploadClient` instance with a TencentCloud API key.

```
using System;
using VodSDK;

VodUploadClient client = new VodUploadClient("your secretId", "your
secretKey");
```

Constructing upload request object

Set the local media upload path.

```
VodUploadRequest request = new VodUploadRequest();
request.MediaFilePath = "/data/videos/Wildlife.wmv";
```

Calling upload

Call the upload method and pass in the access point region and upload request.

```
try
{
    VodUploadResponse response = client.Upload("ap-guangzhou", request);
    // Print the media FileId
    Console.WriteLine(response.FileId);
}
catch (Exception e)
{
    // The business team performs troubleshooting
    Console.WriteLine(e);
}
```

Note:

The upload method automatically selects simple upload or multipart upload based on the file size, eliminating your need to take care of every step in multipart upload.

Advanced Features

Uploading a cover

```
using System;
using VodSDK;

VodUploadClient client = new VodUploadClient("your secretId", "your
secretKey");
VodUploadRequest request = new VodUploadRequest();
request.MediaFilePath = "/data/videos/Wildlife.wmv";
request.CoverFilePath = "/data/videos/Wildlife.jpg";
try
{
    VodUploadResponse response = client.Upload("ap-guangzhou", request);
    // Print the media FileId
    Console.WriteLine(response.FileId);
}
catch (Exception e)
{
```

```
    // The business team performs troubleshooting
    Console.WriteLine(e);
}
```

Specifying a task flow

First, create a task flow template and name it. When initiating the task flow, you can set the `Procedure` parameter with the task flow template name, and the task flow will be executed automatically upon upload success.

```
using System;
using VodSDK;

VodUploadClient client = new VodUploadClient("your secretId", "your
secretKey");
VodUploadRequest request = new VodUploadRequest();
request.MediaFilePath = "/data/videos/Wildlife.wmv";
request.Procedure = "Your Procedure Name";
try
{
    VodUploadResponse response = client.Upload("ap-guangzhou", request);
    // Print the media FileId
    Console.WriteLine(response.FileId);
}
catch (Exception e)
{
    // The business team performs troubleshooting
    Console.WriteLine(e);
}
```

Uploading to a subapplication

Pass in a subapplication ID. After the upload is successful, the resource will belong only to the specified subapplication.

```
using System;
using VodSDK;

VodUploadClient client = new VodUploadClient("your secretId", "your
secretKey");
VodUploadRequest request = new VodUploadRequest();
request.MediaFilePath = "/data/videos/Wildlife.wmv";
request.SubAppId = 101;
try
{
    VodUploadResponse response = client.Upload("ap-guangzhou", request);
    // Print the media FileId
```

```
    Console.WriteLine(response.FileId);
}
catch (Exception e)
{
    // The business team performs troubleshooting
    Console.WriteLine(e);
}
```

Specifying a storage region

In the console, confirm that the target storage region has been activated. If not, you can do so as instructed in Upload Storage Settings and then set the abbreviation of the storage region through the `StorageRegion` attribute.

```
using System;
using VodSDK;

VodUploadClient client = new VodUploadClient("your secretId", "your
secretKey");
VodUploadRequest request = new VodUploadRequest();
request.MediaFilePath = "/data/videos/Wildlife.wmv";
request.StorageRegion = "ap-chongqing";
try
{
    VodUploadResponse response = client.Upload("ap-guangzhou", request);
    // Print the media FileId
    Console.WriteLine(response.FileId);
}
catch (Exception e)
{
    // The business team performs troubleshooting
    Console.WriteLine(e);
}
```

API Description

Upload client class `VodUploadClient`

Attribute Name	Attribute Description	Type	Required
secretId	TencentCloud API key ID.	String	Yes
secretKey	TencentCloud API key.	String	Yes

Upload request class `VodUploadRequest`

Attribute Name	Attribute Description	Type	Required
MediaFilePath	Path to the media file to be uploaded, which must be a local path and does not support URLs.	String	Yes
SubAppId	ID of a subapplication in VOD. If you need to access a resource in a subapplication, enter the subapp ID in this field; otherwise, leave it empty.	Integer	No
MediaType	Type of the media file to be uploaded. For the valid values, please see Video Upload Overview. If the `MediaFilePath` path contains a file extension, this parameter can be left empty.	String	No
MediaName	Name of the media file after being uploaded. If this parameter is left empty, the filename in `MediaFilePath` will be used by default.	String	No
CoverFilePath	Path to the cover file to be uploaded, which must be a local path and does not support URLs.	String	No
CoverType	Type of the cover file to be uploaded. For the valid values, please see Video Upload Overview. If the `CoverFilePath` path contains a file extension, this parameter can be left empty.	String	No
Procedure	Name of the task flow to be automatically executed after upload is completed. This parameter is specified when the task flow is created through the API or console. For more information, please see Task Flow Overview.	String	No
ExpireTime	Expiration time of the media file in ISO 8601 format. For more information, please see Notes on ISO Date Format.	String	No
ClassId	Category ID, which is used to categorize the media for management. A category can be created and its ID can be obtained by CreateClass.	Integer	No
SourceContext	Source context of up to 250 characters, which is used to pass through the user request information and will be returned by the upload callback API.	String	No
StorageRegion	Storage region, which specifies the region where to store the file. This field should be filled in with a region abbreviation.	String	No

Upload response class `VodUploadResponse`

Attribute Name	Attribute Description	Type

FileId	Unique ID of the media file.	String
MediaUrl	Media playback address.	String
CoverUrl	Media cover address.	String
RequestId	Unique ID of the request. Each request returns a unique ID. The RequestId is required to troubleshoot issues.	String

Upload method `VodUploadClient.Upload(String region, VodUploadRequest request)`

Parameter Name	Description	Type	Required
region	Access point region, i.e., the region where to request a VOD server. This is different from the storage region. For more information, please see the list of supported regions.	String	Yes
request	Upload request.	VodUploadRequest	Yes

Error Codes

Status Code	Description
InternalError	Internal error.
InvalidParameter.ExpireTime	Incorrect parameter value: Expiration time.
InvalidParameterValue.CoverType	Incorrect parameter value: Cover type.
InvalidParameterValue.MediaType	Incorrect parameter value: Media type.
InvalidParameterValue.SubAppId	Incorrect parameter value: Subapplicationp ID.
InvalidParameterValue.VodSessionKey	Incorrect parameter value: VOD session.
ResourceNotFound	The resource does not exist.

SDK for PHP

Last updated：2024-12-02 18:35:02

VOD provides an SDK for PHP for uploading videos from a server. For more information on the upload process, see Upload from Server Guide.

Integration Methods

Importing by using composer

```
{
    "require": {
        "qcloud/vod-sdk-v5": "v2.4.0"
    }
}
```

Installing by using source package

1. If the Composer tool is not used for dependency management in the project, you can download the source code from GitHub and import it into the project.
2. Decompress the vod-sdk.zip file into the project and import the `autoload.php` file.

Uploading Videos

Initializing upload object

Initialize a `VodUploadClient` instance with a TencentCloud API key.

Import by using composer

```
<?php
require 'vendor/autoload.php';

use Vod\\VodUploadClient;

$client = new VodUploadClient("your secretId", "your secretKey");
```

Import by using source code

```
<?php
require 'vod-sdk-v5/autoload.php';
```

```
use Vod\\VodUploadClient;

$client = new VodUploadClient("your secretId", "your secretKey");
```

Constructing upload request object

```
use Vod\\Model\\VodUploadRequest;

$req = new VodUploadRequest();
$req->MediaFilePath = "/data/videos/Wildlife.wmv";
```

Calling upload method

Call the upload method and pass in the access point region and upload request.

```
try {
    $rsp = $client->upload("ap-guangzhou", $req);
    echo "FileId -> ". $rsp->FileId . "\\n";
    echo "MediaUrl -> ". $rsp->MediaUrl . "\\n";
} catch (Exception $e) {
    // Handle upload exception
    echo $e;
}
```

Note:

The upload method automatically selects simple upload or multipart upload based on the file size, saving efforts of users.

Advanced Features

Uploading thumbnail

```
<?php
require 'vendor/autoload.php';

use Vod\\VodUploadClient;
use Vod\\Model\\VodUploadRequest;

$client = new VodUploadClient("your secretId", "your secretKey");
$req = new VodUploadRequest();
$req->MediaFilePath = "/data/videos/Wildlife.wmv";
$req->CoverFilePath = "/data/videos/Wildlife-Cover.png";
```

```
try {
    $rsp = $client->upload("ap-guangzhou", $req);
    echo "FileId -> ". $rsp->FileId . "\\n";
    echo "MediaUrl -> ". $rsp->MediaUrl . "\\n";
    echo "CoverUrl -> ". $rsp->CoverUrl . "\\n";
} catch (Exception $e) {
    // Handle upload exception
    echo $e;
}
```

Specifying task flow

First, create a task flow template and name it. When initiating the task flow, you can set the `Procedure` parameter with the task flow template name, and the task flow will be executed automatically upon upload success.

```
<?php
require 'vendor/autoload.php';

use Vod\\VodUploadClient;
use Vod\\Model\\VodUploadRequest;

$client = new VodUploadClient("your secretId", "your secretKey");
$req = new VodUploadRequest();
$req->MediaFilePath = "/data/videos/Wildlife.wmv";
$req->Procedure = "Your Procedure Name";
try {
    $rsp = $client->upload("ap-guangzhou", $req);
    echo "FileId -> ". $rsp->FileId . "\\n";
    echo "MediaUrl -> ". $rsp->MediaUrl . "\\n";
} catch (Exception $e) {
    // Handle upload exception
    echo $e;
}
```

Uploading to subapplication

Pass in a subapplication ID. After the upload is successful, the resource will belong only to the specified subapplication.

```
<?php
require 'vendor/autoload.php';

use Vod\\VodUploadClient;
use Vod\\Model\\VodUploadRequest;

$client = new VodUploadClient("your secretId", "your secretKey");
```

```
$req = new VodUploadRequest();
$req->MediaFilePath = "/data/videos/Wildlife.wmv";
$req->SubAppId = 101;
try {
    $rsp = $client->upload("ap-guangzhou", $req);
    echo "FileId -> ". $rsp->FileId . "\\n";
    echo "MediaUrl -> ". $rsp->MediaUrl . "\\n";
} catch (Exception $e) {
    // Handle upload exception
    echo $e;
}
```

Specifying storage region

In the console, confirm that the target storage region has been activated. If not, you can do so as instructed in Upload Storage Settings and then set the abbreviation of the storage region through the `StorageRegion` attribute.

```
<?php
require 'vendor/autoload.php';

use Vod\\VodUploadClient;
use Vod\\Model\\VodUploadRequest;

$client = new VodUploadClient("your secretId", "your secretKey");
$req = new VodUploadRequest();
$req->MediaFilePath = "/data/videos/Wildlife.wmv";
$req->StorageRegion = "ap-chongqing";
try {
    $rsp = $client->upload("ap-guangzhou", $req);
    echo "FileId -> ". $rsp->FileId . "\\n";
    echo "MediaUrl -> ". $rsp->MediaUrl . "\\n";
} catch (Exception $e) {
    // Handle upload exception
    echo $e;
}
```

Uploading with temporary credentials

Pass in the relevant key information of the temporary credentials to use the temporary credentials for authentication and upload.

```
<?php
require 'vendor/autoload.php';

use Vod\\VodUploadClient;
use Vod\\Model\\VodUploadRequest;
```

```
$client = new VodUploadClient("Credentials TmpSecretId", "Credentials
TmpSecretKey", "Credentials Token");
$req = new VodUploadRequest();
$req->MediaFilePath = "/data/videos/Wildlife.wmv";
try {
    $rsp = $client->upload("ap-guangzhou", $req);
    echo "FileId -> ". $rsp->FileId . "\\n";
    echo "MediaUrl -> ". $rsp->MediaUrl . "\\n";
} catch (Exception $e) {
    // Handle upload exception
    echo $e;
}
```

Setting upload proxy

Set an upload proxy, and then the protocol and data involved will be processed by the proxy. In this way, you can use the proxy to upload files to Tencent Cloud over your organization's private network.

```
<?php
require 'vendor/autoload.php';

use Vod\\VodUploadClient;
use Vod\\Model\\VodUploadRequest;
use Vod\\Model\\VodUploadHttpProfile;

$client = new VodUploadClient("your secretId", "your secretKey");
$uploadHttpProfile = new VodUploadHttpProfile("your proxy addr");
$client->setHttpProfile($uploadHttpProfile);
$req = new VodUploadRequest();
$req->MediaFilePath = "/data/videos/Wildlife.wmv";
try {
    $rsp = $client->upload("ap-guangzhou", $req);
    echo "FileId -> ". $rsp->FileId . "\\n";
    echo "MediaUrl -> ". $rsp->MediaUrl . "\\n";
} catch (Exception $e) {
    // Handle upload exception
    echo $e;
}
```

Uploading adaptive bitstream file

The adaptive bitstream formats supported by this SDK for upload include HLS and DASH, and the media files referenced by the `manifest` (M3U8 or MPD) must be relative paths (i.e., URLs and absolute paths cannot be used) and be located in the same-level directory or subdirectory of `manifest` (i.e., `../` cannot be used). When

calling the SDK's upload APIs, enter the `manifest` path as the `MediaFilePath` parameter, and the SDK will parse the list of related media files and upload them together.

```
<?php
require 'vendor/autoload.php';

use Vod\\VodUploadClient;
use Vod\\Model\\VodUploadRequest;

$client = new VodUploadClient("your secretId", "your secretKey");
$req = new VodUploadRequest();
$req->MediaFilePath = "/data/videos/prog_index.m3u8";
try {
    $rsp = $client->upload("ap-guangzhou", $req);
    echo "FileId -> ". $rsp->FileId . "\\n";
    echo "MediaUrl -> ". $rsp->MediaUrl . "\\n";
} catch (Exception $e) {
    // Handle upload exception
    echo $e;
}
```

API Description

Upload client class `VodUploadClient`

Attribute Name	Attribute Description	Type	Required
secretId	The ID of TencentCloud API key	String	Yes
secretKey	TencentCloud API key	String	Yes

Upload request class `VodUploadRequest`

Attribute Name	Attribute Description	Type	Required
MediaFilePath	Path to the media file to be uploaded, which must be a local path and does not support URLs.	String	Yes
SubAppId	ID of a subapplication in VOD. If you need to access a resource in a subapplication, enter the subapplication ID in this field; otherwise, leave it empty.	Integer	No
MediaType	Type of the media file to be uploaded. For the valid values, see Overview of media upload. If the `MediaFilePath` path	String	No

	contains a file extension, this parameter can be left empty.		
MediaName	Name of the media file after being uploaded. If this parameter is left empty, the filename in `MediaFilePath` will be used by default.	String	No
CoverFilePath	Path to the thumbnail file to be uploaded, which must be a local path and does not support URLs.	String	No
CoverType	Type of the thumbnail file to be uploaded. For the valid values, see Overview of media upload. If the `CoverFilePath` path contains a file extension, this parameter can be left empty.	String	No
Procedure	Name of the task flow to be automatically executed after upload is completed. This parameter is specified when the task flow is created through the API or console. For more information, see Task Flow.	String	No
ExpireTime	Expiration time of media file in ISO 8601 format. For more information, see notes on ISO date format.	String	No
ClassId	Category ID, which is used to categorize the media for management. A category can be created and its ID can be obtained by using the category creating API.	Integer	No
SourceContext	Source context of up to 250 characters, which is used to pass through the user request information and will be returned by the upload callback API.	String	No
StorageRegion	Storage region, which specifies the region where to store the file. This field should be filled in with a region abbreviation.	String	No

Upload response class `VodUploadResponse`

Attribute Name	Attribute Description	Type
FileId	Unique ID of the media file.	String
MediaUrl	Media playback address.	String
CoverUrl	Media thumbnail address.	String
RequestId	Unique ID of the request. Each request returns a unique ID. The `RequestId` is required to troubleshoot issues.	String

Upload method `VodUploadClient.upload(String region, VodUploadRequest request)`

Parameter	Description	Type	Required
region	Access point region, i.e., the region where to request a VOD server. This is different from the storage region. For more information, see the list of supported regions.	String	Yes
request	Upload request.	VodUploadRequest	Yes

Error Codes

Status Code	Description
InternalError	Internal error.
InvalidParameter.ExpireTime	Incorrect parameter value: Expiration time.
InvalidParameterValue.CoverType	Incorrect parameter value: Thumbnail type.
InvalidParameterValue.MediaType	Incorrect parameter value: Media type.
InvalidParameterValue.SubAppId	Incorrect parameter value: Subapplication ID.
InvalidParameterValue.VodSessionKey	Incorrect parameter value: VOD session.
ResourceNotFound	Resource does not exist.

SDK for Python

Last updated：2022-06-24 16:07:44

VOD provides an SDK for Python for uploading videos from a server. For more information on the upload process, please see Guide.

Integration Methods

Installing by using pip

```
pip install vod-python-sdk
```

Installing through source package

If pip is not used in your project, you can directly download the source code and import it into the project:

Access from GitHub

Click here to download the SDK for Python

Download the latest code and decompress:

```
$ cd vod-python-sdk
$ python setup.py install
```

Simple Video Upload

Initializing upload object

Initialize a `VodUploadClient` instance with a TencentCloud API key.

```
from qcloud_vod.vod_upload_client import VodUploadClient

client = VodUploadClient("your secretId", "your secretKey")
```

Constructing upload request object

```
from qcloud_vod.model import VodUploadRequest

request = VodUploadRequest()
request.MediaFilePath = "/data/file/Wildlife.mp4"
```

Calling upload method

Call the upload method and pass in the access point region and upload request.

```
try:
    response = client.upload("ap-guangzhou", request)
    print(response.FileId)
    print(response.MediaUrl)
except Exception as err:
    # Handle business exception
    print(err)
```

Note:

The upload method automatically selects simple upload or multipart upload based on the file size, eliminating your need to take care of every step in multipart upload.

Advanced Features

Uploading cover

```
from qcloud_vod.vod_upload_client import VodUploadClient
from qcloud_vod.model import VodUploadRequest

client = VodUploadClient("your secretId", "your secretKey")
request = VodUploadRequest()
request.MediaFilePath = "/data/file/Wildlife.mp4"
request.CoverFilePath = "/data/file/Wildlife-Cover.png"
try:
    response = client.upload("ap-guangzhou", request)
    print(response.FileId)
    print(response.MediaUrl)
    print(response.CoverUrl)
except Exception as err:
    # Handle business exception
    print(err)
```

Specifying task flow

First, create a task flow template and name it. When initiating the task flow, you can set the `Procedure` parameter with the task flow template name, and the task flow will be executed automatically upon upload success.

```
from qcloud_vod.vod_upload_client import VodUploadClient
from qcloud_vod.model import VodUploadRequest
```

```
client = VodUploadClient("your secretId", "your secretKey")
request = VodUploadRequest()
request.MediaFilePath = "/data/file/Wildlife.mp4"
request.Procedure = "Your Procedure Name"
try:
    response = client.upload("ap-guangzhou", request)
    print(response.FileId)
    print(response.MediaUrl)
except Exception as err:
    # Handle business exception
    print(err)
```

Uploading to subapplication

Pass in a subapplication ID. After the upload is successful, the resource will belong only to the specified subapplication.

```
from qcloud_vod.vod_upload_client import VodUploadClient
from qcloud_vod.model import VodUploadRequest

client = VodUploadClient("your secretId", "your secretKey")
request = VodUploadRequest()
request.MediaFilePath = "/data/file/Wildlife.mp4"
request.SubAppId = 101
try:
    response = client.upload("ap-guangzhou", request)
    print(response.FileId)
    print(response.MediaUrl)
except Exception as err:
    # Handle business exception
    print(err)
```

Specifying storage region

In the console, confirm that the target storage region has been activated. If not, you can do so as instructed in Upload Storage Settings and then set the abbreviation of the storage region through the `StorageRegion` attribute.

```
from qcloud_vod.vod_upload_client import VodUploadClient
from qcloud_vod.model import VodUploadRequest

client = VodUploadClient("your secretId", "your secretKey")
request = VodUploadRequest()
request.MediaFilePath = "/data/file/Wildlife.mp4"
request.StorageRegion = "ap-chongqing"
try:
    response = client.upload("ap-guangzhou", request)
```

```
    print(response.FileId)
    print(response.MediaUrl)
except Exception as err:
    # Handle business exception
    print(err)
```

Specifying the number of concurrent parts

The number of concurrent parts is applicable to uploading a large file in multiple parts simultaneously. The advantage of multipart upload lies in that a large file can be uploaded quickly. The SDK automatically selects simple upload or multipart upload based on the file size, eliminating your need to take care of every step in multipart upload. The number of concurrent parts of the file is specified by the `ConcurrentUploadNumber` parameter.

```
from qcloud_vod.vod_upload_client import VodUploadClient
from qcloud_vod.model import VodUploadRequest

client = VodUploadClient("your secretId", "your secretKey")
request = VodUploadRequest()
request.MediaFilePath = "/data/file/Wildlife.mp4"
request.ConcurrentUploadNumber = 5
try:
    response = client.upload("ap-guangzhou", request)
    print(response.FileId)
    print(response.MediaUrl)
except Exception as err:
    # Handle business exception
    print(err)
```

Uploading with temporary credentials

Pass in the relevant key information of the temporary credentials to use the temporary credentials for authentication and upload.

```
from qcloud_vod.vod_upload_client import VodUploadClient
from qcloud_vod.model import VodUploadRequest

client = VodUploadClient("Credentials TmpSecretId", "Credentials TmpSecretKey",
"Credentials Token")
request = VodUploadRequest()
request.MediaFilePath = "/data/file/Wildlife.mp4"
try:
    response = client.upload("ap-guangzhou", request)
    print(response.FileId)
    print(response.MediaUrl)
except Exception as err:
```

```
    # Handle business exception
    print(err)
```

Uploading adaptive bitstream file

The adaptive bitstream formats supported by this SDK for upload include HLS and DASH, and the media files referenced by the `manifest` (M3U8 or MPD) must be relative paths (i.e., URLs and absolute paths cannot be used) and be located in the same-level directory or subdirectory of `manifest` (i.e., `../` cannot be used). When calling the SDK's upload APIs, enter the `manifest` path as the `MediaFilePath` parameter, and the SDK will parse the list of related media files and upload them together.

```
from qcloud_vod.vod_upload_client import VodUploadClient
from qcloud_vod.model import VodUploadRequest

client = VodUploadClient("your secretId", "your secretKey")
request = VodUploadRequest()
request.MediaFilePath = "/data/file/prog_index.mp4"
try:
    response = client.upload("ap-guangzhou", request)
    print(response.FileId)
    print(response.MediaUrl)
except Exception as err:
    # Handle business exception
    print(err)
```

API Description

Upload client class `VodUploadClient` :

Attribute Name	Attribute Description	Type	Required
secretId	TencentCloud API key ID.	String	Yes
secretKey	TencentCloud API key.	String	Yes

Upload request class `VodUploadRequest` :

Attribute Name	Attribute Description	Type	Required
MediaFilePath	Path of the media file to be uploaded, which must be a local path and does not support URLs.	String	Yes
SubAppId	ID of subapplication in VOD. If you need to access a	Integer	No

	resource in a subapplication, enter the subapplication ID in this field; otherwise, leave it empty.		
MediaType	Type of the media file to be uploaded. For the valid values, please see Overview of media upload. If the `MediaFilePath` path contains a file extension, this parameter can be left empty.	String	No
MediaName	Name of the media file after being uploaded. If this parameter is left empty, the filename in `MediaFilePath` will be used by default.	String	No
CoverFilePath	Path of the cover file to be uploaded, which must be a local path and does not support URLs.	String	No
CoverType	Type of the cover file to be uploaded. For the valid values, please see Overview of media upload. If the `CoverFilePath` path contains a file extension, this parameter can be left empty.	String	No
Procedure	Name of the task flow to be automatically executed after upload is completed. This parameter is specified when the task flow is created through the API or console. For more information, please see Task Flow.	String	No
ExpireTime	Expiration time of media file in ISO 8601 format. For more information, please see the notes on ISO date format.	String	No
ClassId	Category ID, which is used to categorize the media for management. A category can be created, and its ID can be obtained by using the CreateClass API.	Integer	No
SourceContext	Source context of up to 250 characters, which is used to pass through the user request information and will be returned by the upload callback API.	String	No
StorageRegion	Storage region, which specifies the region where to store the file. This field should be filled in with a region abbreviation.	String	No
ConcurrentUploadNumber	Number of concurrent parts, which is valid when a large file is uploaded in multiple parts.	Integer	No

Upload response class `VodUploadResponse`

Attribute	Attribute Description	Type

Name		
FileId	Unique ID of media file.	String
MediaUrl	Media playback address.	String
CoverUrl	Media cover address.	String
RequestId	Unique ID of request. Each request returns a unique ID. The `RequestId` is required to troubleshoot issues.	String

Upload method `VodUploadClient.upload(String region, VodUploadRequest request)`

Parameter Name	Description	Type	Required
region	Access point region, i.e., the region where to request a VOD server. This is different from the storage region. For more information, please see the list of supported regions.	String	Yes
request	Upload request.	VodUploadRequest	Yes

Error Codes

Status Code	Description
InternalError	Internal error.
InvalidParameter.ExpireTime	Incorrect parameter value: expiration time.
InvalidParameterValue.CoverType	Incorrect parameter value: cover type.
InvalidParameterValue.MediaType	Incorrect parameter value: media type.
InvalidParameterValue.SubAppId	Incorrect parameter value: subapplication ID.
InvalidParameterValue.VodSessionKey	Incorrect parameter value: VOD session.
ResourceNotFound	The resource does not exist.

SDK for Node.js

Last updated：2022-06-24 15:58:17

VOD provides an SDK for Node.js for uploading videos from a server. For more information on the upload process, please see Guide.

Integration Methods

Installing by using npm

```
npm i vod-node-sdk --save
```

Installing through source package

If the npm tool is not used for dependency management in the project, you can download the source code and import it into the project:

Access from GitHub

Click here to download the SDK for Node.js

Simple Video Upload

Initializing upload object

Initialize a `VodUploadClient` instance with a TencentCloud API key.

```
const { VodUploadClient, VodUploadRequest } = require('vod-node-sdk');

client = new VodUploadClient("your secretId", "your secretKey");
```

Constructing upload request object

```
let req = new VodUploadRequest();
req.MediaFilePath = "/data/file/Wildlife.mp4";
```

Calling upload method

Call the upload method, pass in the access point region and upload request, and get the returned information through the callback.

```
client.upload("ap-guangzhou", req, function (err, data) {
    if (err) {
        // Handle business exception
        console.log(err)
    } else {
        // Get information after successful upload
        console.log(data.FileId);
        console.log(data.MediaUrl);
    }
});
```

Note:

The upload method automatically selects simple upload or multipart upload based on the file size, eliminating your need to take care of every step in multipart upload.

Advanced Features

Uploading cover

```
const { VodUploadClient, VodUploadRequest } = require('vod-node-sdk');

client = new VodUploadClient("your secretId", "your secretKey");
let req = new VodUploadRequest();
req.MediaFilePath = "/data/file/Wildlife.mp4";
req.CoverFilePath = "/data/file/Wildlife-cover.png";
client.upload("ap-guangzhou", req, function (err, data) {
    if (err) {
        // Handle business exception
        console.log(err)
    } else {
        // Get information after successful upload
        console.log(data.FileId);
        console.log(data.MediaUrl);
        console.log(data.CoverUrl);
    }
});
```

Specifying task flow

First, create a task flow template and name it. When initiating the task flow, you can set the `Procedure` parameter with the task flow template name, and the task flow will be executed automatically upon upload success.

```
const { VodUploadClient, VodUploadRequest } = require('vod-node-sdk');
```

```
client = new VodUploadClient("your secretId", "your secretKey");
let req = new VodUploadRequest();
req.MediaFilePath = "/data/file/Wildlife.mp4";
req.Procedure = "Your Procedure Name";
client.upload("ap-guangzhou", req, function (err, data) {
    if (err) {
        // Handle business exception
        console.log(err)
    } else {
        // Get information after successful upload
        console.log(data.FileId);
        console.log(data.MediaUrl);
    }
});
```

Uploading to subapplication

Pass in a subapplication ID. After the upload is successful, the resource will belong only to the specified subapplication.

```
const { VodUploadClient, VodUploadRequest } = require('vod-node-sdk');

client = new VodUploadClient("your secretId", "your secretKey");
let req = new VodUploadRequest();
req.MediaFilePath = "/data/file/Wildlife.mp4";
req.SubAppId = 101;
client.upload("ap-guangzhou", req, function (err, data) {
    if (err) {
        // Handle business exception
        console.log(err)
    } else {
        // Get information after successful upload
        console.log(data.FileId);
        console.log(data.MediaUrl);
    }
});
```

Specifying storage region

In the console, confirm that the target storage region has been activated. If not, you can do so as instructed in Upload Storage Settings and then set the abbreviation of the storage region through the `StorageRegion` attribute.

```
const { VodUploadClient, VodUploadRequest } = require('vod-node-sdk');

client = new VodUploadClient("your secretId", "your secretKey");
```

```
let req = new VodUploadRequest();
req.MediaFilePath = "/data/file/Wildlife.mp4";
req.StorageRegion = "ap-chongqing";
client.upload("ap-guangzhou", req, function (err, data) {
    if (err) {
        // Handle business exception
        console.log(err)
    } else {
        // Get information after successful upload
        console.log(data.FileId);
        console.log(data.MediaUrl);
    }
});
```

Uploading with temporary credentials

Pass in the relevant key information of the temporary credentials to use the temporary credentials for authentication and upload.

```
const { VodUploadClient, VodUploadRequest } = require('vod-node-sdk');

client = new VodUploadClient("Credentials TmpSecretId", "Credentials
TmpSecretKey", "Credentials Token");
let req = new VodUploadRequest();
req.MediaFilePath = "/data/file/Wildlife.mp4";
client.upload("ap-guangzhou", req, function (err, data) {
    if (err) {
        // Handle business exception
        console.log(err)
    } else {
        // Get information after successful upload
        console.log(data.FileId);
        console.log(data.MediaUrl);
    }
});
```

Uploading adaptive bitstream file

The adaptive bitstream formats supported by this SDK for upload include HLS and DASH, and the media files referenced by the `manifest` (M3U8 or MPD) must be relative paths (i.e., URLs and absolute paths cannot be used) and be located in the same-level directory or subdirectory of `manifest` (i.e., `../` cannot be used). When calling the SDK's upload APIs, enter the `manifest` path as the `MediaFilePath` parameter, and the SDK will parse the list of related media files and upload them together.

```
const { VodUploadClient, VodUploadRequest } = require('vod-node-sdk');
```

```
client = new VodUploadClient("your secretId", "your secretKey");
let req = new VodUploadRequest();
req.MediaFilePath = "/data/file/prog_index.m3u8";
client.upload("ap-guangzhou", req, function (err, data) {
    if (err) {
        // Handle business exception
        console.log(err)
    } else {
        // Get information after successful upload
        console.log(data.FileId);
        console.log(data.MediaUrl);
    }
});
```

API Description

Upload client class `VodUploadClient`

Attribute Name	Attribute Description	Type	Required
secretId	TencentCloud API key ID.	String	Yes
secretKey	TencentCloud API key.	String	Yes

Upload request class `VodUploadRequest`

Attribute Name	Attribute Description	Type	Required
MediaFilePath	Path of the media file to be uploaded, which must be a local path (i.e., a path on your server) and does not support URLs.	String	Yes
SubAppId	ID of subapplication in VOD. If you need to access a resource in a subapplication, enter the subapplication ID in this field; otherwise, leave it empty.	Integer	No
MediaType	Type of the media file to be uploaded. For the valid values, please see Overview of media upload. If the `MediaFilePath` path contains a file extension, this parameter can be left empty.	String	No
MediaName	Name of the media file after being uploaded. If this parameter is left empty, the filename in `MediaFilePath` will be used by default.	String	No
CoverFilePath	Path of the cover file to be uploaded, which must be a local path (i.e., a path on your server) and does not support URLs.	String	No

CoverType	Type of the cover file to be uploaded. For the valid values, please see Overview of media upload. If the `CoverFilePath` path contains a file extension, this parameter can be left empty.	String	No
Procedure	Name of the task flow to be automatically executed after upload is completed. This parameter is specified when the task flow is created through the API or console. For more information, please see Task Flow.	String	No
ExpireTime	Expiration time of media file in ISO 8601 format. For more information, please see the notes on ISO date format.	String	No
ClassId	Category ID, which is used to categorize the media for management. A category can be created, and its ID can be obtained by using the CreateClass API.	Integer	No
SourceContext	Source context of up to 250 characters, which is used to pass through the user request information and will be returned by the upload callback API.	String	No
StorageRegion	Storage region, which specifies the region where to store the file. This field should be filled in with a region abbreviation.	String	No

Upload response class `VodUploadResponse`

Attribute Name	Attribute Description	Type
FileId	Unique ID of media file.	String
MediaUrl	Media playback address.	String
CoverUrl	Media cover address.	String
RequestId	Unique ID of request. Each request returns a unique ID. The `RequestId` is required to troubleshoot issues.	String

Upload method `VodUploadClient.upload(String region, VodUploadRequest request, function callback)`

Parameter Name	Description	Type	Required
region	Access point region, i.e., the region where to request a VOD server. This is different from the storage region. For more information, please see the list of supported regions.	String	Yes

request	Upload request.	VodUploadRequest	Yes
callback	Upload completion callback function.	function	Yes

Upload completion callback function `function(err, data)`

Parameter Name	Description	Type	Required
err	Error message.	Exception	Yes
data	Upload response result.	VodUploadResponse	Yes

Error Codes

Status Code	Description
InternalError	Internal error.
InvalidParameter.ExpireTime	Incorrect parameter value: expiration time.
InvalidParameterValue.CoverType	Incorrect parameter value: cover type.
InvalidParameterValue.MediaType	Incorrect parameter value: media type.
InvalidParameterValue.SubAppId	Incorrect parameter value: subapplication ID.
InvalidParameterValue.VodSessionKey	Incorrect parameter value: VOD session.
ResourceNotFound	The resource does not exist.

SDK for Go

Last updated：2022-06-24 15:40:47

VOD provides an SDK for Go for uploading videos from a server. For more information on the upload process, please see Guide.

Integration Methods

Importing by using `go get`

```
go get -u github.com/tencentcloud/tencentcloud-sdk-go
go get -u github.com/tencentyun/cos-go-sdk-v5
go get -u github.com/tencentyun/vod-go-sdk
```

Installing through source package

If you need to directly import the source code in your project, you can directly download the source code and import it into the project:

Access from GitHub

Click here to download the SDK for Go

Simple Video Upload

Initializing upload object

Initialize a `VodUploadClient` instance with a TencentCloud API key.

```
import (
    "github.com/tencentyun/vod-go-sdk"
)

client := &vod.VodUploadClient{}
client.SecretId = "your secretId"
client.SecretKey = "your secretKey"
```

Constructing upload request object

```
import (
    "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common"
```

```
)

req := vod.NewVodUploadRequest()
req.MediaFilePath = common.StringPtr("/data/video/Wildlife.mp4")
```

Calling upload method

Call the upload method and pass in the access point region and upload request.

```
rsp, err := client.Upload("ap-guangzhou", req)
if err != nil {
    fmt.Println(err)
    return
}
fmt.Println(*rsp.Response.FileId)
fmt.Println(*rsp.Response.MediaUrl)
```

Note:

The upload method automatically selects simple upload or multipart upload based on the file size, eliminating your need to take care of every step in multipart upload.

Advanced Features

Uploading cover

```
package main

import (
    "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common"
    "github.com/tencentyun/vod-go-sdk"
    "fmt"
)

func main() {
    client := &vod.VodUploadClient{}
    client.SecretId = "your secretId"
    client.SecretKey = "your secretKey"

    req := vod.NewVodUploadRequest()
    req.MediaFilePath = common.StringPtr("/data/video/Wildlife.mp4")
    req.CoverFilePath = common.StringPtr("/data/video/Wildlife-cover.png")

    rsp, err := client.Upload("ap-guangzhou", req)
    if err != nil {
```

```
        fmt.Println(err)
        return
    }
    fmt.Println(*rsp.Response.FileId)
    fmt.Println(*rsp.Response.MediaUrl)
    fmt.Println(*rsp.Response.CoverUrl)
}
```

Specifying task flow

First, create a task flow template and name it. When initiating the task flow, you can set the `Procedure` parameter with the task flow template name, and the task flow will be executed automatically upon upload success.

```
package main

import (
    "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common"
    "github.com/tencentyun/vod-go-sdk"
    "fmt"
)

func main() {
    client := &vod.VodUploadClient{}
    client.SecretId = "your secretId"
    client.SecretKey = "your secretKey"

    req := vod.NewVodUploadRequest()
    req.MediaFilePath = common.StringPtr("/data/video/Wildlife.mp4")
    req.Procedure = common.StringPtr("Your Proceducre Name")

    rsp, err := client.Upload("ap-guangzhou", req)
    if err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println(*rsp.Response.FileId)
    fmt.Println(*rsp.Response.MediaUrl)
}
```

Uploading to subapplication

Pass in a subapplication ID. After the upload is successful, the resource will belong only to the specified subapplication.

```
package main
```

```
import (
    "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common"
    "github.com/tencentyun/vod-go-sdk"
    "fmt"
)

func main() {
    client := &vod.VodUploadClient{}
    client.SecretId = "your secretId"
    client.SecretKey = "your secretKey"

    req := vod.NewVodUploadRequest()
    req.MediaFilePath = common.StringPtr("/data/video/Wildlife.mp4")
    req.SubAppId = common.Uint64Ptr(101)

    rsp, err := client.Upload("ap-guangzhou", req)
    if err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println(*rsp.Response.FileId)
    fmt.Println(*rsp.Response.MediaUrl)
}
```

Specifying storage region

In the console, confirm that the target storage region has been activated. If not, you can do so as instructed in Upload Storage Settings and then set the abbreviation of the storage region through the `StorageRegion` attribute.

```
package main

import (
    "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common"
    "github.com/tencentyun/vod-go-sdk"
    "fmt"
)

func main() {
    client := &vod.VodUploadClient{}
    client.SecretId = "your secretId"
    client.SecretKey = "your secretKey"

    req := vod.NewVodUploadRequest()
    req.MediaFilePath = common.StringPtr("/data/video/Wildlife.mp4")
    req.StorageRegion = common.StringPtr("ap-chongqing")
```

```
    rsp, err := client.Upload("ap-guangzhou", req)
    if err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println(*rsp.Response.FileId)
    fmt.Println(*rsp.Response.MediaUrl)
}
```

Specifying the number of concurrent parts

The number of concurrent parts is applicable to uploading a large file in multiple parts simultaneously. The advantage of multipart upload lies in that a large file can be uploaded quickly. The SDK automatically selects simple upload or multipart upload based on the file size, eliminating your need to take care of every step in multipart upload. The number of concurrent parts of the file is specified by the `ConcurrentUploadNumber` parameter.

```
package main

import (
    "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common"
    "github.com/tencentyun/vod-go-sdk"
    "fmt"
)

func main() {
    client := &vod.VodUploadClient{}
    client.SecretId = "your secretId"
    client.SecretKey = "your secretKey"

    req := vod.NewVodUploadRequest()
    req.MediaFilePath = common.StringPtr("/data/video/Wildlife.mp4")
    req.ConcurrentUploadNumber = common.Uint64Ptr(5)

    rsp, err := client.Upload("ap-guangzhou", req)
    if err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println(*rsp.Response.FileId)
    fmt.Println(*rsp.Response.MediaUrl)
}
```

Uploading with temporary credentials

Pass in the relevant key information of the temporary credentials to use the temporary credentials for authentication and upload.

```
package main

import (
    "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common"
    "github.com/tencentyun/vod-go-sdk"
    "fmt"
)

func main() {
    client := &vod.VodUploadClient{}
    client.SecretId = "Credentials TmpSecretId"
    client.SecretKey = "Credentials TmpSecretKey"
    client.Token = "Credentials Token"

    req := vod.NewVodUploadRequest()
    req.MediaFilePath = common.StringPtr("/data/video/Wildlife.mp4")

    rsp, err := client.Upload("ap-guangzhou", req)
    if err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println(*rsp.Response.FileId)
    fmt.Println(*rsp.Response.MediaUrl)
}
```

Setting upload proxy

Set an upload proxy, and then the protocol and data involved will be processed by the proxy. In this way, you can use the proxy to upload files to Tencent Cloud over your organization's private network.

```
package main

import (
    "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common"
    "github.com/tencentyun/vod-go-sdk"
    "fmt"
    "net/http"
    "net/url"
)

func main() {
    client := &vod.VodUploadClient{}
```

```
    client.SecretId = "your secretId"
    client.SecretKey = "your secretKey"
    proxyUrl, _ := url.Parse("your proxy url")
    client.Transport = &http.Transport{
        Proxy: http.ProxyURL(proxyUrl),
    }

    req := vod.NewVodUploadRequest()
    req.MediaFilePath = common.StringPtr("/data/video/Wildlife.mp4")

    rsp, err := client.Upload("ap-guangzhou", req)
    if err != nil {
        fmt.Println(err)
        return
    }
    fmt.Println(*rsp.Response.FileId)
    fmt.Println(*rsp.Response.MediaUrl)
}
```

Uploading adaptive bitstream file

The adaptive bitstream formats supported by this SDK for upload include HLS and DASH, and the media files referenced by the `manifest` (M3U8 or MPD) must be relative paths (i.e., URLs and absolute paths cannot be used) and be located in the same-level directory or subdirectory of `manifest` (i.e., `../` cannot be used). When calling the SDK's upload APIs, enter the `manifest` path as the `MediaFilePath` parameter, and the SDK will parse the list of related media files and upload them together.

```
package main

import (
    "github.com/tencentcloud/tencentcloud-sdk-go/tencentcloud/common"
    "github.com/tencentyun/vod-go-sdk"
    "fmt"
)

func main() {
    client := &vod.VodUploadClient{}
    client.SecretId = "your secretId"
    client.SecretKey = "your secretKey"

    req := vod.NewVodUploadRequest()
    req.MediaFilePath = common.StringPtr("/data/video/prog_index.m3u8")

    rsp, err := client.Upload("ap-guangzhou", req)
    if err != nil {
        fmt.Println(err)
```

```
            return
        }
        fmt.Println(*rsp.Response.FileId)
        fmt.Println(*rsp.Response.MediaUrl)
        fmt.Println(*rsp.Response.CoverUrl)
    }
```

API Description

Upload client class `VodUploadClient`

Attribute Name	Attribute Description	Type	Required
SecretId	TencentCloud API key ID.	String	Yes
SecretKey	TencentCloud API key.	String	Yes

Upload request class `VodUploadRequest`

Attribute Name	Attribute Description	Type	Required
MediaFilePath	Path of the media file to be uploaded, which must be a local path and does not support URLs.	String pointer	Yes
SubAppId	ID of subapplication in VOD. If you need to access a resource in a subapplication, enter the subapplication ID in this field; otherwise, leave it empty.	uint64 pointer	No
MediaType	Type of the media file to be uploaded. For the valid values, please see Overview of media upload. If the `MediaFilePath` path contains a file extension, this parameter can be left empty.	String pointer	No
MediaName	Name of the media file after being uploaded. If this parameter is left empty, the filename in `MediaFilePath` will be used by default.	String pointer	No
CoverFilePath	Path of the cover file to be uploaded, which must be a local path and does not support URLs.	String pointer	No
CoverType	Type of the cover file to be uploaded. For the valid values, please see Overview of media upload. If the `CoverFilePath` path contains a file extension, this parameter can be left empty.	String pointer	No

Procedure	Name of the task flow to be automatically executed after upload is completed. This parameter is specified when the task flow is created through the API or console. For more information, please see Task Flow.	String pointer	No
ExpireTime	Expiration time of media file in ISO 8601 format. For more information, please see the notes on ISO date format.	String pointer	No
ClassId	Category ID, which is used to categorize the media for management. A category can be created, and its ID can be obtained by using the CreateClass API.	int64 pointer	No
SourceContext	Source context of up to 250 characters, which is used to pass through the user request information and will be returned by the upload callback API.	String pointer	No
StorageRegion	Storage region, which specifies the region where to store the file. This field should be filled in with a region abbreviation.	String pointer	No
ConcurrentUploadNumber	Number of concurrent parts, which is valid when a large file is uploaded in multiple parts.	Integer	No

Upload response class `VodUploadResponse`

Attribute Name	Attribute Description	Type
Response	Upload return result information.	struct
Response.FileId	Unique ID of media file.	String pointer
Response.MediaUrl	Media playback address.	String pointer
Response.CoverUrl	Media cover address.	String pointer
Response.RequestId	Unique ID of request. Each request returns a unique ID. The `RequestId` is required to troubleshoot issues.	String pointer

Upload method `VodUploadClient.Upload(region string, request *VodUploadRequest)`

Parameter Name	Description	Type	Required

region	Access point region, i.e., the region where to request a VOD server. This is different from the storage region. For more information, please see the list of supported regions.	String	Yes
request	Upload request.	VodUploadRequest pointer	Yes

Error Codes

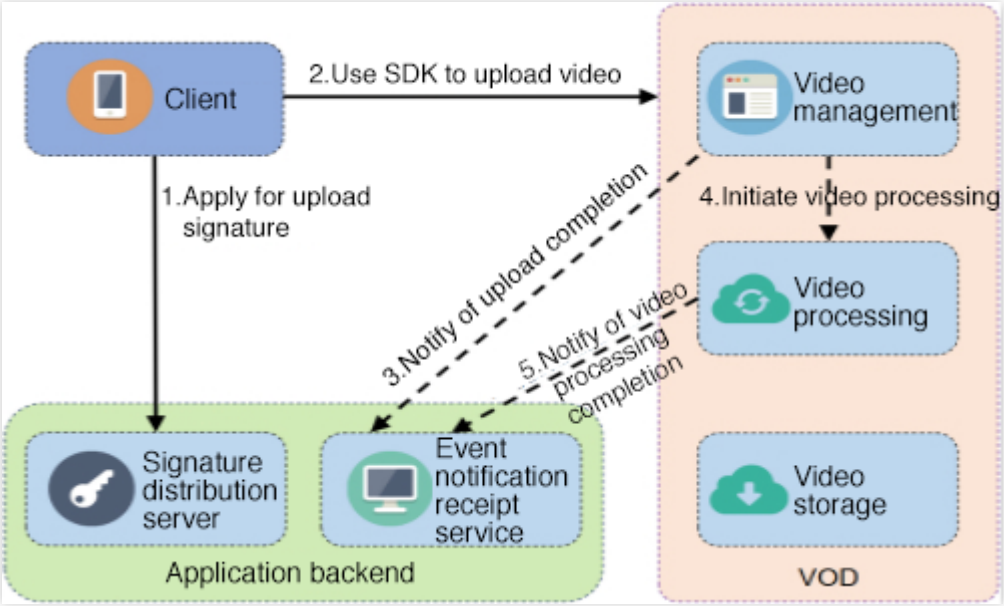
Status Code	Description
InternalError	Internal error.
InvalidParameter.ExpireTime	Incorrect parameter value: expiration time.
InvalidParameterValue.CoverType	Incorrect parameter value: cover type.
InvalidParameterValue.MediaType	Incorrect parameter value: media type.
InvalidParameterValue.SubAppId	Incorrect parameter value: subapplication ID.
InvalidParameterValue.VodSessionKey	Incorrect parameter value: VOD session.
ResourceNotFound	The resource does not exist.

Upload from Client Guide

Last updated：2022-03-24 16:00:57

Overview

Video upload from client refers to uploading local videos to the VOD platform by an end user of application. This document describes how to upload videos using a client.



Prerequisites

1. Activate the service

Activate VOD.

2. Get TencentCloud API key

Get the security credentials (i.e., `SecretId` and `SecretKey`) required to call the server API in the following steps:

1. Log in to the console and select **Products** > **Cloud Access Management** > **API Key Management** to enter the "API Key Management" page.

2. Get the TencentCloud API key. If you have not created a key, click **Create Key** to create a pair of `SecretId` and `SecretKey` .

Directions

1. Apply for upload signature

The client needs to apply to the signature distribution server of the application for an upload signature. For detailed directions, please see Signature for Upload from Client. Below are samples of generating signatures in different programming languages:

Sample of Signature in PHP

Sample of Signature in Java

Sample of Signature in Node.js

Sample of Signature in C#

Sample of Signature in Python

Note:

Upload from client is to directly upload video files from a client to the VOD platform, without the need to relay files through the application server. Therefore, VOD has to authenticate the client that initiates the request.

The application shall not disclose `SecretKey` , which has ultimate permissions, to the client in order to avoid serious security breaches. Therefore, before initiating a request, the client needs to apply for an upload signature.

2. Use the SDK to upload video

VOD provides SDKs for multiple platforms to help upload videos from client with ease. For more information, please see:

Upload SDK for Android

Upload SDK for iOS

Upload SDK for Web

Advanced features

Specify a task flow during upload

If you want to automatically initiate a video processing task flow such as transcoding and screencapturing upon video upload completion, you can set the `procedure` parameter when generating the upload signature, and the parameter value should the name of the desired task flow template. VOD supports creating task flow templates and naming them. When initiating a task flow, you can use the task flow template name to indicate the desired task.

Specify a storage region during upload

The storage region provided by VOD is "Singapore" by default. If you want to store files in another region, you need to

activate it in the console. For more information, please see Upload Storage Settings. After the settings are made, the storage region can be specified by the `storageRegion` parameter when the upload signature is generated, and the parameter value should be a region abbreviation.

Upload a video with cover

VOD allows you to upload a video with its cover by entering the path to the cover in the upload SDK API. For more information, please see:

Upload SDK for Android

Upload SDK for iOS

Upload SDK for Web

One-time signature

During video upload, the signature distributed by the application backend can be used multiple times within its validity period. If the application has high requirements for video upload security, you can use the one-time signature feature. How to use one-time signature: you just need to set `oneTimeValid` to 1 when the application backend distributes the signature. For more information, please see Signature for Upload from Client.

Note:

The one-time signature can be used only once. Though this approach is more secure, the application has to perform extra processing. For example, when upload fails, you cannot simply use the SDK to upload the video again; instead, you need to apply for a new upload signature.

Resumable upload

During the video upload process, when the upload is terminated unexpectedly, you can upload the file again from where it left off.

Note:

The effective time for resumption is 1 day, i.e., if the upload of a video is interrupted and then resumed within 1 day, it can be directly resumed; otherwise, the full video will be uploaded again by default.

You can enable the resumable upload feature for the application as shown below:

For the upload SDK for Android, set `enableResume` to `True` during upload.

For the upload SDK for iOS, set `enableResume` to `True` during upload.

For the upload SDK for Web, resumable upload is a built-in feature with no additional operation needed.

Pause/resume/cancel upload

During video upload, the VOD SDK allows you to pause, resume, or cancel upload. For more information, please see:

Upload SDK for Android

Upload SDK for iOS

Upload SDK for Web

FAQs

How do I enable automatic transcoding after video upload is completed?

You can use the `procedure` parameter in the signature for upload from client to specify the video processing method after video upload is completed. For more information, please see Specifying a Task Flow During Upload.

How does the application backend identify which client uploaded the video when it receives the video upload completion notification?

You can add the `sourceContext` parameter to the signature for upload from client to carry the user identity information. The video upload completion notification will pass this parameter to the application backend. For more information, please see Event Notification.

3. Event notification

After a video upload is completed, VOD will initiate an event notification - video upload completion to the application backend, through which the application backend can become aware of the video upload event. To receive event notifications, you need to go to Console - Callbacks to enable event notification. Event notification - video upload completion mainly contains the following information:

`FileId` and URL of the uploaded video.

VOD supports specifying passthrough fields during video upload, which will be sent to the application backend upon event completion. The following fields are in the event notification:

`SourceType` : this field is always `ServerUpload` , indicating that the upload originates from a server.

`SourceContext` : this is a custom passthrough field specified by the application backend during signature distribution, which corresponds to the `sourceContext` parameter in the signature.

VOD supports automatic video processing upon video upload completion. If a video processing task flow is specified during upload, the task ID will also be included in the event notification content, i.e., the `data.procedureTaskId` field.

For more information, please see:

Task Management and Event Notification

Event Notification - Video Upload Completion

Client Upload Acceleration

Last updated：2022-09-15 17:35:34

Based on Tencent Cloud's globally deployed acceleration network, client upload acceleration intelligently selects the optimal access point and transfer linkage based on end users' requests, increasing their upload speed and upload success rate. In addition, it supports data transfer over the QUIC protocol to improve the efficiency and stability of data transfers under poor network conditions.

Major Factors Affecting Upload Quality

Long-distance transfer

VOD deploys storage centers in many regions globally. You can enable them as needed for nearby storage during upload. For more information, see How to Increase the Speed and Success Rate of Media File Upload. However, some end users are still too far away from storage centers, and some users need to upload content across regions or even overseas. Long-distance upload means a longer network linkage and higher transfer latency. Moreover, once a problem such as network jitter and packet loss occurs at one part of the linkage, the upload speed and success rate of the entire linkage will be lowered.

Poor network conditions

Poor network conditions lead to high latency and high packet loss rate. Today, mobile networks have a wide coverage, and upload requests from mobile devices account for a large proportion of network usage. However, mobile devices often experience poor network conditions; for example, when the mobile device is in a region with poor mobile network coverage or the user frequently switches between network devices while moving around. In this case, guaranteeing stable data transfer and upload quality is a difficult challenge.

Inefficient network protocols

Most files uploaded by VOD users are large video files. However, the most frequently used network protocol for upload is still HTTP/1.1. This protocol is essentially based on the serial model and has problems such as head-of-line (HOL) blocking, which can lead to a performance bottleneck when a massive amount of data is transferred.

Acceleration Scheme for Upload from Client

Global linkage acceleration enabled by high-availability channels

To address the problem of poor upload quality due to a long network linkage in long-distance transfers, VOD provides a set of global acceleration channels based on Tencent Cloud's globally deployed acceleration network and edge

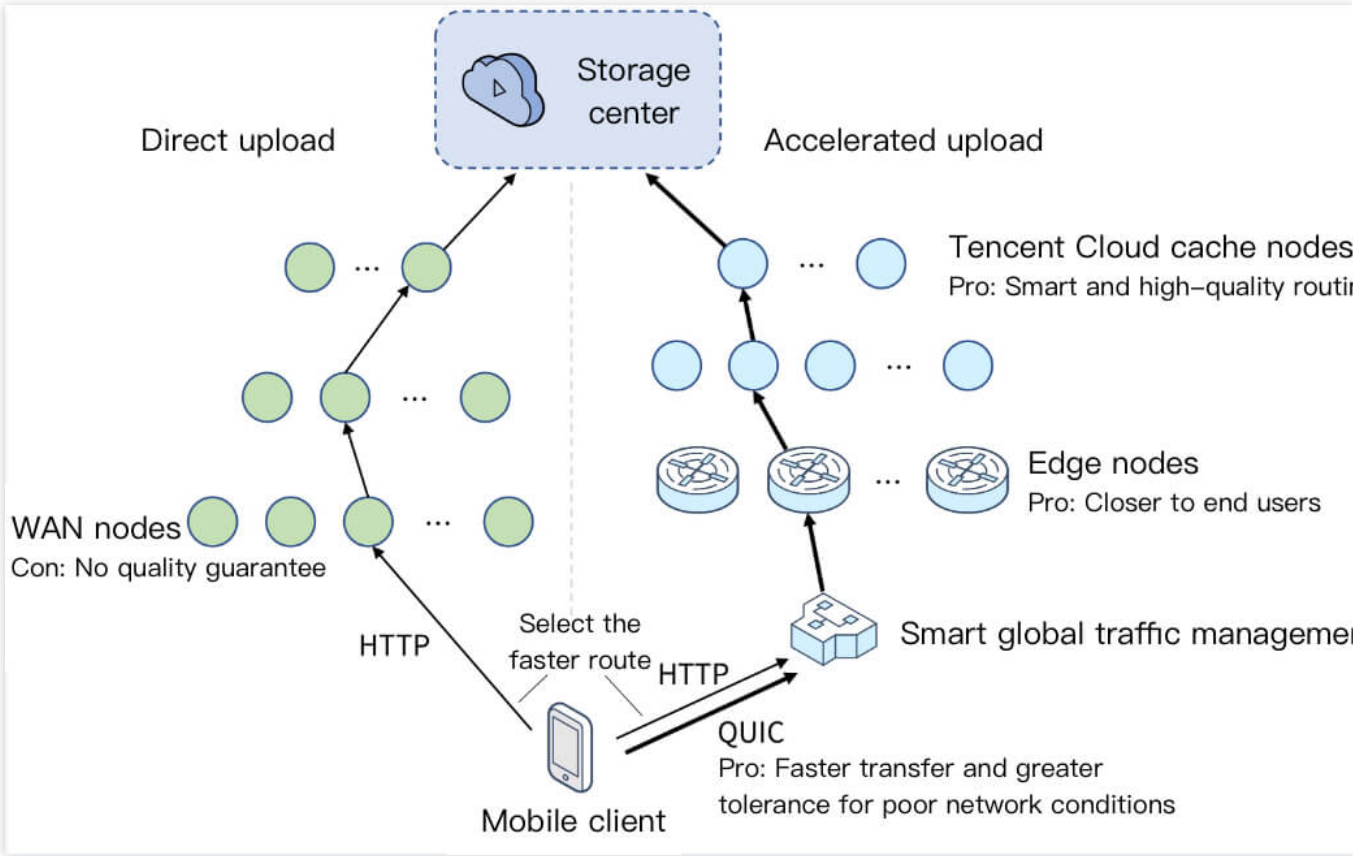
nodes. By leveraging Tencent Cloud's smart global traffic management platform, VOD sends the upload request from an end user to the edge node nearest to the user. Then, VOD selects the optimal linkage to send data to the storage center via the acceleration network, which is continuously optimized by Tencent Cloud.

Faster and more stable QUIC protocol

To help overcome poor network conditions and inefficient network protocols, VOD supports the QUIC protocol for upload from the client. The QUIC protocol is a UDP-based low-latency and high-reliability communication protocol. The current standard HTTP/3 protocol is implemented based on QUIC. QUIC supports 0-RTT connection establishment and non-HOL blocking multiplexing to transfer more data with a lower bandwidth, enabling high-quality data transfer even under poor network conditions with a high packet loss rate and network latency. It also supports connection migration to enable a smooth network switch even if the network of a mobile device is switched frequently, guaranteeing an uninterrupted network connection.

Easy-to-use smart channel selection

VOD provides an easy-to-use upload acceleration solution that can be enabled simply in the console. When you use the SDK for upload, it intelligently compares the speed of the general channel and acceleration channel and automatically selects the better channel. It also automatically detects the connection conditions and determines whether to upload the data over the QUIC protocol.



How to Use

You can enable the client upload acceleration feature with the following steps:

1. Enable **Global Linkage Acceleration** as instructed in Upload Storage Settings and enable **QUIC-based Transfer** as needed.

2. Make sure that pre-upload is called during application startup on Android or iOS. To enable **QUIC-based Transfer**, you must use the SDK for Android 9.6 or later or SDK for iOS 10.4 or later.

Note:

The SDKs for Android and iOS support both upload acceleration and QUIC-based transfer.

Currently, the SDKs for web and mini program support only upload acceleration but not QUIC-based transfer.

Billing

The client upload acceleration feature involves the following fees:

Global linkage acceleration fees: Upload acceleration traffic fees incurred while using global linkage acceleration.

QUIC-based transfer fees: Upload acceleration traffic fees incurred while using QUIC-based transfer.

For billing details, see Billing Overview.

Signature for Upload from Client

Last updated：2024-05-16 14:48:59

Before a client initiates an upload, it needs to apply to the application's signature distribution server for an upload signature which must be carried during the upload operation, so that VOD can verify whether the upload is authorized.

Signature Generation Steps

1. **Get TencentCloud API key**

Get the security credentials (i.e., `SecretId` and `SecretKey`) required to call the server API in the following steps:

2. Log in to the console and select **Products** > **Cloud Access Management** > API Key Management to enter the "API Key Management" page.

3. Get the TencentCloud API key. If you have not created a key, click **Create Key** to create a pair of `SecretId` and `SecretKey`.

4. **Splice the plaintext string** `original`

Splice the plaintext signature string `original` based on the format requirement of URL QueryString as shown below:

```
secretId=[secretId]&currentTimeStamp=[currentTimeStamp]&expireTime=
[expireTime]&random=[random]
```

Note：

`[secretId]`, `[currentTimeStamp]`, `[expireTime]`, and `[random]` in the above `original` should be replaced with actual parameter values.

`original` must contain four required parameters (`secretId`, `currentTimeStamp`, `expireTime`, and `random`) and may contain any number of optional parameters. For more information, please see Signature Parameters.

The parameter values must be URL-encoded; otherwise, `QueryString` parsing may fail.

5. **Convert the plaintext string into a signature** (with code in Java as an example)

6. Use the `SecretKey` to encrypt the plaintext string `original` with the HMAC-SHA1 algorithm to get `signatureTmp`：

```
 Mac mac = Mac.getInstance("HmacSHA1");
SecretKeySpec secretKey = new SecretKeySpec(this.secretKey.getBytes("UTF-8"), mac.g
mac.init(secretKey);
 byte[] signatureTmp = mac.doFinal(original.getBytes("UTF-8"));
```

Note：

`signatureTmp` is a byte array encoded with UTF-8 and encrypted with HMAC-SHA1.

7. Encode the plaintext string `original` into a byte array with UTF-8, merge the array with `signatureTmp`, and then Base64-encode the combination to get the signature:

```
String signature = base64Encode(byteMerger(signatureTmp, original.getBytes("utf8")))
```

Note：

`byteMerger` and `base64Encode` are methods of array merging and Base64-encoding, respectively. For more information, please see Sample Code of Signature in Java.

Example of Signature Generation

VOD also provides **sample code for signature generation** and a signature generator for your reference and verification:

Upload from client - sample code for signature generation

Upload from client - signature generator

Upload from client - signature checker

Descriptions of Signature Parameters

Parameter Name	Required	Type	Description
secretId	Yes	String	`SecretId` in the TencentCloud API key. For more information on how to get it, please see Guide for Upload from Client - Get TencentCloud API Key.
currentTimeStamp	Yes	Integer	Current Unix timestamp.
expireTime	Yes	Integer	Unix timestamp for signature expiration. `expireTime = currentTimeStamp + signature validity period` The maximum value for signature validity period is 7,776,000 (i.e., 90 days).
random	Yes	Integer	A parameter used to construct plaintext signature string. Decimal number. The maximum value is `4294967295` (2^32-1, which is the maximum value of a 32-bit unsigned binary number).
classId	No	Integer	Video file category. Default value: 0.
procedure	No	String	Subsequent task operation on a video, i.e., after a video file is

			uploaded, task flow operations will be initiated automatically. This parameter value is a task flow template name. VOD supports creating task flow templates and naming the templates.
taskPriority	No	Integer	Priority of subsequent video task (only valid if `procedure` is specified). Value range: [-10, 10]. Default value: 0.
taskNotifyMode	No	String	Notification mode for task flow status change (only valid if `procedure` is specified). Finish: an event notification will be initiated only after the task flow is completely executed. Change: an event notification will be initiated as soon as the status of a subtask in the task flow changes. None: no callback for the task flow will be accepted. Default value: Finish.
sourceContext	No	String	Source context, which is used to pass through the user request information. The upload callback API will return the value of this field. It can contain up to 250 characters.
oneTimeValid	No	Integer	Whether a signature is valid only for once. For more information, please see Guide for Upload from Client - One-time Signature. 0 (default value): not enabled; 1: enabled. For relevant error codes, please see One-time Signature Description.
vodSubAppId	No	Integer	Subapplication ID. If this parameter is left empty, `0`, or your Tencent Cloud `AppId`, the manipulated subapplication will be the "primary application".
sessionContext	No	String	Session context, which is used to pass through the user request information. If the `procedure` parameter is specified, the task flow status change callback API will return the value of this field. It can contain up to 1,000 characters.
storageRegion	No	String	Specifies the storage region. You can add storage regions in the console by yourself. For more information, please see Upload Storage Settings. This field should be filled in with a region abbreviation.

One-time signature description

After the one-time signature feature is enabled, the signature server needs to ensure that the signatures distributed to users are different each time (for example, it should be ensured that the `random` parameters in the signatures distributed at the same time are unique); otherwise, a duplicate signature error will occur.
If an upload fails due to a signature error, a new signature needs to be obtained for retry.

The error code for signature errors caused by the SDKs for Android and Java is `1001`.

Example of Signature Generation

Last updated：2021-01-27 17:43:29

Sample Signature in PHP

```
<?php
// Determine the TencentCloud API key of the application
$secret_id = "XXXXXXXXXXXXXXXXXX";
$secret_key = "AAAAAAAAAAAAAAAAAA";

// Determine the current time and expiration time of the signature
$current = time();
$expired = $current + 86400;  // Signature validity period: 1 day

// Enter parameters into the parameter list
$arg_list = array(
    "secretId" => $secret_id,
    "currentTimeStamp" => $current,
    "expireTime" => $expired,
    "random" => rand());

// Calculate the signature
$original = http_build_query($arg_list);
$signature = base64_encode(hash_hmac('SHA1', $original, $secret_key, true).$origina

echo $signature;
echo "\\n";
?>
```

Sample Signature in Java

```
import java.util.Random;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import sun.misc.BASE64Encoder;

// Signature tool class
class Signature {
    private String secretId;
    private String secretKey;
```

```
    private long currentTime;
    private int random;
    private int signValidDuration;

    private static final String HMAC_ALGORITHM = "HmacSHA1"; // Signature algorithm
    private static final String CONTENT_CHARSET = "UTF-8";

    public static byte[] byteMerger(byte[] byte1, byte[] byte2) {
        byte[] byte3 = new byte[byte1.length + byte2.length];
        System.arraycopy(byte1, 0, byte3, 0, byte1.length);
        System.arraycopy(byte2, 0, byte3, byte1.length, byte2.length);
        return byte3;
    }

    // Get the signature
    public String getUploadSignature() throws Exception {
        String strSign = "";
        String contextStr = "";

        // Generate the original parameter string
        long endTime = (currentTime + signValidDuration);
        contextStr += "secretId=" + java.net.URLEncoder.encode(secretId, "utf8");
        contextStr += "&currentTimeStamp=" + currentTime;
        contextStr += "&expireTime=" + endTime;
        contextStr += "&random=" + random;

        try {
            Mac mac = Mac.getInstance(HMAC_ALGORITHM);
            SecretKeySpec secretKey = new SecretKeySpec(this.secretKey.getBytes(CON
            mac.init(secretKey);

            byte[] hash = mac.doFinal(contextStr.getBytes(CONTENT_CHARSET));
            byte[] sigBuf = byteMerger(hash, contextStr.getBytes("utf8"));
            strSign = base64Encode(sigBuf);
            strSign = strSign.replace(" ", "").replace("\\n", "").replace("\\r", ""
        } catch (Exception e) {
            throw e;
        }
        return strSign;
    }

    private String base64Encode(byte[] buffer) {
        BASE64Encoder encoder = new BASE64Encoder();
        return encoder.encode(buffer);
    }

    public void setSecretId(String secretId) {
```

```
        this.secretId = secretId;
    }

    public void setSecretKey(String secretKey) {
        this.secretKey = secretKey;
    }

    public void setCurrentTime(long currentTime) {
        this.currentTime = currentTime;
    }

    public void setRandom(int random) {
        this.random = random;
    }

    public void setSignValidDuration(int signValidDuration) {
        this.signValidDuration = signValidDuration;
    }
}

public class Test {
    public static void main(String[] args) {
        Signature sign = new Signature();
        // Set the TencentCloud API key of the application
        sign.setSecretId("Secret ID of your API key");
        sign.setSecretKey("Secret key of your API key");
        sign.setCurrentTime(System.currentTimeMillis() / 1000);
        sign.setRandom(new Random().nextInt(java.lang.Integer.MAX_VALUE));
        sign.setSignValidDuration(3600 * 24 * 2); // Signature validity period: 2 d

        try {
            String signature = sign.getUploadSignature();
            System.out.println("signature : " + signature);
        } catch (Exception e) {
            System.out.print("Failed to get the signature");
            e.printStackTrace();
        }
    }
}
```

For Java v1.9 and above, the packages related to `sun.misc.BASE64Encoder` have been removed. You can replace the corresponding implementation in the `base64Encode` method with `java.util.Base64`. For more information, please see the following code:

```
import java.util.Base64;
```

```
private String base64Encode(byte[] buffer) {
    Base64.Encoder encoder = Base64.getEncoder();
    return encoder.encodeToString(buffer);
}
```

Sample Signature in Node.js

```
var querystring = require("querystring");
var crypto = require('crypto');

// Determine the TencentCloud API key of the application
var secret_id = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
var secret_key = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";

// Determine the current time and expiration time of the signature
var current = parseInt((new Date()).getTime() / 1000)
var expired = current + 86400;  // Signature validity period: 1 day

// Enter parameters into the parameter list
var arg_list = {
    secretId : secret_id,
    currentTimeStamp : current,
    expireTime : expired,
    random : Math.round(Math.random() * Math.pow(2, 32))
}

// Calculate the signature
var orignal = querystring.stringify(arg_list);
var orignal_buffer = new Buffer(orignal, "utf8");

var hmac = crypto.createHmac("sha1", secret_key);
var hmac_buffer = hmac.update(orignal_buffer).digest();

var signature = Buffer.concat([hmac_buffer, orignal_buffer]).toString("base64");

console.log(signature);
```

Sample Signature in C#

```
using System;
```

```
using System.Security.Cryptography;
using System.Text;
using System.Threading;

class Signature
{
    public string m_strSecId;
    public string m_strSecKey;
    public int m_iRandom;
    public long m_qwNowTime;
    public int m_iSignValidDuration;
    public static long GetIntTimeStamp()
    {
        TimeSpan ts = DateTime.UtcNow - new DateTime(1970, 1, 1);
        return Convert.ToInt64(ts.TotalSeconds);
    }
    private byte[] hash_hmac_byte(string signatureString, string secretKey)
    {
        var enc = Encoding.UTF8; HMACSHA1 hmac = new HMACSHA1(enc.GetBytes(secretKe
        hmac.Initialize();
        byte[] buffer = enc.GetBytes(signatureString);
        return hmac.ComputeHash(buffer);
    }
    public string GetUploadSignature()
    {
        string strContent = "";
        strContent += ("secretId=" + Uri.EscapeDataString((m_strSecId)));
        strContent += ("&currentTimeStamp=" + m_qwNowTime);
        strContent += ("&expireTime=" + (m_qwNowTime + m_iSignValidDuration));
        strContent += ("&random=" + m_iRandom);

        byte[] bytesSign = hash_hmac_byte(strContent, m_strSecKey);
        byte[] byteContent = System.Text.Encoding.Default.GetBytes(strContent);
        byte[] nCon = new byte[bytesSign.Length + byteContent.Length];
        bytesSign.CopyTo(nCon, 0);
        byteContent.CopyTo(nCon, bytesSign.Length);
        return Convert.ToBase64String(nCon);
    }
}
class Program
{
    static void Main(string[] args)
    {
        Signature sign = new Signature();
        sign.m_strSecId = "Secret ID of your API key";
        sign.m_strSecKey = "Secret key of your API key";
        sign.m_qwNowTime = Signature.GetIntTimeStamp();
```

```
        sign.m_iRandom = new Random().Next(0, 1000000);
        sign.m_iSignValidDuration = 3600 * 24 * 2;

        Console.WriteLine(sign.GetUploadSignature());
    }
}
```

Sample Signature in Python

```
#!/usr/local/bin/python3
#coding=utf-8

import time
import random
import hmac
import hashlib
import base64

SecretId = 'IamSecretId'
SecretKey = 'IamSecretKey'
#TimeStamp = int(time.time())
TimeStamp = 1571215095
ExpireTime = TimeStamp + 86400 * 365 * 10
#Random = random.randint(0, 999999)
Random = 220625

Original = "secretId=" + SecretId + "&currentTimeStamp=" + str(TimeStamp) + "&expir

Hmac = hmac.new(bytes(SecretKey, 'utf-8'), bytes(Original, 'utf-8'), hashlib.sha1)
Sha1 = Hmac.digest()
Signature = bytes(Sha1) + bytes(Original, 'utf-8')
Signature2 = base64.b64encode(Signature)

#return str(signature2, 'UTF-8')

print("Original: ", Original)
print("HMAC-SHA1: ", Sha1)
print("Signature before BASE64: ", Signature)
print("Signature after BASE64: ", str(Signature2))
```

Sample Signature in Go

```
package main

import (
    "crypto/hmac"
    "crypto/sha1"
    "encoding/base64"
    "fmt"
    "math/rand"
    "strconv"
    "time"
)

func generateHmacSHA1(secretToken, payloadBody string) []byte {
    mac := hmac.New(sha1.New, []byte(secretToken))
    sha1.New()
    mac.Write([]byte(payloadBody))
    return mac.Sum(nil)
}

func main() {
    rand.Seed(time.Now().Unix())
    secretId := "IamSecretId"
    secretKey := "IamSecretKey"
    // timestamp := time.Now().Unix()
    timestamp := int64(1571215095)
    expireTime := timestamp + 86400*365*10
    timestampStr := strconv.FormatInt(timestamp, 10)
    expireTimeStr := strconv.FormatInt(expireTime, 10)

    random := 220625
    randomStr := strconv.Itoa(random)
    original := "secretId=" + secretId + "&currentTimeStamp=" + timestampStr + "&ex
    signature := generateHmacSHA1(secretKey, original)
    signature = append(signature, []byte(original)...)
    signatureB64 := base64.StdEncoding.EncodeToString(signature)
    fmt.Println(signatureB64)
}
```

Upload SDK for Web

Last updated：2023-03-07 11:20:50

VOD provides an SDK for uploading files from browsers. You can download the SDK source code at GitHub.

Uploading Videos

Importing the SDK

Importing by using a script tag

If Webpack is not used, you can import the SDK using a script tag. This method will expose the global variable `TcVod`. You can choose either of the two ways below:

Download to the local file system Download the SDK source code to your local file system and use the code below to import the SDK:

```
<script src="./vod-js-sdk-v6.js"></script>
```

Note:

Change the value of `src` to the local path of the source code.

Import from CDN

Use the code below to import the SDK from a CDN:

```
<script src="https://cdn-go.cn/cdn/vod-js-sdk-v6/latest/vod-js-sdk-v6.js"></script>
```

Click here to try a demo that imported the SDK using a script tag. The source code of the demo can be found here.

Importing by using npm

If Webpack (such as Vue or React) is used, You can use npm to import the SDK:

```
// Run `npm install vod-js-sdk-v6`, and use the command below to import the SDK dir
import TcVod from 'vod-js-sdk-v6'
```

Click here to view the source code of a demo that imports the SDK using npm.

Note:

The SDK relies on promises, which you should import if your browser version is old.

Defining the function to get an upload signature

```
function getSignature() {
  return axios.post(url).then(function (response) {
```

```
    return response.data.signature;
  })
};
```

Note:

`url` is the URL of your signature distribution service. For more information, see the Guide for upload from a client.

For details on how to calculate `signature`, see Signature for Upload from Client.

The upload signature contains information such as the **subapplication ID**, **video category**, and **task flow**. For more information, see Descriptions of Signature Parameters.

Video upload example

```
// If the SDK is imported using the `import` command, run `new TcVod(opts)`.
// If the SDK is imported using a script tag, use `new TcVod.default(opts)`.
const tcVod = new TcVod.default({
  getSignature: getSignature // The function to get the upload signature
})

const uploader = tcVod.upload({
  mediaFile: mediaFile, // The media file (video, audio, or image), whose data type
})
uploader.on('media_progress', function(info) {
  console.log(info.percent) // The upload progress
})

// Callback of the result
// type doneResult = {
//   fileId: string,
//   video: {
//     url: string
//   },
//   cover: {
//     url: string
//   }
// }
uploader.done().then(function (doneResult) {
  // Deal with doneResult
}).catch(function (err) {
  // Deal with error
})
```

Note:

`opts` in `new TcVod(opts)` refers to parameters of the `TcVod` API. For details, see API Description.

The upload API automatically selects simple upload or multipart upload based on the file size. You don't need to manually set up multipart upload.
To upload to a subapplication, see Subapplication System - Upload from client.

Advanced Features

Uploading both the video and thumbnail

```
const uploader = tcVod.upload({
  mediaFile: mediaFile,
  coverFile: coverFile,
})

uploader.done().then(function (doneResult) {
  // Deal with doneResult
})
```

Getting the upload progress

The SDK can notify you of the upload progress via callbacks:

```
const uploader = tcVod.upload({
  mediaFile: mediaFile,
  coverFile: coverFile,
})
// When the video upload is completed
uploader.on('media_upload', function(info) {
  uploaderInfo.isVideoUploadSuccess = true;
})
// The video upload progress
uploader.on('media_progress', function(info) {
  uploaderInfo.progress = info.percent;
})
// When the thumbnail upload is completed
uploader.on('cover_upload', function(info) {
  uploaderInfo.isCoverUploadSuccess = true;
})
// The thumbnail upload progress
uploader.on('cover_progress', function(info) {
  uploaderInfo.coverProgress = info.percent;
})

uploader.done().then(function (doneResult) {
  // Deal with doneResult
```

```
})
```

For details about the return values of `xxx_upload` and `xxx_progress`, see Object Operations.

Canceling upload

The SDK supports canceling ongoing video or thumbnail upload:

```
const uploader = tcVod.upload({
  mediaFile: mediaFile,
  coverFile: coverFile,
})

uploader.cancel()
```

Checkpoint restart

The SDK supports automatic checkpoint restart for uploads. If an upload is interrupted unexpectedly (for example, because the browser is closed or the network is disconnected), you can continue uploading the file from where it left off.

API Description

TcVod

Parameter	Required	Type	Description
getSignature	Yes	Function	The function used to get the upload signature.
appId	No	number	If this parameter is set, it will be carried by the built-in statistical report system.
reportId	No	number	If this parameter is set, it will be carried by the built-in statistical report system.

TcVod.upload

Parameter	Required	Type	Description
mediaFile	No	File	The media file (video, audio, or image).
coverFile	No	File	The thumbnail file.
mediaName	No	string	The filename, which will overwrite the filename in the metadata.

fileId	No	string	The ID of the new thumbnail file.
reportId	No	number	If this parameter is set, it will be carried by the built-in statistical report system and will overwrite the settings in the constructor.
fileParallelLimit	No	number	The maximum number of concurrent uploads allowed in the same instance. Default value: 3.
chunkParallelLimit	No	number	The maximum number of upload parts allowed for the same file. Default value: 6.
chunkRetryTimes	No	number	The maximum number of retry attempts for multipart upload. Default value: 2 (three upload requests in total).
chunkSize	No	number	The part size (bytes) for multipart upload. Default value: 8388608 (8 MB).
progressInterval	No	number	The interval (ms) of sending the `onProgress` callback. Default value: 1000.

Events

Event Name	Required	Description
media_upload	No	The media file is successfully uploaded.
cover_upload	No	The thumbnail is successfully uploaded.
media_progress	No	The media file upload progress.
cover_progress	No	The thumbnail file upload progress.

FAQs

1. **How do I get the file object?**

Use the `input` tag and set `type` to `file`.

2. **Is there a size limit for upload?**

The maximum file size allowed is 60 GB.

3. **What browsers does the SDK support?**

The SDK supports Chrome, Firefox, and other mainstream browsers that support HTML5. It can also be used on IE 10 or later.

4. **How to pause and resume an upload?**

Automatic checkpoint restart is implemented at the underlying layer of the SDK. Therefore, to pause an upload, simply

call `uploader.cancel()` , and to resume an upload after pause, call `tcVod.upload` . Note that when you use `tcVod.upload` to resume an upload, you need to pass in the same parameters used when you initiate the upload (you can use a global variable to save the parameters when you initiate the upload and delete them after upload.)

5. **Does the SDK support `https:` upload?**

Yes, it does. The SDK uses `http:` for upload on HTTP pages and `https:` on non-HTTP pages.

Upload SDK for Android

Last updated：2025-03-07 16:08:37

Upload VOD provides an SDK for uploading videos from Android clients. For details about the upload process, see Guide.

SDK Name	Vod Upload SDK For Android
Version	V1.2.6.0
SDK Introduce	Providing a scenario for end-users of an app to upload local videos to a cloud video on demand platform:
Developer	Tencent Cloud Computing (Beijing) Co., Ltd.
Download SDK	1. Click to download the iOS upload Demo and source code, unzip the downloaded package, and you can see the Demo directory. 2. Upload the source code in the `Demo/ugcupload` directory.

Integrating the Source code and Libraries

1. Copy the upload source code directory `Demo/ugcupload` (i.e., the `ugcupload` module) to your project directory. If required, you may manually modify the package name.
2. If manual integration is required, you can refer to `Demo/ugcupload/build.gradle` to add dependencies in your project:

```
implementation 'com.qcloud.cos:cos-android-nobeacon:5.9.25'
```

Note：
You can also refer to the manual integration documentation to integrate the corresponding version of the dependency library.
3. If you need to use the QUIC capability, you need to copy the `ugcquic` module to your project directory and make it a dependency of the `ugcupload` module. If you need to perform manual integration, you can add the following dependencies:

```
implementation 'com.qcloud.cos:quic:1.5.45'
```

4. To use video uploading, you need network and storage access permissions. You can add the following permission declarations in the `AndroidManifest.xml` file:

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

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

5. Video uploading requires refreshing the upload IP based on network changes. You can dynamically register broadcasts according to your business needs, as shown in the following example:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // your code......
    registerNetReceiver();
}

private void registerNetReceiver() {
  if (null == mNetWorkStateReceiver) {
      mNetWorkStateReceiver = new TVCNetWorkStateReceiver();
      IntentFilter intentFilter = new IntentFilter(ConnectivityManager.CONNECTIVITY
      registerReceiver(mNetWorkStateReceiver, intentFilter);
   }
}

private void unRegisterNetReceiver() {
    if (null != mNetWorkStateReceiver) {
      unregisterReceiver(mNetWorkStateReceiver);
    }
}

@Override
protected void onDestroy() {
    super.onDestroy();
    // your code......
    unRegisterNetReceiver();
}
```

Simple video uploading

Initializing the upload object

```
TXUGCPublish mVideoPublish = new TXUGCPublish(this.getApplicationContext(), "indepe
```

Setting upload object callbacks

```
mVideoPublish.setListener(new TXUGCPublishTypeDef.ITXVideoPublishListener() {
    @Override
    public void onPublishProgress(long uploadBytes, long totalBytes) {
        mProgress.setProgress((int) (100*uploadBytes/totalBytes));
    }

    @Override
    public void onPublishComplete(TXUGCPublishTypeDef.TXPublishResult result) {
        mResultMsg.setText(result.retCode + " Msg:" + (result.retCode == 0 ? result
    }
});
```

Constructing upload parameters

```
TXUGCPublishTypeDef.TXPublishParam param = new TXUGCPublishTypeDef.TXPublishParam()

param.signature = "xxx";
param.videoPath = "xxx";
```

`Signature` calculation rules, please refer to Client-side Upload Signature.

Call upload

```
int publishCode = mVideoPublish.publishVideo(param);
```

Simple image upload

Initialize the upload object

```
TXUGCPublish mVideoPublish = new TXUGCPublish(this.getApplicationContext(), "indepe
```

Set upload object callbacks

```
mVideoPublish.setListener(new TXUGCPublishTypeDef.ITXMediaPublishListener() {
    @Override
    public void onMediaPublishProgress(long uploadBytes, long totalBytes) {
        mProgress.setProgress((int) (100*uploadBytes/totalBytes));
    }
    @Override
    public void onMediaPublishComplete(TXUGCPublishTypeDef.TXMediaPublishResult med
        mResultMsg.setText(result.retCode + " Msg:" + (result.retCode == 0 ? result
    }
```

```
});
```

Construct upload parameters

```
TXUGCPublishTypeDef.TXMediaPublishParam param = new TXUGCPublishTypeDef.TXMediaPubl

param.signature = "xxx";
param.mediaPath = "xxx";
```

`Signature` calculation rules, please refer to Client-side Upload Signature.

Call upload

```
int publishCode = mVideoPublish.publishMedia(param);
```

Note：

The upload method automatically chooses normal upload or slice upload based on the length of the user's file, and the user does not need to worry about the steps of slice upload, which can realize slice upload.

If you need to upload to a specific application, please refer Application System - Client-side Upload.

Advanced Features

With cover

Just include the cover path

```
TXUGCPublishTypeDef.TXPublishParam param = new TXUGCPublishTypeDef.TXPublishParam()
param.signature = "xxx";
param.videoPath = "xxx";
param.coverPath = "xxx";
```

`Signature` calculation rules, please refer to Client-side Upload Signature.

Cancel and Resume Upload

Cancel the upload by calling the `cancelublish()` interface of `TXUGCPublish` .

```
mVideoPublish.canclePublish();
```

Resume the upload by using the same upload parameters (the video path and cover image path remain unchanged) and call the `publishVideo` method of `TXUGCPublish` again.

Breakpoint resume

During the video upload process, cloud video on demand supports breakpoint resume, which means that if the upload is accidentally terminated, users can continue uploading the file from where it was interrupted, reducing the time spent on re-uploading. The valid time for breakpoint resume is 1 day, meaning that if the same video upload is interrupted, it can be directly uploaded from the breakpoint within 1 day. After 1 day, it will default to re-uploading the entire video. The `enableResume` parameter in the upload settings is the switch for breakpoint resume, which is enabled by default.

Pre-upload

In actual upload processes, a large portion of errors are caused by network connection failures or timeouts. To optimize such issues, we have added pre-upload optimization logic. Pre-upload includes: HTTPDNS resolution, obtaining suggested upload regions, and detecting the optimal upload region.
We recommend that you call `TXUGCPublishOptCenter.getInstance().prepareUpload(signature)` when your app starts. The pre-upload module will cache the<domain name, IP> mapping table and the optimal upload region locally. If network broadcasts have been dynamically registered previously, upon subscription to network switches, the cache will be cleared and automatically refreshed.
`Signature` calculation rules, please refer to Client-side Upload Signature.

Enabling HTTPS Upload

To enable HTTPS upload, simply set the `enableHTTPS` parameter in `TXPublishParam` to `true` . By default, it is set to `false` .

```
TXUGCPublishTypeDef.TXPublishParam param = new TXUGCPublishTypeDef.TXPublishParam()
param.enableHttps = true;
```

Disable logging

To disable logging, you need to operate through the `setIsDebug` method of `TXUGCPublish` . By default, it is enabled. When enabled, it will print logcat logs and also save the logs to the app's private directory.

```
// false Disable logging
mTXUGCPublish.setIsDebug(false);
```

Video Upload Interface Description

Initialize the upload object: `TXUGCPublish`

Parameter Name	Parameter Description	Type	Required
context	application context	Context	YES

customKey	Used to distinguish different users, it is recommended to use the App's account ID for easier subsequent issue tracking.	String	NO

Set the Vod app ID: `TXUGCPublish.setAppId`

Parameter Name	Parameter Description	Type	Required
appId	vod appId。	int	YES

upload video： `TXUGCPublish.publishVideo`

Parameter Name	Parameter Description	Type	Required
param	upload params.	TXUGCPublishTypeDef.TXPublishParam	YES

upload params： `TXUGCPublishTypeDef.TXPublishParam`

Parameter Name	Parameter Description	Type	Required
signature	Signature for Upload from Client	String	YES
videoPath	Local video file path.	String	YES
coverPath	Local cover file path, default without cover file.	String	NO
enableResume	Whether to enable resuming from the breakpoint, default is enabled.	boolean	NO
enableHttps	Whether to enable HTTPS, default is disabled.	boolean	NO
fileName	The name of the video file uploaded to Tencent Cloud, if not filled, the default is the local file name.	String	NO
enablePreparePublish	Whether to enable the pre-upload mechanism, default is enabled. The pre-upload mechanism can significantly improve the upload quality of files	boolean	NO
sliceSize	Chunk size, supports a minimum of	long	NO

	1MB and a maximum of 10MB, default is the uploaded file size divided by 10		
concurrentCount	The maximum number of concurrent uploads for chunked uploads, default is 4.	int	NO
trafficLimit	The speed limit value setting range is 819200 ~ 838860800, that is, 100KB/s ~ 100MB/s. If it exceeds this range, a 400 error will be returned. It is not recommended to set this value too small to prevent timeouts. -1 indicates no speed limit.	long	NO
uploadResumeController	The resume controller, which can be customized to calculate and save the resume key values, defaults to using MD5 to calculate the file key values.	IUploadResumeController	NO

set upload callback：`TXUGCPublish.setListener`

Parameter Name	Parameter Description	Type	Required
listener	Upload progress and result callback subscription.	TXUGCPublishTypeDef.ITXVideoPublishListener	YES

progress callback：`TXUGCPublishTypeDef.ITXVideoPublishListener.onPublishProgress`

Parameter Name	Parameter Description	Type
uploadBytes	The number of bytes uploaded.	long
totalBytes	Total number of bytes.	long

result callback：`TXUGCPublishTypeDef.ITXVideoPublishListener.onPublishComplete`

Parameter Name	Parameter Description	Type
result	upload result	TXUGCPublishTypeDef.TXPublishResult

upload result： `TXUGCPublishTypeDef.TXPublishResult`

Parameter Name	Parameter Description	Type
retCode	result code	int
descMsg	Error description for upload failure.	String
videoId	Vod video file ID.	String
videoURL	Video storage address.	String
coverURL	Cover storage address.	String

Pre-upload： `TXUGCPublishOptCenter.prepareUpload`

Parameter Name	Parameter Description	Type	Required
signature	Signature for Upload from Client	String	YES

Image upload interface description

Initialize upload object： `TXUGCPublish`

Parameter Name	Parameter Description	Type	Required
context	application context	Context	YES
customKey	Used to distinguish different users, it is recommended to use the App's account ID for easier subsequent issue tracking.	String	NO

Set the Vod app ID: `TXUGCPublish.setAppId`

Parameter Name	Parameter Description	Type	Required
appId	vod appId。	int	YES

upload image： `TXUGCPublish.publishMedia`

Parameter Name	Parameter Description	Type	Required
param	upload params.	TXUGCPublishTypeDef.TXMediaPublishParam	YES

upload params： `TXUGCPublishTypeDef.TXMediaPublishParam`

Parameter Name	Parameter Description	Type	Required
signature	Signature for Upload from Client	String	YES
mediaPath	Local image file path.	String	YES
enableResume	Whether to enable resuming from the breakpoint, default is enabled.	boolean	NO
enableHttps	Whether to enable HTTPS, default is disabled.	boolean	NO
fileName	The name of the video file uploaded to Tencent Cloud, if not filled, the default is the local file name.	String	NO
enablePreparePublish	Whether to enable the pre-upload mechanism, default is enabled. The pre-upload mechanism can significantly improve the upload quality of files	boolean	NO
sliceSize	Chunk size, supports a minimum of 1MB and a maximum of 10MB, default is the uploaded file size divided by 10	long	NO
concurrentCount	The maximum number of concurrent uploads for chunked uploads, default is 4.	int	NO
trafficLimit	The speed limit value setting range is 819200 ~ 838860800, that is, 100KB/s ~ 100MB/s. If it exceeds this range, a 400 error will be returned. It is not recommended to set this value too small to prevent timeouts. -1 indicates no speed limit.	long	NO
uploadResumeController	The resume controller, which can be customized to calculate and save the resume key values, defaults to using MD5 to calculate the file key values.	IUploadResumeController	NO

set upload callback：`TXUGCPublish.setListener`

Parameter Name	Parameter Description	Type	Required
listener	Upload progress and result callback subscription.	TXUGCPublishTypeDef.ITXMediaPublishListener	YES

progress callback：`TXUGCPublishTypeDef.ITXMediaPublishListener.onPublishProgress`

Parameter Name	Parameter Description	Type
uploadBytes	The number of bytes uploaded.	long
totalBytes	Total number of bytes.	long

result callback：`TXUGCPublishTypeDef.ITXMediaPublishListener.onPublishComplete`

Parameter Name	Parameter Description	Type
result	upload result	TXUGCPublishTypeDef.TXPublishResult

upload result：`TXUGCPublishTypeDef.TXMediaPublishResult`

Parameter Name	Parameter Description	Type
retCode	result code	int
descMsg	Error description for upload failure.	String
mediaId	vod media file ID.	String
mediaURL	Media resource storage address.	String

Pre-upload：`TXUGCPublishOptCenter.prepareUpload`

Parameter Name	Parameter Description	Type	Required
signature	Signature for Upload from Client	String	YES

Error code

SDK uses the `TXUGCPublishTypeDef.ITXVideoPublishListener\\ITXMediaPublishListener` interface to subscribe to the status of video uploading. Therefore, you can use the `retCode` in `TXUGCPublishTypeDef.TXPublishResult\\TXMediaPublishResult` to determine the situation of video uploading.

Status code	Corresponding constant in TVCConstants	Meaning
0	NO_ERROR	Upload successful.
1001	ERR_UGC_REQUEST_FAILED	Request upload failed, usually due to an expired or invalid client signature, requiring the app to reapply for a signature.
1002	ERR_UGC_PARSE_FAILED	Failed to parse request information.
1003	ERR_UPLOAD_VIDEO_FAILED	Failed to upload video.
1004	ERR_UPLOAD_COVER_FAILED	Failed to upload cover.
1005	ERR_UGC_FINISH_REQUEST_FAILED	Failed to end upload request.
1006	ERR_UGC_FINISH_RESPONSE_FAILED	End upload response error.
1007	ERR_CLIENT_BUSY	Client is busy (object cannot handle more requests).
1008	ERR_FILE_NOEXIT	Uploaded file does not exist.
1009	ERR_UGC_PUBLISHING	Video is currently being uploaded.
1010	ERR_UGC_INVALID_PARAM	Video is currently being uploaded.
1012	ERR_UGC_INVALID_SIGNATURE	Video upload signature is empty.
1013	ERR_UGC_INVALID_VIDOPATH	Path to video file is empty.
1014	ERR_UGC_INVALID_VIDEO_FILE	Video file does not exist at current path.
1015	ERR_UGC_FILE_NAME	Video upload file name is too long (exceeds 40 characters) or contains special characters.
1016	ERR_UGC_INVALID_COVER_PATH	Incorrect cover path for video file, file does not exist.
1017	ERR_USER_CANCEL	User canceled upload.

1020	ERR_UPLOAD_SIGN_EXPIRED	Signature expired.

Upload SDK for iOS

Last updated：2025-03-07 16:08:37

Upload VOD provides an SDK for uploading videos from iOS clients. For details about the upload process, see Guide.

SDK name	Vod Upload SDK For IOS
Version	V1.2.6.0
SDK Introduce	Providing a scenario for end-users of an app to upload local videos to a cloud video on demand platform:
Developer	Tencent Cloud Computing (Beijing) Co., Ltd.
Download SDK	1. Click to download the iOS upload Demo and source code, unzip the downloaded package, and you can see the Demo directory. 2. The upload source code is located in the `LiteAVSDK_UGC_Upload_iOS/TXVodUpload` directory.

Integrating the Source code and Libraries

1. Copy the upload source code directory `TXVodUpload` and `TXVodUpload.podspec` into your project.
2. Add the following dependency to your Podfile:

```
  pod 'TXVodUpload', :path => '/The path to the location of `TXVodUpload.podspec`'
```

3. If manual integration is required, you can add the following dependencies to your `Podfile` :

```
pod 'QCloudQuic','6.3.7'
pod 'QCloudCOSXML/Slim','6.4.4'
// Based on your project, the dependency is already available, so there is no need
pod 'AFNetworking','4.0.1'
```

4. Under the **Build Settings** tab, add `-ObjC` to **Other Linker Flags**.

Uploading Videos

Initialize an upload object

```
TXUGCPublish *_videoPublish = [[TXUGCPublish alloc] initWithUserID:@"upload_video_u
```

Set the upload object callback

```
_videoPublish.delegate = self;
```

```
#pragma mark - TXVideoPublishListener

- (void)onPublishProgress:(NSInteger)uploadBytes totalBytes:(NSInteger)totalBytes {
    self.progressView.progress = (float)uploadBytes/totalBytes;
    NSLog(@"onPublishProgress [%ld/%ld]", uploadBytes, totalBytes);
}

- (void)onPublishComplete:(TXPublishResult*)result {
    NSString *string = [NSString stringWithFormat:@"Upload completed; error code: [
    [self showErrorMessage:string];
    NSLog(@"onPublishComplete [%d/%@]", result.retCode, result.retCode == 0? result
}
```

Construct upload parameters

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

publishParam.signature  = @"The signature generated by your business backend";
publishParam.videoPath  = @"The path of the video file";
```

For details on how to calculate the signature, see Signature for Upload from Client.

Call the upload API

```
[_videoPublish publishVideo:publishParam];
```

Note:

The upload API automatically selects simple upload or multipart upload based on the file size. You don't need to manually set up multipart upload.

To upload to a subapplication, see Subapplication System - Upload from client.

Advanced Features

Uploading a thumbnail

To upload a thumbnail, pass in the thumbnail path.

```
TXPublishParam *publishParam = [[TXPublishParam alloc] init];
publishParam.signature  = @"The signature generated by your business backend";
```

```
publishParam.coverPath = @"The path of the thumbnail image";
publishParam.videoPath  = @"The path of the video file";
```

Canceling and resuming upload

To cancel an upload, call the `cancelPublish` API.

```
[_videoPublish cancelPublish];
```

To resume an upload, call `publishVideo` of `TXUGCPublish` again, passing in the same upload parameters and video and thumbnail paths.

Setting up checkpoint restart

VOD supports checkpoint restart. If an upload is interrupted, when you upload the same file again, the upload can start from where it left off. This works only if a file is uploaded again within one day. If the interval exceeds one day, you will need to upload the full video again.
You can use the `enableResume` parameter to enable or disable checkpoint start. It's enabled by default.

Enabling HTTPS upload

To enable HTTPS upload, set `enableHTTPS` in `TXPublishParam` to `true`.

```
TXPublishParam *publishParam = [[TXPublishParam alloc] init];
publishParam.enableHTTPS  = true;
```

Turn off logs

Turning off logs needs to be done through the setIsDebug method of TXUGCPublish, which is enabled by default. When enabled, logcat logs will be printed, and logs will also be saved to the app's private directory.

```
// NO: Turn off logs
[_videoPublish setIsDebug:NO];
```

Uploading Images and Other Media Files

```
// Create an object
TXUGCPublish *_imagePublish = [[TXUGCPublish alloc] initWithUserID:@"upload_image_u

// Set the callback
_imagePublish.mediaDelegate = self;
```

```
// Construct upload parameters
TXMediaPublishParam *publishParam = [[TXMediaPublishParam alloc] init];
publishParam.signature  = @"The signature generated by your business backend";
publishParam.mediaPath = @"Path of the image file";

// Upload an image or media file
[_imagePublish publishMedia:publishParam];
```

Video Upload APIs

`TXUGCPublish::initWithUserID` : Initialize an upload object

Parameter	Description	Type	Required
userID	The user ID.	NSString	No

`TXUGCPublish.publishVideo` : Upload a video

Parameter	Description	Type	Required
param	The publishing parameters.	TXPublishParam	Yes

`TXPublishParam` : Upload parameters

Parameter	Description	Type	Required
signature	The client upload signature.	NSString*	YES
videoPath	The path of the local video file.	NSString*	YES
coverPath	The path of the local thumbnail image (optional).	NSString*	NO
fileName	The name of the uploaded file in Tencent Cloud. If this parameter is left empty, the original filename will be used.	NSString*	NO
enableResume	Whether to enable checkpoint restart. It's enabled by default.	BOOL	NO
enableHttps	Whether to enable HTTPS. It's disabled by default.	BOOL	NO

fileName	The name of the video file uploaded to Tencent Cloud, if not filled, the default is the local file name.	String	NO
enablePreparePublish	Whether to enable the pre-upload mechanism, default is enabled. The pre-upload mechanism can significantly improve the upload quality of files	boolean	NO
sliceSize	Chunk size, supports a minimum of 1MB and a maximum of 10MB, default is the uploaded file size divided by 10	long	NO
concurrentCount	The maximum number of concurrent uploads for chunked uploads, default is 4.	int	NO
trafficLimit	The speed limit value setting range is 819200 ~ 838860800, that is, 100KB/s ~ 100MB/s. If it exceeds this range, a 400 error will be returned. It is not recommended to set this value too small to prevent timeouts. -1 indicates no speed limit.	long	NO
uploadResumeController	The resume controller, which can be customized to calculate and save the resume key values, defaults to using MD5 to calculate the file key values.	IUploadResumeController	NO

`TXUGCPublish.delegate` : Set upload callbacks

Member variable	Description	Type	Required
delegate	The upload progress and result callbacks.	TXVideoPublishListener	Yes

`onPublishProgress` : The upload progress callback

Member variable	Description	Type
uploadBytes	Uploaded bytes.	NSInteger
totalBytes	Total bytes.	NSInteger

`onPublishComplete` : The upload result callback

Member variable	Description	Type
result	The upload result.	TXPublishResult

`onPublishEvent` : The upload event callback

Member variable	Description	Type
evt	The upload event, which can be printed and used for debugging.	NSDictionary

`TXPublishResult` : The upload result

Member variable	Description	Type
retCode	The error code	int
descMsg	The error message.	NSString
videoId	The VOD file ID.	NSString
videoURL	The video URL.	NSString
coverURL	The thumbnail URL.	NSString

`TXUGCPublishOptCenter.prepareUpload` : Set up pre-upload

Parameter	Description	Type	Required
signature	The client upload signature.	NSString	Yes

Image and Other Media Upload APIs

`TXUGCPublish::initWithUserID` : Initialize an upload object

Parameter	Description	Type	Required
userID	The user ID.	NSString	No

`TXUGCPublish.publishMedia` : Start an upload

Parameter	Description	Type	Required
param	The publishing parameters.	TXMediaPublishParam	Yes

`TXMediaPublishParam` : Upload parameters

Parameter	Description	Type	Required
signature	The client upload signature.	NSString*	YES
mediaPath	The path of the local media file.	NSString*	YES
fileName	The name of the uploaded file in Tencent Cloud. If this parameter is left empty, the original filename will be used.	NSString*	NO
enableResume	Whether to enable checkpoint restart. It's enabled by default.	BOOL	NO
enableHttps	Whether to enable HTTPS. It's disabled by default.	BOOL	NO
fileName	The name of the video file uploaded to Tencent Cloud, if not filled, the default is the local file name.	String	NO
enablePreparePublish	Whether to enable the pre-upload mechanism, default is enabled. The pre-upload mechanism can significantly improve the upload quality of files	boolean	NO
sliceSize	Chunk size, supports a minimum of 1MB and a maximum of 10MB, default is the uploaded file size divided by 10	long	NO
concurrentCount	The maximum number of concurrent uploads for chunked uploads, default is 4.	int	NO
trafficLimit	The speed limit value setting range is 819200 ~ 838860800, that is, 100KB/s ~ 100MB/s. If it exceeds this range, a 400 error will be returned. It is not recommended to set this value too	long	NO

	small to prevent timeouts. -1 indicates no speed limit.		
uploadResumeController	The resume controller, which can be customized to calculate and save the resume key values, defaults to using MD5 to calculate the file key values.	IUploadResumeController	NO

`TXUGCPublish.TXMediaPublishListener` : Set upload callbacks

Member variable	Description	Type	Required
mediaDelegate	The upload progress and result callbacks.	TXMediaPublishListener	Yes

`onMediaPublishProgress` : The upload progress callback

Member variable	Description	Type
uploadBytes	Uploaded bytes.	NSInteger
totalBytes	Total bytes.	NSInteger

`onMediaPublishComplete` : The upload result callback

Member variable	Description	Type
result	The upload result.	TXMediaPublishResult

`onMediaPublishEvent` : The upload event callback

Member variable	Description	Type
evt	The upload event, which can be printed and used for debugging.	NSDictionary

`TXMediaPublishResult` : The upload result

Member variable	Description	Type
retCode	The error code.	int
descMsg	The error message.	NSString
mediaId	The file ID of the image/media file.	NSString
mediaURL	The URL of the image/media file.	NSString

`TXUGCPublishOptCenter.prepareUpload` : Set up pre-upload

Parameter	Description	Type	Required
signature	The client upload signature.	NSString	Yes

Error codes

The SDK listens for video upload status using `TXMediaPublishListener` . Therefore, to get the upload status, check `retCode` in `TXMediaPublishResult` .

Error Codes	TVCCommon Constant	Description
0	TVC_OK	Uploaded successfully.
1001	TVC_ERR_UGC_REQUEST_FAILED	The upload request failed, usually because the client signature has expired or is invalid. The app needs to reapply for a signature.
1002	TVC_ERR_UGC_PARSE_FAILED	Request information parsing failed.
1003	TVC_ERR_VIDEO_UPLOAD_FAILED	Upload video failed.
1004	TVC_ERR_COVER_UPLOAD_FAILED	Upload cover failed.
1005	TVC_ERR_UGC_FINISH_REQ_FAILED	Failed to end upload request.
1006	TVC_ERR_UGC_FINISH_RSP_FAILED	End upload response error.
1008	TVC_ERR_FILE_NOT_EXIST	The file does not exist at the specified file path.
1009	TVC_ERR_ERR_UGC_PUBLISHING	The video is currently uploading.
1010	TVC_ERR_UGC_INVALID_PARAME	Invalid parameter.
1012	TVC_ERR_INVALID_SIGNATURE	Upload signature is empty.
1013	TVC_ERR_INVALID_VIDEOPATH	Video path is empty.
1017	TVC_ERR_USER_CANCLE	User initiated upload cancellation.
1020	TVC_ERR_UPLOAD_SIGN_EXPIRED	Signature expired.

Upload SDK for Flutter

Last updated：2025-03-07 16:08:37

VOD provides an SDK for uploading videos from Flutter clients. For details about the upload process, see Guide.

SDK Name	Cloud Video on Demand Flutter Upload SDK
Version Number	V1.5.0
SDK Introduction	Provides a scenario for app end-users to upload local videos to the Cloud Video on Demand platform.
Download SDK	1. Click to download the Flutter Upload SDK and source code, unzip the downloaded compressed package, and you can see the vod_upload directory. 2. The upload source code is located in the `vod_upload/lib` directory.

Environment Setup

Flutter：

Flutter 3.0.0 and above

Dart 2.19.2 and below 4.0

Android:

Android Studio 3.5 and above

Android 4.1 and above

iOS：

Xcode 11.0 and above

iOS 9.0 and above

Make sure your project has a valid developer signature set up

Quick Integration

Add Dependencies

1. Copy the SDK source code to your project directory.

2. Add the SDK to `pubspec.yaml`

```
vod_upload_flutter:
  path: ./vod_upload
```

3. Run the command `flutter pub get` in the root directory of your project to refresh the dependencies.

Note：

1. It is recommended to run `flutter pub get` command separately in the `root directory`, `SDK directory`, and `SDK Example directory` to avoid potential errors.
2. The `SDK Example directory` is the test project for the SDK. You can delete it if not needed.

Add Native Configurations

Android

Add the following configurations to `AndroidManifest.xml`.

```
<!-- Network permissions -->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

iOS

Add the following configuration to `Info.plist` in `iOS`.

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
</dict>
```

Note：

If you want to run the provided `Demo` in the `SDK`, you should also declare permission to use the photo library.

Usage

1. Import the file.

```
import 'package:vod_upload_flutter/txugc_publish.dart';
```

2. Create an object.

```
var uploader = TXUGCPublish(
    id: "",
);
```

Note：

The `id` can be any string as long as it is `unique`. The main purpose is to map the Flutter object to the native layer object.

API

Upload Video

```
uploader.publishVideo(TXPublishParam(
      signature: "",
      videoPath: "",
      fileName: "",
));
```

Cancel Video Upload

```
uploader.cancelUploadVideo();
```

Resume Video Upload

```
uploader.resumeUploadVideo(TXPublishParam(
      signature: "",
      videoPath: "",
      fileName: "",
));
```

Upload Media File

```
uploader.publishMedia(TXMediaPublishParam(
      signature: "",
      mediaPath: "",
      fileName: "",
));
```

Cancel Media File Upload

```
uploader.cancelUploadMedia();
```

Resume Media File Upload

```
uploader.resumeUploadMedia(TXMediaPublishParam(
      signature: "",
      mediaPath: "",
      fileName: "",
));
```

Prepare Upload

```
TXUGCPublish.prepareUpload(signature, callback);
```

Note：

Prepare upload is a `static method` .

Get Upload Information

```
// On Android, you can only get information during the upload process, while on iOS
uploader.getStatusInfo();
```

Report `AppId`

```
uploader.setAppId(appId);
```

Set Video Upload Callback

```
uploader.setVideoListener(listener);
```

Set Media Upload Callback

```
uploader.setMediaListener(listener);
```

Callback Interfaces and Parameter Explanations

Video Upload Parameters

`TXPublishParam`

Field	Type	Required	Explanation	Default Value
signature	string	Yes	Signature	null
videoPath	string	Yes	Video path	null
fileName	string	Yes	File name	null
enableResume	boolean	No	Enable resumable upload	true

enableHttps	boolean	No	Enable HTTPS	false
coverPath	string	No	Cover image	null
enablePreparePublish	boolean	No	Enable prepare upload (can be manually triggered if disabled)	true
sliceSize	integer	No	Chunk size (minimum 1M, maximum 10M, default 0, which means the file size divided by 10)	0
concurrentCount	integer	No	Concurrent number of chunk uploads (if <=0, the default value of 2 will be used)	-1

Media Upload Parameters

`TXMediaPublishParam`

Field	Type	Required	Explanation	Default Value
signature	string	Yes	Signature	null
mediaPath	string	Yes	Media file path	null
fileName	string	Yes	File name	null
enableResume	boolean	No	Enable resumable upload	true
enableHttps	boolean	No	Enable HTTPS	false
coverPath	string	No	Cover image	null
enablePreparePublish	boolean	No	Enable prepare upload (can be manually triggered if disabled)	true
sliceSize	integer	No	Chunk size (minimum 1M, maximum 10M, default 0, which means the file size divided by 10)	0
concurrentCount	integer	No	Concurrent number of chunk uploads (if <=0, the default value of 2 will be used)	-1

Video Upload Callback

`ITXVideoPublishListener`

Method	Return	Explanation

	Type	
onPublishProgress	void	Upload progress callback
onPublishComplete	void	Upload completion callback

Parameter explanation:

`onPublishProgress`

Parameter	Type	Explanation
uploadBytes	integer	Number of bytes uploaded
totalBytes	integer	Total number of bytes

`onPublishComplete`

Parameter	Type	Explanation
result	TXPublishResult	Upload result

`TXPublishResult`

Parameter	Type	Explanation
retCode	integer	Error code
descMsg	string	Error description
videoId	string	Video file ID
videoURL	string	Video playback URL
coverURL	string	Cover image storage URL

Media File Upload Callback

`ITXMediaPublishListener`

Method	Return Type	Explanation
onMediaPublishProgress	void	Upload progress callback
onMediaPublishComplete	void	Upload completion callback

Parameter explanation:

`onMediaPublishProgress`

Parameter	Type	Explanation
uploadBytes	integer	Number of bytes uploaded
totalBytes	integer	Total number of bytes

`onMediaPublishComplete`

Parameter	Type	Explanation
result	TXMediaPublishResult	Upload result

`TXMediaPublishResult`

Parameter	Type	Explanation
retCode	integer	Error code
descMsg	string	Error description
mediaId	string	Media file ID
mediaURL	string	Media file URL

Prepare Upload Callback

`IPrepareUploadCallback`

Method	Return Type	Explanation
onLoading	void	Prepare upload start callback
onFinish	void	Prepare upload completion callback

Upload Status Information

`ReportInfo`

Field	Type	Explanation
reqType	string	Request type, indicating the current step
errCode	string	Error code

cosErrCode	string	COS upload error code
errMsg	string	Error message
reqTime	string	Request start time for the current step
reqTimeCost	string	Time spent on the current step
fileSize	string	File size
fileType	string	File type
fileName	string	File name
fileId	string	File ID
appId	string	VOD App ID set through TXUGCPublish
reqServerIp	string	IP address accessed during the current step
reportId	string	Custom report ID provided by the customer, can be passed through the TXUGCPublish constructor
reqKey	string	Request key, usually composed of the last modification time of the file and the start time of this upload
vodSessionKey	string	Session key from the VOD server, obtained from the upload request interface
cosRegion	string	Region accessed during the current upload
requestId	string	Request ID for the current COS upload
cosVideoPath	string	Path for the current COS video upload
vodErrCode	integer	Signaling request error code
useHttpDNS	integer	Whether to use httpDns for domain name resolution
useCosAcc	integer	Whether COS domain name acceleration is enabled
tcpConnTimeCost	integer	Time spent on connecting to the server in the current step
recvRespTimeCost	integer	Time spent on receiving server response in the current step

Media Processing
Video Processing
Just-in-Time Transcoding

Last updated：2024-05-07 19:06:43

Traditional transcoding technology requires decoding and encoding the entire audio and video before playback, which operates in an asynchronous processing mode, leading to longer waiting times for users. However, Just-in-time transcoding technology allows for immediate playback without waiting, regardless of the video's length, achieving a seamless start within seconds and providing users with an entirely new playback experience.

Note:

If you need to use the Just-in-time transcoding feature, please contact sales to apply for activation.

Just-in-Time Transcoding Template

The JIT transcoding template includes parameters such as resolution and bitrate. VOD uses a JIT transcoding template to represent a set of transcoding parameters. Through the template, you can specify the following transcoding-related parameters.

Item	Parameter	Description
Video transcoding	Resolution	Supported width range：128px - 1920px Supported height range：128px - 1920px
	Bitrate	Supported video bitrate range：128kbps - 10000kbps
Watermark	Watermark image	the base64 of watermark image
	Watermark position	the position of watermark
	Watermark resolution	the resolution of watermark

For common usage scenarios, VOD provides the following preset JIT transcoding templates.

Template name	Video resolution	Bitrate	Watermark
hls_avc_540_preset	540P	1000kbps	none
hls_avc_720_preset	720P	1800kbps	none
hls_avc_1080_preset	1080P	2500kbps	none

In addition, you can create and manage custom JIT transcoding templates through the server-side API.

Guide

1. After uploading the video to VOD, you could obtain the URL address of the video through the VOD console or server API :

```
http://example.com/dir1/dir2/myVideo.mp4
```

2. Splice the JIT transcoding parameters and replace templateName in the URL with the template name to get the JIT transcoded playback URL.

```
http://example.com/dir1/dir2/myVideo.mp4$JM!Transcode,Template=
{templateName}/index.m3u8
```

3. The following case uses the preset template name `hls_avc_720_preset` as an example to show how to splice the JIT transcoding playback URL.

```
http://example.com/dir1/dir2/myVideo.mp4$JM!Transcode,Template=hls_avc_720_pres
et/index.m3u8
```

Overview

Last updated：2021-10-29 11:18:33

Video processing is a process in which a source video is analyzed or processed to generate a new video.

Category	Name	Description
Video editing	LVB push	Clipping: cuts a clip out of a video to generate a new video. Splicing: splices multiple videos to generate a new video.
	Video editing	Manipulates media files through operations such as clipping, splicing, overlaying, and flipping to achieve effects such as audio mixing, audio extraction, and picture-in-picture.
Video conversion	Transcoding	Transcodes a video to a new one in the specified format and with the specified resolution.
	Screencapture	Screencaptures a video at the specified time point or interval.
	Watermarking	Adds a text or image watermark to a video during transcoding.
	Animated image generating	Converts a video segment into an animated image in GIF or WebP format.
	Adaptive bitrate streaming	Transcodes a video to adaptive bitstream in HLS or Dash format.
	Video encryption	Encrypts a video by using commercial-grade DRM (FairPlay or Widevine).
Video AI	Intelligent video content recognition	Intelligently recognizes porn, terrorism, and politically sensitive information in video content.
	Video content analysis	Intelligently analyzes video content (e.g., categorization, tagging, and cover generating).
	Video content recognition	Intelligently recognizes video content.

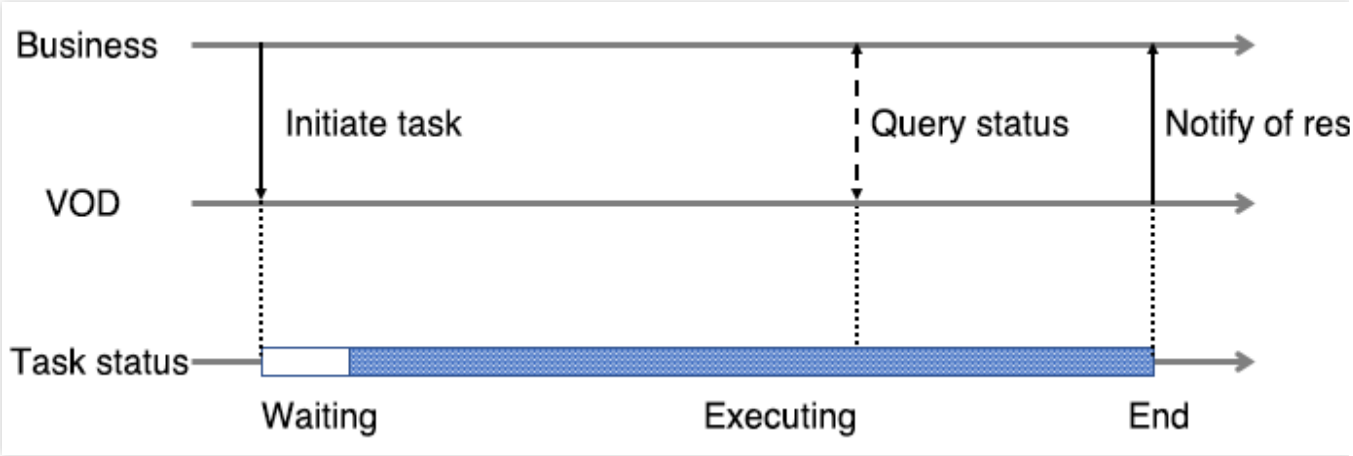
The above is a list of video processing features provided by VOD. In addition to a range of basic processing capabilities such as transcoding, screencapture, and watermarking, VOD also has the following two distinctive

capabilities:
intelligent content recognition and analysis with the aid of Tencent Cloud's powerful AI.
integration with commercial-grade DRM for high-level video encryption.
After a video processing task is initiated, the processing result cannot be output immediately (i.e., the result cannot be obtained synchronously). Therefore, video processing is performed as an offline task. For more information on how to initiate a video processing task and get the task result, please see Video Processing Task System.

Video Processing Task System

Last updated：2021-10-29 11:22:21

After a video processing task is initiated, it takes a few minutes to a few hours for the task to complete execution and output the result. Video processing is essentially an offline task. Taking into account the characteristics of video processing tasks, VOD provides a task system allowing you to initiate tasks synchronously and receive task execution result notifications asynchronously.



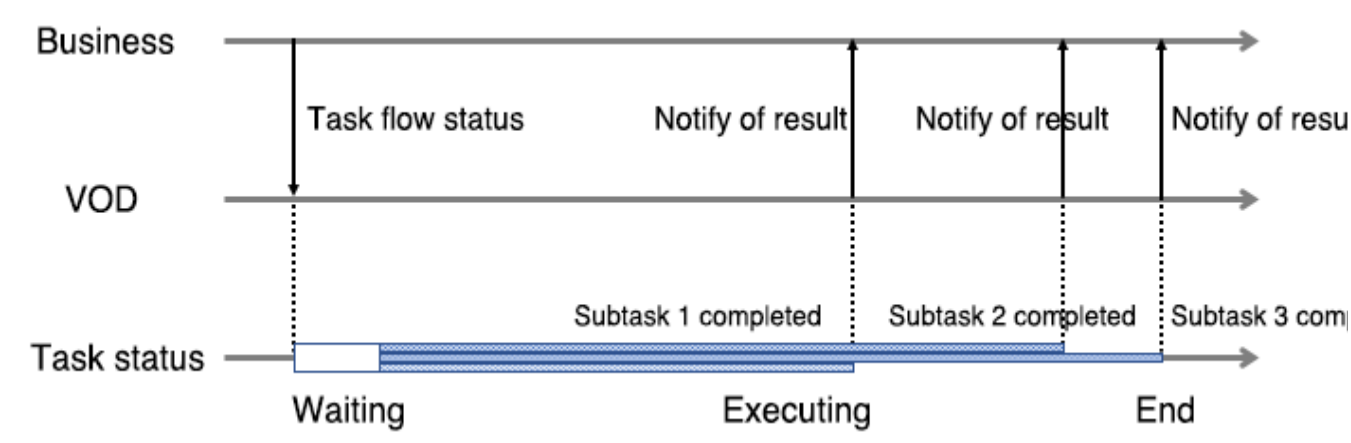
Initiate a task: after a video processing task is submitted, VOD will immediately return a task ID to you and will wait for some time to start executing the task.
Notify of result: upon task completion, VOD will send you a result notification, which contains the task ID and execution result.
Query a task: after submitting a task, you can query the execution status and result of the task by task ID at any time.

Parameter Template

Video processing parameters are usually quite complicated. For example, video transcoding involves dozens of parameters such as container format, codec, bitrate, resolution, and frame rate. In order to simplify the parameters of video processing tasks, VOD offers a variety of integrated parameter templates (e.g., transcoding templates), which are identified by template ID.
Preset parameter templates: as for common video processing parameter sets, VOD provides a batch of preset parameter templates. For more information, please see List of Preset Parameter Templates.
Custom parameter templates: VOD supports customizing parameter templates through the console or server API.

Task Flow

In VOD, the following video processing operations are independent tasks:
Transcoding to MP4 LD video
Transcoding to MP4 SD video
Sampled screencapturing at intervals of 10s
Intelligent recognition
Intelligent categorization
If several independent tasks are executed at the same time, there will be multiple task IDs, and you will have to receive and deal with multiple task result notifications. In order to simply the initiation and notification of multiple tasks, VOD offers a task flow scheme. A task flow is essentially a "parent task" composed of multiple subtasks. Initiating a task flow is equivalent to initiating all the subtasks.



As shown in the figure, the task flow contains three subtasks and ends when the last subtask (subtask 3) is completed. Task flow result notification will be triggered when the task flow ends as well as when each of the subtasks is completed, enabling you to perceive the execution result of any subtask in real time.
Most video processing tasks in VOD are performed in the form of a task flow, which can be regarded as a special type of task. VOD also supports creating task flow templates and naming them. When initiating a task flow, you can use the task flow template name to indicate the desired task.

Task Initiation

There are three ways to initiate a video processing task, namely, initiating through server API, initiating through the console, and specifying a task upon upload.

Initiating through server API

Through server APIs, you can directly initiate a task for a video in VOD or edit it and specify the tasks to be executed for the generated new video.
ProcessMedia

ProcessMediaByUrl
EditMedia

Initiating through console

You can initiate a task for a video in VOD through the console. For more information, please see Processing Videos.

Specifying task upon upload

VOD offers three ways to upload a video: upload from client, upload from server, and upload through the console. All of them support specifying the task to be executed upon upload.
Upload from client: you can specify a task upon upload through the `procedure` parameter in the signature for upload from client.
Upload from server: you can specify a task upon upload through the `procedure` parameter in the ApplyUpload API.
Upload through console: you can upload a video through the console, select **Process Video During Upload**, and specify the task upon upload. For detailed directions, please see Uploading Videos.

Result Notification

After initiating a video processing task, you need to perceive the task execution result asynchronously through "result notification".
Video processing result notifications mainly include the following types:
Task Flow Status Change
Video Editing Completion
Video processing result notifications are a type of "event notifications" in VOD, which can be received in two modes: "HTTP normal callback" and "reliable callback". For more information, please see Event Notification.

Task Query

In addition to perceiving the task execution result through result notifications, you can poll task execution status by task ID as scheduled, which is called "task query". Currently, VOD only provides the DescribeTasks and DescribeTaskDetail server APIs for querying task execution status and execution result.

Preset Parameter Template List

Last updated：2025-03-07 16:08:37

VOD offers preset parameter templates for different video processing scenarios. Instead of configuring complicated parameter sets, you can use the ready-made templates to initiate video processing tasks.

Video Conversion

Preset parameter templates for video conversion:
Preset transcoding templates
Preset remuxing templates
Preset animated image generating templates
Preset time point screencapturing templates
Preset sampled screencapturing templates
Preset image sprite generating templates
Preset adaptive bitrate streaming templates

Preset transcoding templates

Video transcoding

Clarity	Template ID	Format	Video Parameters				Audio Parameters		
			Resolution	Bitrate (Kbps)	Frame Rate (fps)	Codec	Bitrate (Kbps)	Sample Rate (Hz)	Sc Cl
Smooth	100010	MP4	Vertical: 360; horizontal: Proportionally scaled	400	25	H.264	64	44100	St
	100210	HLS							
SD	100020	MP4	Vertical: 540; horizontal: Proportionally scaled	1000					
	100220	HLS							
HD	100030	MP4	Vertical: 720; horizontal: Proportionally scaled	1800			128		
	100230	HLS							

FHD	100040	MP4	Vertical: 1080; horizontal: Proportionally scaled	2500				
	100240	HLS						
2K	100070	MP4	Vertical: 1440; horizontal: Proportionally scaled	3000			160	
	100270	HLS						
4K	100080	MP4	Vertical: 2160; horizontal: Proportionally scaled	6000				
	100280	HLS						

Audio transcoding

Template ID	Format	Bitrate	Codec	Sound Channels	Sample Rate
1100	M4A	24 Kbps	AAC	Stereo	44,100 Hz
1110		48 Kbps			
1120		96 Kbps			
1130		192 Kbps			
1140		256 Kbps			
1010	MP3	128 Kbps	MP3		
1020		320 Kbps			

Preset Top Speed Codec (TSC) transcoding templates

Clarity	Template ID	Format	Video Parameters				Audio Parameters	
			Resolution	Maximum Bitrate	Frame Rate (fps)	Codec	Bitrate	Sample Rate
Same as source	100800	MP4	Same as source	No limit	25	H.264	Same as source	44,100 Hz
Smooth	100810		Vertical: 360; horizontal: proportionally scaled				64 Kbps	
SD	100820		Vertical: 540; horizontal: proportionally scaled					
HD	100830		Vertical: 720; horizontal: proportionally scaled				128 Kbps	
FHD	100840		Vertical: 1080; horizontal: proportionally scaled					

Preset remuxing templates

Template ID	Format
875	MP4
876	HLS

Preset animated image generating templates

Template ID	Format	Resolution	Frame Rate (fps)
20000	GIF	Same as source	2
20001	WEBP	Same as source	2

Preset time point screenshot templates

Template ID	Format	Width	Height	Fill Mode
10	JPG	Same as source	Same as source	Stretch

Preset sampled screenshot templates

Template ID	Format	Width	Height	Interval Measurement	Interval	Fill Mode
10	JPG	Same as source	Same as source	By percent	10%	Stretch

Preset image sprite templates

Template ID	Format	Subimage Width	Subimage Height	Subimage Rows	Subimage Columns	Interval Measurement	Interval (seconds)
10	JPG	142	80	10	10	By time	10

Preset adaptive bitrate streaming templates

Adaptive Bitrate Streaming Video Format

Template ID	Package Type	Encryption Type	Stream Info	Disable Low-Res to High-Res Conversion
10	HLS	Not encrypted	Contains video streams of six specifications from "Smooth" to "4K".	Yes
12	HLS	SimpleAES	Contains video streams of six specifications from "Smooth" to "4K".	Yes
20	MPEG-DASH	Not encrypted	Contains video streams of six specifications from "Smooth" to "4K".	No

Adaptive Bitrate Streaming Substream information

Stream Clarity	Video Parameters				Audio Parameters			
	Resolution	Bitrate	Frame Rate (fps)	Codec	Bitrate	Sample Rate	Sound Channels	Codec

Smooth	Vertical: 240; horizontal: proportionally scaled	256 Kbps	24	H.264	48 Kbps	44,100 Hz	Stereo	AAC
SD	Vertical: 480; horizontal: proportionally scaled	512 Kbps	24	H.264	48 Kbps	44,100 Hz	Stereo	AAC
HD	Vertical: 720; horizontal: proportionally scaled	1024 Kbps	24	H.264	48 Kbps	44,100 Hz	Stereo	AAC
FHD	Vertical: 1080; horizontal: proportionally scaled	2,500 Kbps	24	H.264	48 Kbps	44,100 Hz	Stereo	AAC
2K	Vertical: 1440; horizontal: proportionally scaled	3,072 Kbps	24	H.264	48 Kbps	44,100 Hz	Stereo	AAC
4K	Vertical: 2160; horizontal: proportionally scaled	6,144 Kbps	24	H.264	48 Kbps	44,100 Hz	Stereo	AAC

Adaptive Bitrate Streaming Audio Format

Template ID	PackageType	SegmentType	EncryptionType	SubstreamInfo
1010	HLS	ts	No Encryption	Contains audio streams of 4 specifications from "Smooth" to "FHD".
1100	HLS	fmp4	No Encryption	Contains audio streams of 4 specifications from "Smooth" to "FHD".

Adaptive Bitrate Streaming Substream Information

Stream Clarity	Audio Parameters			
	Bitrate	SampleRate	SoundSystem	Codec
Smooth	48kbps	44100Hz	Stereo	AAC
SD	96kbps	44100Hz	Stereo	AAC
HD	128kbps	44100Hz	Stereo	AAC
FHD	192kbps	44100Hz	Stereo	AAC

Media AI

Preset parameter templates for media AI:

Preset audio/video moderation templates

Preset audio/video content analysis templates

Preset audio/video content recognition templates

Preset audio/video moderation templates

Template ID	Porn	Terror	Moan
10	Yes	Yes	Yes

Preset audio/video content analysis templates

Template ID	Intelligent Classification	Intelligent Labeling	Intelligent Thumbnail Generation	Intelligent Labeling by Frame
10	Yes	Yes	Yes	No
20	Yes	Yes	Yes	Yes

Preset audio/video content recognition templates

Template ID	Face Recognition	Speech Full Text Recognition (AsrFullText)	Speech Translation Recognition (AsrTranslate)	Speech Keyword Recognition (AsrWords)	Full Text Recognition (OcrFullText)	Text Keyword Recognition (OcrWords)
10	Yes (Use	No	No	No	No	No

	Default Character Library)					
111	No	Yes (Source Language of Audio: Chinese, Generate .vtt Subtitles)	No	No	No	No
112	No	Yes (Source Language of Audio: English, Generate .vtt Subtitles)	No	No	No	No
113	No	Yes (Source Language of Audio: Japanese, Generate .vtt Subtitles)	No	No	No	No
121	No	No	Yes (Source Language of Audio: Chinese, Target Language for Translation: English, Generate .vtt Subtitles)	No	No	No
122	No	No	Yes (Source Language of Audio: English, Target	No	No	No

			Language for Translation: Chinese, Generate .vtt Subtitles)			
123	No	No	Yes (Source Language of Audio: Malay, Target Language for Translation: English, Generate .vtt Subtitles)	No	No	No
124	No	No	Yes (Source Language of Audio: English, Target Language for Translation: Malay, Generate .vtt Subtitles)	No	No	No
125	No	No	Yes (Source Language of Audio: Chinese, Target Language for Translation: Japanese, Generate .vtt Subtitles)	No	No	No
126	No	No	Yes (Source Language of Audio: Chinese,	No	No	No

			Target Language for Translation: Korean, Generate .vtt Subtitles)			

Legacy Transcoding

Legacy preset transcoding templates

Video transcoding

Clarity	Template ID	Format	Video Parameters				Audio Parameters
			Resolution	Bitrate	Frame Rate (fps)	Codec	Codec
Smooth	10	MP4	Horizontal: 320; vertical: proportionally scaled	256 Kbps	24	H.264	AAC
	510	MP4	Vertical: 240; horizontal: proportionally scaled	250 Kbps	15	H.265	AAC
	210	HLS	Horizontal: 320; vertical: proportionally scaled	256 Kbps	24	H.264	AAC
	610	HLS	Vertical: 240; horizontal: proportionally scaled	250 Kbps	15	H.265	AAC
	10046	FLV	Horizontal: 320; vertical: proportionally scaled	256 Kbps	24	H.264	MP3
	710	FLV	Vertical: 240; horizontal: proportionally scaled	250 Kbps	15	H.265	AAC
SD	20	MP4	Horizontal: 640;	512	24	H.264	AAC

			vertical: proportionally scaled	Kbps			
	520	MP4	Vertical: 480; horizontal: proportionally scaled	600 Kbps	24	H.265	AAC
	220	HLS	Horizontal: 640; vertical: proportionally scaled	512 Kbps	24	H.264	AAC
	620	HLS	Vertical: 480; horizontal: proportionally scaled	600 Kbps	24	H.265	AAC
	10047	FLV	Horizontal: 640; vertical: proportionally scaled	512 Kbps	24	H.264	MP3
	720	FLV	Vertical: 480; horizontal: proportionally scaled	600 Kbps	24	H.265	AAC
HD	30	MP4	Horizontal: 1280; vertical: proportionally scaled	1,024 Kbps	24	H.264	AAC
	530	MP4	Vertical: 720; horizontal: proportionally scaled	800 Kbps	25	H.265	AAC
	230	HLS	Horizontal: 1280; vertical: proportionally scaled	1,024 Kbps	24	H.264	AAC
	630	HLS	Vertical: 720; horizontal: proportionally scaled	800 Kbps	25	H.265	AAC
	10048	FLV	Horizontal: 1280; vertical: proportionally scaled	1,024 Kbps	24	H.264	MP3
	730	FLV	Vertical: 720; horizontal: proportionally scaled	800 Kbps	25	H.265	AAC
FHD	40	MP4	Horizontal: 1920;	2,500	24	H.264	AAC

			vertical: proportionally scaled	Kbps			
	540	MP4	Vertical: 1080; horizontal: proportionally scaled	1,400 Kbps	30	H.265	AAC
	240	HLS	Horizontal: 1920; vertical: proportionally scaled	2,500 Kbps	24	H.264	AAC
	640	HLS	Vertical: 1080; horizontal: proportionally scaled	1,400 Kbps	30	H.265	AAC
	10049	FLV	Horizontal: 1920; vertical: proportionally scaled	2,500 Kbps	24	H.264	MP3
	740	FLV	Vertical: 1080; horizontal: proportionally scaled	1,400 Kbps	30	H.265	AAC
2K	70	MP4	Vertical: 1440; horizontal: proportionally scaled	3,072 Kbps	30	H.264	AAC
	570	MP4	Vertical: 1440; horizontal: proportionally scaled	2,048 Kbps	30	H.265	AAC
	270	HLS	Vertical: 1440; horizontal: proportionally scaled	3,072 Kbps	30	H.264	AAC
	670	HLS	Vertical: 1440; horizontal: proportionally scaled	2,048 Kbps	30	H.265	AAC
	370	FLV	Vertical: 1440; horizontal: proportionally scaled	3,072 Kbps	30	H.264	MP3
	770	FLV	Vertical: 1440; horizontal: proportionally scaled	2,048 Kbps	30	H.265	AAC
4K	80	MP4	Vertical: 2160;	6,144	30	H.264	AAC

			horizontal: proportionally scaled	Kbps			
	580	MP4	Vertical: 2160; horizontal: proportionally scaled	4,096 Kbps	30	H.265	AAC
	280	HLS	Vertical: 2160; horizontal: proportionally scaled	6,144 Kbps	30	H.264	AAC
	680	HLS	Vertical: 2160; horizontal: proportionally scaled	4,096 Kbps	30	H.265	AAC
	380	FLV	Vertical: 2160; horizontal: proportionally scaled	6,144 Kbps	30	H.264	MP3
	780	FLV	Vertical: 2160; horizontal: proportionally scaled	4,096 Kbps	30	H.265	AAC

Parameters not listed in the above table are the same for all preset templates:

Category	Parameter	Description
Video Parameters	Profile	If Codec is H.264, Profile is High If Codec is H.265, Profile is Main
	GOP length	240 frames
	Color space	YUV420p
	Bitrate control method	VBR
Audio Parameters	Sample rate	44,100 Hz
	Bitrate	48 Kbps
	Sound channels	Stereo

Video Clipping
Video Editing

Last updated：2020-12-09 15:24:13

Video editing is an offline task that clips and splices videos in VOD. Specifically, it includes the following features:
Video clipping: this refers to clipping a file in VOD to generate a new video.
Video splicing: this refers to splicing multiple files in VOD to generate a new video.
Video clip splicing: this refers to clipping multiple files in VOD and then splicing the clips to generate a new video.
Live stream transcoding: this refers to transcoding a stream in VOD to generate a new video.
Live stream clipping: this refers to clipping a stream in VOD to generate a new video.
Live stream splicing: this refers to splicing multiple streams in VOD to generate a new video.
Live stream clip splicing: this refers to clipping multiple streams in VOD and then splicing the clips to generate a new video.

Note:
If you want to clip, splice, or perform other operations on a live stream, please be sure to manipulate it after it ends; otherwise, the generated video may be incomplete.
The container format of the generated video is MP4. When initiating a video editing task, you can specify whether to perform a task flow on the new video.

Task Initiation

You can initiate a video editing task by calling a server API. The return result of the API contains the task ID, which is used to associate with the corresponding task result when getting result.

Result Getting

After initiating an editing task, you can wait for result notification asynchronously or perform task query synchronously to get the task execution result. Below is an example of getting the result notification in normal callback mode after the editing task is initiated (the fields with null value are omitted):

```
{
    "EventType":"EditMediaComplete",
    "EditMediaCompleteEvent":{
        "TaskId":"EditMedia-f5ac8127b3b6b85cdc13f237c6005d8",
        "Status":"FINISH",
        "ErrCode":0,
```

```
        "Message":"SUCCESS",
        "Input":{
            "InputType":"File",
            "FileInfoSet":[
                {
                    "FileId":"24961954183381008",
                    "StartTimeOffset":0,
                    "EndTimeOffset":300
                },
                {
                    "FileId":"24961954183381009",
                    "StartTimeOffset":0,
                    "EndTimeOffset":300
                },
                {
                    "FileId":"24961954183381010",
                    "StartTimeOffset":0,
                    "EndTimeOffset":300
                }
            ]
        },
        "Output":{
            "FileType":"mp4",
            "FileId":"24961954183923290",
            "FileUrl":"http://125676836723.vod2.myqcloud.com/xxx/xxx/f0.mp4"
        },
        "ProcedureTaskId":""
    }
}
```


In the callback result, `Input.InputType` is `File`, indicating that the type of the edited video is of a file type. `Input.FileInfoSet` contains three elements, of which `StartTimeOffset` is `0` and `EndTimeOffset` is `300`, indicating to clip the first 5 minutes of each of the three videos and then splice them into a 15-minute video. `Output.FileId` is the `FileId` of the generated video, whose playback URL is the value in `FileUrl`.

Video Compositing

Last updated：2022-12-30 16:30:24

Video compositing is an offline task that performs a series of complicated operations on a video in VOD such as clipping, splicing, overlaying, and flipping. It can achieve the following effects:
Rotation: rotates videos or images by certain degrees or in a certain direction.
Audio control: turns up/down sound volume in videos/audios or mutes videos.
Overlaying: overlays videos/images in sequence to achieve effects such as picture-in-picture.
Audio mixing: mixes the sound in videos/audios.
Audio extraction: extracts sound from videos (without retaining the image).
Clipping: clips segments within the specified period of time out of videos/audios.
Splicing: splices videos/audios/images in chronological order.
Transition: adds transition effects between segments during video or image splicing.
Speed adjustment: Adjusts the playback speed of the video or audio material.
The container format of the media file after compositing is MP4 (video) or MP3 (audio).

Initiating a Task

You can initiate a video compositing task by calling a server API. The return result of the API contains the task ID, which is used to associate with the corresponding task result when getting result.

Getting the Result

After initiating a compositing task, you can wait for result notification asynchronously or perform task query synchronously to get the task execution result. Below is an example of getting the result notification in normal callback mode after the video compositing task is initiated (the fields with null value are omitted):

```
{
    "EventType": "ComposeMediaComplete",
    "ComposeMediaCompleteEvent": {
        "TaskId": "ComposeMedia-f5ac8127b3b6b85cdc13f237c6005d8",
        "Status": "FINISH",
        "ErrCode": 0,
        "Message": "SUCCESS",
        "Input":{
            "Tracks": [{
                    "Type": "Video",
                    "TrackItems": [{
```

```
                        "Type": "Video",
                        "SourceMedia": "5285485487985271487",
                        "AudioOperations": [{
                            "Type": "Volume",
                            "VolumeParam": {
                                "Mute": 1
                            }
                        }]
                    }]
                },
                {
                    "Type": "Audio",
                    "TrackItems": [{
                            "Type": "Empty",
                            "EmptyItem": {
                                "Duration": 5
                            }
                        },
                        {
                            "Type": "Audio",
                            "AudioItem": {
                                "SourceMedia": "5285485487985271488",
                                "Duration": 15
                            }
                        },
                        {
                            "Type": "Audio",
                            "AudioItem": {
                                "SourceMedia": "5285485487985271489",
                                "SourceMediaStartTime": 2,
                                "Duration": 14
                            }
                        }
                    ]
                }
            ],
            "Output": {
                "FileName": "Video compositing effect test",
                "Container": "mp4"
            }
        },
        "Output": {
            "FileType": "mp4",
            "FileId": 5285485487985271490,
            "FileUrl": "http://125676836723.vod2.myqcloud.com/xxx/xxx/xxx.mp4"
        }
    }
```

```
}
```

In the callback result, `Input.Tracks` contains two elements in `Type` of `Video` and `Audio`, indicating the composited video contains a video track and an audio track.

Video track: the ID of the source video is `5285485487985271487`, and the video is muted.

Audio track: it includes 5 seconds of silence and two voiceover bits lasting 15s and 14s, respectively.

`Output.FileId` is the `FileId` of the new video generated after video compositing, and the playback URL is the value in `FileUrl`.

Video Conversion
Transcoding

Last updated：2024-09-03 11:02:49

Transcoding is an offline task that converts the source audio/video bitstream. It changes parameters of the source bitstream, such as codec, resolution, and bitrate, to adapt it to different devices and network conditions. The following benefits can be achieved with transcoding:
Compatible with multiple clients: A source video can be transcoded to formats (.mp4 for example) that are compatible with more types of devices for smooth playback.
Adapt to different bandwidths: a source video can be transcoded for output in multiple definitions such as smooth, SD, HD, and FHD. End users can select the appropriate bitrate depending on their network conditions.
Improved playback efficiency: The moov atom can be moved from the end of an MP4 file to its beginning, so the video can be played before it is entirely downloaded.
Watermarking: you can add a watermark to a video to mark ownership or copyright. For more information, please see Watermarking.
Reduce bandwidth usage: use advanced encoding modes (H.265 for example) for transcoding to reduce the bitrate of a video substantially with the original quality retained, thus lowering the payback bandwidth usage.
After a video is transcoded, the playback URL of the output video can be obtained according to Getting the Result.
You can use your own player or a third-party player to play back the output video.

Note:

The transcoding feature is mainly suitable for **UGSV** scenarios. For **long video** scenarios (video websites, online education, etc.), adaptive bitrate streaming can deliver a better user experience.

Transcoding Template

The target specification of an output video after transcoding is specified by parameters such as codec, resolution, and bitrate. VOD integrates these parameters in the transcoding template as shown below:

Note:

For more audio/video transcoding types, see Supported transcoding types.

Type	Parameter	Description
Muxing	Container format	Supported video and audio container formats for transcoding:Video: MP4, TS, HLS, and FLVAudio: MP3, M4A, FLAC, and Ogg
	Deleting video stream	If this is enabled, the output video will contain only the audio stream with no video stream.

	Deleting audio stream	If this is enabled, the output video will contain only the video stream with no audio stream.
Video codec	Codec	H.264 and H.265 are supported
	Bitrate	Supported bitrate range: 10 Kbps - 35 Mbps
	Frame Rate	Supported frame rate range: 1-60 fps; common values: 24, 25, and 30
	Resolution	Supported width range: 128-4096 pxSupported height range: 128-4096 px
	GOP length	Supported GOP length range: 1-10s
	Profile	When the video codec is H.264, the baseline, main, and high profiles are supported.When the video codec is H.265, only the main profile is supported.
	Color Space	YUV420p is supported.
Audio codec	Codec	MP3, AAC, AC3, and FLAC are supported
	Sample rate	The following audio sample rates are supported:34000 Hz44100 Hz48000 Hz
	Bitrate	Supported bitrate range: 26-256 Kbps, including the following values:48 Kbps64 Kbps128 Kbps
	Channel	MonoDualStereo

VOD provides a List of Preset Parameter Templates for common transcoding specifications. You can also create and manage custom transcoding templates on the console (see Template Settings for detailed directions) or through server API.

Initiating a Task

There are three ways to initiate a transcoding task, namely, directly initiating through server API, directly initiating through the console, and specifying a task upon upload. For more information, please see Video Processing Task System for video processing.

Methods of initiating transcoding tasks:

Call the server API ProcessMedia to initiate a task: specify the transcoding template ID in the `MediaProcessTask.TranscodeTaskSet` parameter in the request.

Initiate a task on a video through the console: add a task flow in the console, set the specifications of transcoding output in it, and use it to initiate video processing.

Specify a task upon upload from server: add a task flow in the console, set the specifications of transcoding output in it, and specify it as the `procedure` parameter in the ApplyUpload API.
Specify a task upon upload from client: add a task flow in the console, set the specifications of transcoding output in it, and specify it as the `procedure` parameter in the signature for upload from client.
Upload through console: add a task flow in the console, set the specifications of transcoding output in it, upload a video through the console, select Process Video During Upload, and specify to execute this task flow upon video upload completion.

Getting the Result

After initiating a transcoding task, you can wait for result notification asynchronously or perform task query synchronously to get the task execution result. Below is an example of getting the result notification in normal callback (the fields with null value are omitted):


```
{
    "EventType":"ProcedureStateChanged",
    "ProcedureStateChangeEvent":{
        "TaskId":"1256768367-Procedure-2e1af2456351812be963e309cc133403t0",
        "Status":"FINISH",
        "FileId":"5285890784246869930",
        "FileName":"Animal World",
        "FileUrl":"http://1256768367.vod2.myqcloud.com/xxx/xxx/AtUCmy6gmIYA.mp4",
        "MetaData":{
            "AudioDuration":60,
            "AudioStreamSet":[
                {
                    "Bitrate":383854,
                    "Codec":"aac",
                    "SamplingRate":48000
                }
            ],
            "Bitrate":1021028,
            "Container":"mov,mp4,m4a,3gp,3g2,mj2",
            "Duration":60,
            "Height":480,
            "Rotate":0,
            "Size":7700180,
            "VideoDuration":60,
            "VideoStreamSet":[
                {
                    "Bitrate":637174,
                    "Codec":"h264",
                    "Fps":23,
```

```
                "Height":480,
                "Width":640
            }
        ],
        "Width":640
    },
    "MediaProcessResultSet":[
        {
            "Type":"Transcode",
            "TranscodeTask":{
                "Status":"SUCCESS",
                "ErrCode":0,
                "Message":"",
                "Input":{
                    "Definition":220
                },
                "Output":{
                    "Url":"http://1256768367.vod2.myqcloud.com/xxx/xxx/v.f20.m3
                    "Size":63120997,
                    "Container":"mov,mp4,m4a,3gp,3g2,mj2",
                    "Height":480,
                    "Width":640,
                    "Bitrate":513402,
                    "Md5":"084d403c73930ca2f835679af1f37bd3",
                    "Duration":60,
                    "VideoStreamSet":[
                        {
                            "Bitrate":473101,
                            "Codec":"h264",
                            "Fps":24,
                            "Height":480,
                            "Width":640
                        }
                    ],
                    "AudioStreamSet":[
                        {
                            "Bitrate":48581,
                            "Codec":"aac",
                            "SamplingRate":44100
                        }
                    ],
                    "Definition":220
                }
            }
        }
    ],
    "TasksPriority":0,
```

```
        "TasksNotifyMode":""
    }
}
```

In the callback result, `ProcedureStateChangeEvent.MediaProcessResultSet` contains the transcoding result with `Type` as `Transcode` and `Definition` as `220`.

Watermarking

Last updated：2020-04-02 17:07:23

Watermarking is an offline task that adds an image or text at the specified position of the video during video transcoding or screencapturing. VOD supports the following types of watermarks:
Static image watermark: this refers to an image watermark in PNG format. It can be a copyright owner's or TV station's logo and is generally used to indicate the video copyright ownership.
Animated image watermark: this refers to an image watermark in APNG format, which can be animated.
Text watermark: this refers to a multi-lingual text watermark. It can be a user's nickname and is generally used to identify the producer of UGSV content.
VOD can add multiple watermarks to a video or screenshot. The size and position can be customized individually.

Watermarking Template

The target specification of a watermark is subject to parameters such as watermark type, width, height, and position, which can be customized in the form of VOD watermarking template as shown below:

Parameter	Description
Type	Image and text watermarks are supported: Image watermark: Static or animated images are supported. Text watermark: Texts in various languages are supported.
Position	Relative position of a watermark in the video.
ImageSize	Size of a watermark in the video.
ImageContent	Binary content of a watermark.
FontSize	Font size of a text watermark.
FontType	Font of a text watermark, e.g., Times New Roman.
FontColor	Color of a text watermark, e.g., 0xRRGGBB.
FontAlpha	Transparency of text watermark. Value range: 0–100%.

You can use the console (for detailed directions, please see Template Settings) or call a server API to create and manage custom watermarking templates.

Task Initiation

There are three ways to initiate a transcoding task with watermark, namely, directly initiating through server API, directly initiating through the console, and specifying a task upon upload. For more information, please see Task Initiation for video processing.

Below are instructions for initiating transcoding tasks with watermark in these ways:

Call the server API ProcessMedia to initiate a task: specify the watermarking template ID in the `MediaProcessTask.TranscodeTaskSet` parameter in the request.

Initiate a task on a video through the console: add a task flow in the console, set the watermark specification in the task flow, and use the task flow to initiate video processing.

Specify a task upon upload from server: add a task flow in the console, set the target watermark specification in the task flow, and specify this task flow as the `procedure` in the ApplyUpload request.

Specify a task upon upload from client: add a task flow in the console, set the target watermark specification in the task flow, and specify this task flow as the `procedure` parameter in the signature for upload from client.

Upload through console: add a task flow in the console, set the target watermark specification in the task flow, upload a video through the console, select Process Video During Upload, and specify to execute this task flow upon video upload completion.

Getting Result

After initiating a transcoding task with watermark, you can wait for result notification asynchronously or perform task query synchronously to get the task execution result. Below is an example of getting the result notification in normal callback mode after the transcoding task with watermark is initiated (the fields with null value are omitted):

```
{
    "EventType":"ProcedureStateChanged",
    "ProcedureStateChangeEvent":{
        "TaskId":"1256768367-Procedure-2e1af2456351812be963e309cc133403t0",
        "Status":"FINISH",
        "FileId":"5285890784246869930",
        "FileName":"Animal World",
        "FileUrl":"http://1256768367.vod2.myqcloud.com/xxx/xxx/AtUCmy6gmIYA.mp4",
        "MetaData":{
            "AudioDuration":60,
            "AudioStreamSet":[
                {
                    "Bitrate":383854,
                    "Codec":"aac",
                    "SamplingRate":48000
                }
```

```
            ],
            "Bitrate":1021028,
            "Container":"mov,mp4,m4a,3gp,3g2,mj2",
            "Duration":60,
            "Height":480,
            "Rotate":0,
            "Size":7700180,
            "VideoDuration":60,
            "VideoStreamSet":[
                {
                    "Bitrate":637174,
                    "Codec":"h264",
                    "Fps":23,
                    "Height":480,
                    "Width":640
                }
            ],
            "Width":640
        },
        "MediaProcessResultSet":[
            {
                "Type":"Transcode",
                "TranscodeTask":{
                    "Status":"SUCCESS",
                    "ErrCode":0,
                    "Message":"",
                    "Input":{
                        "Definition":220,
                        "WatermarkSet": [
                            {
                                "Definition": 23120
                            }
                        ]
                    },
                    "Output":{
                        "Url":"http://1256768367.vod2.myqcloud.com/xxx/xxx/v.f20.m3
                        "Size":63120997,
                        "Container":"mov,mp4,m4a,3gp,3g2,mj2",
                        "Height":1086,
                        "Width":1920,
                        "Bitrate":513402,
                        "Md5":"084d403c73930ca2f835679af1f37bd3",
                        "Duration":60,
                        "VideoStreamSet":[
                            {
                                "Bitrate":473101,
                                "Codec":"h264",
```

```
                                "Fps":24,
                                "Height":480,
                                "Width":640
                            }
                        ],
                        "AudioStreamSet":[
                            {
                                "Bitrate":48581,
                                "Codec":"aac",
                                "SamplingRate":44100
                            }
                        ],
                        "Definition":220
                    }
                }
            }
        ],
        "TasksPriority":0,
        "TasksNotifyMode":""
    }
}
```


In the callback result, `ProcedureStateChangeEvent.MediaProcessResultSet` contains the transcoding result in `Type` of `Transcode` : the transcoding specification `Definition` is 220, and a watermark is added during transcoding, whose specification `Definition` is 23120.

Screenshots

Last updated：2023-03-07 11:20:50

Screenshot taking is an offline task that takes screenshots of a video at specified times. VOD supports the following types of screenshots:
Time point screenshot: Takes screenshots at specified time points.
Sampled screenshot: Takes screenshots at regular intervals.
Thumbnail: Takes a screenshot at a specified time point and uses it as the video's thumbnail.
Image sprite: Takes multiple screenshots at the specified interval and combines them into one image (image sprite).
Common use cases include the following:
Video thumbnail generation: Take a screenshot of a video and use it as the video's thumbnail.
Image sprite generation: Generate an image sprite (a collection of small images), which is often used as the summary of a video.
Preview: Use image sprites and VTT files to show previews above the progress bar.

Screenshot Templates

A screenshot template is a collection of screenshot parameters, including the output file format and image dimensions.

Time point screenshot templates

You can use a time point screenshot template to take a screenshot at a specific time point or generate a thumbnail.

Parameter	Description
Format	The screenshot format (only JPG is supported currently).
Width	The screenshot width (px). Value range: 128-4096.
Height	The screenshot height (px). Value range: 128-4096.
FillType	The fill mode (`FillType`) specifies how the source video is processed when its aspect ratio does not match the output aspect ratio. The following fill modes are supported: Stretch: The source video is stretched to match the output aspect ratio. This may cause the video to appear distorted. Fill with black: The original aspect ratio is retained, leaving black bars. Fill with white: The original aspect ratio is retained, leaving white bars. Gaussian blur: The original aspect ratio is retained, and Gaussian blur is applied to the blank spaces.

VOD provides preset time point screenshot templates for common parameter combinations. You can also create your own templates and manage them in the console. For details, see Template Settings.

Sampled screenshot templates

You can use a sampled screenshot template to take screenshots at regular intervals.

Parameter	Description
Format	The screenshot format (only JPG is supported currently).
Width	The screenshot width (px). Value range: 128-4096.
Height	The screenshot height (px). Value range: 128-4096.
SampleType	How sampling intervals are measured. Sampling intervals can be measured in two ways: By percent: Intervals are measured by percent. For example, if Interval is set to 5 (%), 20 screenshots will be generated for a video. By time: Intervals are measured by time. For example, if Interval is set to 10 (sec), the number of screenshots generated will depend on the video length.
Interval	The sampling interval. If the interval measurement (SampleType) is by percent, this parameter is a percent value. If interval measurement is by time, this parameter is a time value (sec).
FillType	The fill mode (`FillType`) specifies how the source video is processed when its aspect ratio does not match the output aspect ratio. The following fill modes are supported: Stretch: The source video is stretched to match the output aspect ratio. This may cause the video to appear distorted. Black-leaving: The original aspect ratio is retained, leaving black bars. Blank-leaving: The original aspect ratio is retained, leaving blank spaces. Gaussian blur: The original aspect ratio is retained, and Gaussian blur is applied to the blank spaces.

VOD provides preset sampled screenshot templates for common parameter combinations. You can also create your own templates and manage them in the console. For details, see Template Settings.

Image sprite screenshot templates

You can use an image sprite screenshot template to generate image sprites.

Parameter	Description
Format	The format of the image sprite (only JPG is supported currently).
Width	The width of the subimage in an image sprite.
Height	The height of the subimage in an image sprite.

Rows	The number of image rows in a sprite.
Columns	The number of image columns in a sprite.
SampleType	How sampling intervals are measured. Currently, only sampling by time is supported.
Interval	The time interval for image sampling.

Note:

The result of multiplying `Width` x `Columns` (i.e., sprite width) should be within the range of 128-4096.

The result of multiplying `Height` x `Rows` (i.e., sprite height) should be in the range of 128-4096.

VOD provides preset image sprite screenshot templates for common parameter combinations. You can also create your own templates and manage them in the console. For details, see Template Settings.

Initiating a Screenshot Task

You can initiate a screenshot taking task by calling a server API, via the console, or by specifying the task when uploading videos. For details, see Task Initiation.

Specifically, you can initiate a screenshot task by doing one of the following:

Call the server API ProcessMedia, specifying the ID of the screenshot template in `MediaProcessTask.SnapshotByTimeOffsetTaskSet`.

Add a task flow in the console, specifying the screenshot parameters, and use the task flow to process videos in the console.

Add a task flow in the console, specifying the screenshot parameters. When uploading a video from the server, set the `procedure` parameter to the task flow you created.

Add a task flow in the console, specifying the screenshot parameters. When uploading a video from a client, set `procedure` in the upload signature to the task flow you created.

Add a task flow in the console, specifying the screenshot parameters. When uploading a video via the console, choose Auto-processing after upload and select the task flow you created.

Getting the Result

After initiating a screenshot task, you can wait for the result notification asynchronously or perform a task query synchronously to get the task execution result. Below is an example of the notification received in normal callback mode after a screenshot task is initiated (the fields with null value are omitted):

```
{
    "EventType":"ProcedureStateChanged",
```

```
    "ProcedureStateChangeEvent":{
        "TaskId":"1256768367-Procedure-2e1af2456351812be963e309cc133403t0",
        "Status":"FINISH",
        "FileId":"5285890784246869930",
        "FileName":"Animal World",
        "FileUrl":"http://1256768367.vod2.myqcloud.com/xxx/xxx/AtUCmy6gmIYA.mp4",
        "MetaData":{
            "AudioDuration":60,
            "AudioStreamSet":[
                {
                    "Bitrate":383854,
                    "Codec":"aac",
                    "SamplingRate":48000
                }
            ],
            "Bitrate":1021028,
            "Container":"mov,mp4,m4a,3gp,3g2,mj2",
            "Duration":60,
            "Height":480,
            "Rotate":0,
            "Size":7700180,
            "VideoDuration":60,
            "VideoStreamSet":[
                {
                    "Bitrate":637174,
                    "Codec":"h264",
                    "Fps":23,
                    "Height":480,
                    "Width":640
                }
            ],
            "Width":640
        },
        "MediaProcessResultSet":[
            {
                "Type":"SnapshotByTimeOffset",
                "SnapshotByTimeOffsetTask":{
                    "Status":"SUCCESS",
                    "ErrCode":0,
                    "Message":"",
                    "Input":{
                        "Definition":10,
                        "Definition":[3, 6, 9]
                    },
                    "Output":{
                        "Definition":10,
                        "PicInfoSet":[
```

```
                        {
                            "TimeOffset":3,
                            "Url":"http://1256768367.vod2.myqcloud.com/xxx/xxx/
                        },
                        {
                            "TimeOffset":6,
                            "Url":"http://1256768367.vod2.myqcloud.com/xxx/xxx/
                        },
                        {
                            "TimeOffset":9,
                            "Url":"http://1256768367.vod2.myqcloud.com/xxx/xxx/
                        }
                    ]
                }
            }
        },
        {
            "Type":"SampleSnapshot",
            "SampleSnapshotTask":{
                "Status":"SUCCESS",
                "ErrCode":0,
                "Message":"",
                "Input":{
                    "Definition":10
                },
                "Output":{
                    "Definition":10,
                    "SampleType": "Percent",
                    "Interval": 10,
                    "WaterMarkDefinition": [],
                    "ImageUrlSet":[
                            "http://1256768367.vod2.myqcloud.com/xxx/xxx/xxx1.j
                            "http://1256768367.vod2.myqcloud.com/xxx/xxx/xxx2.j
                            "http://1256768367.vod2.myqcloud.com/xxx/xxx/xxx3.j
                            "http://1256768367.vod2.myqcloud.com/xxx/xxx/xxx4.j
                            "http://1256768367.vod2.myqcloud.com/xxx/xxx/xxx5.j
                            "http://1256768367.vod2.myqcloud.com/xxx/xxx/xxx6.j
                            "http://1256768367.vod2.myqcloud.com/xxx/xxx/xxx7.j
                            "http://1256768367.vod2.myqcloud.com/xxx/xxx/xxx8.j
                            "http://1256768367.vod2.myqcloud.com/xxx/xxx/xxx9.j
                    ]
                }
            }
        },
        {
            "Type":"ImageSprites",
            "ImageSpriteTask":{
```

```
                    "Status":"SUCCESS",
                    "ErrCode":0,
                    "Message":"",
                    "Input":{
                        "Definition":10
                    },
                    "Output":{
                        "Definition":10,
                        "Height":80,
                        "Width":142,
                        "TotalCount":1,
                        "ImageUrlSet":[
                            "http://1256768367.vod2.myqcloud.com/xxx/xxx/xxx1.jpg"
                        ],
                        "WebVttUrl":"http://1256768367.vod2.myqcloud.com/xxx/xxx/xx
                    }
                }
            },
            {
                "Type":"CoverBySnapshot",
                "CoverBySnapshotTask":{
                    "Status":"SUCCESS",
                    "ErrCode":0,
                    "Message":"",
                    "Input":{
                        "Definition":10,
                        "PositionType":"Time",
                        "PositionValue":0
                    },
                    "Output":{
                        "CoverUrl":"http://1256768367.vod2.myqcloud.com/xxx/xxx/xxx
                    }
                }
            }
        ],
        "TasksPriority":0,
        "TasksNotifyMode":""
    }
}
```


In the above callback, `ProcedureStateChangeEvent.MediaProcessResultSet` contains four types of results, namely `SnapshotByTimeOffset`, `SampleSnapshot`, `ImageSprites`, and `CoverBySnapshot`, which represent a time point screenshot task, a sampled screenshot task, an image sprite screenshot task, and a thumbnail generation task respectively.

Animated Image Generating

Last updated：2020-11-04 14:24:21

Animated image generating is an offline task that converts a video to an animated image such as in GIF or WEBP format. An animated image is a seamless cycle of continuous frames, which can deliver an animation effect with a small file size.

Note:

When animated image generating is supported, you can specify the start time and end time in the original video, and then the segment will be captured to generate an animated image.

Animated Image Generating Template

The target specification of an animated image is subject to parameters such as animated image file format, width, height, and frame rate, which can be customized in the form of VOD animated image generating template as shown below:

Parameter	Description
Format	Output format of an animated image file. Currently, only GIF and WEBP are supported.
Width	Animated image width. Value range: 128–4,096 px.
Height	Animated image height. Value range: 128–4,096 px.
FPS	Supported frame rate range: 1–60 fps.

For common specifications, VOD provides a preset animated image generating template. In addition, you can also create and manage custom animated image generating templates in the console. For detailed directions, please see Template Settings.

Task Initiation

There are three ways to initiate an animated image generating task, namely, directly initiating through server API, directly initiating through the console, and specifying a task upon upload. For more information, please see Task Initiation for video processing.

Below are instructions for initiating animated image generating tasks in these ways:

Call the server API ProcessMedia to initiate a task: specify the animated image generating template ID in the `MediaProcessTask.AnimatedGraphicTaskSet` parameter in the request.

Initiate a task on a video through the console: add a task flow in the console, set the target animated image specification in the task flow, and use the task flow to initiate video processing.
Specify a task upon upload from server: add a task flow in the console, set the target animated image specification in the task flow, and specify this task flow as the `procedure` in the ApplyUpload request.
Specify a task upon upload from client: add a task flow in the console, set the target animated image specification in the task flow, and specify this task flow as the `procedure` parameter in the signature for upload from client.
Upload through console: add a task flow in the console, set the target animated image specification in the task flow, upload a video through the console, select **Process Video During Upload**, and specify to execute this task flow upon video upload completion.

Getting Result

After initiating an animated image generating task, you can wait for result notification asynchronously or perform task query synchronously to get the task execution result. Below is an example of getting the result notification in normal callback mode after the animated image generating task is initiated (the fields with null value are omitted):


```
{
    "EventType":"ProcedureStateChanged",
    "ProcedureStateChangeEvent":{
        "TaskId":"1256768367-Procedure-2e1af2456351812be963e309cc133403t0",
        "Status":"FINISH",
        "FileId":"5285890784246869930",
        "FileName":"Animal World",
        "FileUrl":"http://1256768367.vod2.myqcloud.com/xxx/xxx/AtUCmy6gmIYA.mp4",
        "MetaData":{
            "AudioDuration":60,
            "AudioStreamSet":[
                {
                    "Bitrate":383854,
                    "Codec":"aac",
                    "SamplingRate":48000
                }
            ],
            "Bitrate":1021028,
            "Container":"mov,mp4,m4a,3gp,3g2,mj2",
            "Duration":60,
            "Height":480,
            "Rotate":0,
            "Size":7700180,
            "VideoDuration":60,
            "VideoStreamSet":[
                {
```

```
                    "Bitrate":637174,
                    "Codec":"h264",
                    "Fps":23,
                    "Height":480,
                    "Width":640
                }
            ],
            "Width":640
        },
        "MediaProcessResultSet":[
            {
                "Type":"AnimatedGraphics",
                "AnimatedGraphicTask":{
                    "Status":"SUCCESS",
                    "ErrCode":0,
                    "Message":"",
                    "Input":{
                        "Definition":20001,
                        "StartTimeOffset":2,
                        "StartTimeOffset":5
                    },
                    "Output":{
                        "Url":"http://1256768367.vod2.myqcloud.com/xxx/xxx/v.f20001
                        "Definition":20001,
                        "Container":"webp",
                        "Height":480,
                        "Width":640,
                        "Bitrate":324271,
                        "Size":121601,
                        "Md5":"084d403c73930ca2f835679af1f37bd3",
                        "StartTimeOffset":3,
                        "EndTimeOffset":5
                    }
                }
            }
        ],
        "TasksPriority":0,
        "TasksNotifyMode":""
    }
}
```


In the callback result, `ProcedureStateChangeEvent.MediaProcessResultSet` contains the transcoding result in `Type` of `AnimatedGraphics`, and `Definition` is 20001.

Transcoding to Adaptive Bitrate Streaming

Last updated：2024-02-20 11:59:37

Adaptive bitrate streaming refers to the process of transcoding a video into adaptive bitrate streams. The results include audio/video files with different bitrates and a descriptive manifest file. A player can dynamically select the most appropriate bitrate for playback based on the current bandwidth. Currently, the most widely used adaptive bitrate streaming format is HLS master playlist.

VOD can transcode videos to adaptive bitrate streams in HLS and MPEG-DASH formats. This feature has the following benefits:

The player can dynamically select the most appropriate bitrate for playback based on the current bandwidth, delivering a smooth viewing experience.

Mainstream players natively support HLS adaptive bitrate with no customization required.

VOD provides a Player SDK, which you can integrate into your project quickly and conveniently to enable the playback of adaptive bitrate streams.

Adaptive Bitrate Streaming Template

You can specify the "video transcoding parameter", "audio transcoding parameter", and other parameters for each adaptive bitrate substream. VOD uses an adaptive bitrate streaming template to represent the parameters.

Parameter	Description
Protocol	Adaptive bitrate streaming protocol. Currently, HLS and MPEG-DASH are supported.
Encryption	Currently, only HLS supports simple AES encryption. MPEG-DASH does not support encryption.
Substream specification	Controls how many substreams are output and the video and audio transcoding parameters of each substream: Video transcoding parameters: resolution, bitrate, frame rate, codec, etc. Audio transcoding parameters: sample rate, sound channel, codec, etc.
Whether to filter "low resolution to high resolution"	Generally, a source video with a low resolution cannot be converted to high resolution to improve the video and sound quality. Enabling filtering "low resolution to high resolution" can help avoid unnecessary transcoding

VOD provides preset adaptive bitrate streaming templates that include common parameter combinations. You can also customize your own template.

Initiating a Task

There are three ways to initiate a transcoding task, namely, through a server API, via the console, and by specifying a task for video upon upload. For more information, please see “Task Initiation” in Video Processing Task System. Below are instructions for initiating adaptive bitrate streaming tasks in these ways:

Call the server API ProcessMedia to initiate a task: set `MediaProcessTask.AdaptiveDynamicStreamingTaskSet` to the ID of the adaptive bitrate streaming template in the API request.

Initiate a task via the console: call a server API to create an adaptive bitrate task flow (by specifying `MediaProcessTask.AdaptiveDynamicStreamingTaskSet`), and use it to process videos in the console.

Specify a task upon upload from server: call a server API to create an adaptive bitrate streaming task flow, (by specifying `MediaProcessTask.AdaptiveDynamicStreamingTaskSet`), and in the ApplyUpload request, set `procedure` to the created task flow.

Specify a task upon upload from client: call a server API to create an adaptive bitrate streaming task flow (by specifying `MediaProcessTask.AdaptiveDynamicStreamingTaskSet`), and in the request to upload video from client, set `procedure` to the created task flow.

Specify a task upon upload from the console: call a server API to create an adaptive bitrate task flow (by specifying `MediaProcessTask.AdaptiveDynamicStreamingTaskSet`), and when uploading video via the console, choose **Automatic Processing After Upload** and select the created task flow.

Getting Result

After initiating an adaptive bitrate streaming task, you can wait for result notification asynchronously or perform task query synchronously to get the task execution result. Below is an example of getting the result notification in normal callback mode after the adaptive bitrate streaming task is initiated (the fields with null value are omitted):

```
{
    "EventType":"ProcedureStateChanged",
    "ProcedureStateChangeEvent":{
        "TaskId":"1256768367-Procedure-2e1af2456351812be963e309cc133403t0",
        "Status":"FINISH",
        "FileId":"5285890784246869930",
        "FileName":"Animal World",
        "FileUrl":"http://1256768367.vod2.myqcloud.com/xxx/xxx/AtUCmy6gmIYA.mp4",
        "MetaData":{
            "AudioDuration":60,
            "AudioStreamSet":[
                {
                    "Bitrate":383854,
                    "Codec":"aac",
```

```
                "SamplingRate":48000
            }
        ],
        "Bitrate":1021028,
        "Container":"mov,mp4,m4a,3gp,3g2,mj2",
        "Duration":60,
        "Height":480,
        "Rotate":0,
        "Size":7700180,
        "VideoDuration":60,
        "VideoStreamSet":[
            {
                "Bitrate":637174,
                "Codec":"h264",
                "Fps":23,
                "Height":480,
                "Width":640
            }
        ],
        "Width":640
    },
    "MediaProcessResultSet":[
        {
            "Type":"AdaptiveDynamicStreaming",
            "AdaptiveDynamicStreamingTask":{
                "Status":"SUCCESS",
                "ErrCode":0,
                "Message":"",
                "Input":{
                    "Definition":10
                },
                "Output":{
                    "Definition":10,
                    "Package":"hls",
                    "DrmType":"",
                    "Url":"http://1256768367.vod2.myqcloud.com/xxx/xxx/adp.10.m
                }
            }
        },
        {
            "Type":"AdaptiveDynamicStreaming",
            "AdaptiveDynamicStreamingTask":{
                "Status":"SUCCESS",
                "ErrCode":0,
                "Message":"",
                "Input":{
                    "Definition":20
```

```
                    },
                    "Output":{
                        "Definition":20,
                        "Package":"dash",
                        "DrmType":"",
                        "Url":"http://1256768367.vod2.myqcloud.com/xxx/xxx/adp.20.m
                    }
                }
            }
        ],
        "TasksPriority":0,
        "TasksNotifyMode":""
    }
}
```


In the callback, `ProcedureStateChangeEvent.MediaProcessResultSet` contains two adaptive bitrate streams, whose `Type` is `AdaptiveDynamicStreaming` and `Definition` 10 and 20 respectively.

Image Processing Overview

Last updated：2022-12-01 16:30:24

The image processing feature allows you to analyze and process images in VOD.

Type	Description
Real-time image processing	Supported operations: Scaling: Scale an image to specific dimensions (width, height, long side, or short side). Cropping: Crop an image to a circle with a specified radius or to a rectangle with specified dimensions. The geometric center of the original image is kept.

After starting an image processing task, you will get the result immediately (synchronously). For directions on how to start an image processing task and get the result, see Real-Time Image processing.

Real-Time Image Processing

Last updated：2022-12-01 16:30:24

Image scaling and cropping have many use cases. For example, images may be scaled to generate thumbnails for media files, and cropping is needed to make square or circular user profile photos. When images are stored in the cloud, traditional editing methods such as locally run software and online editing tools have many disadvantages, for example:

The process of downloading, editing, and uploading is complicated.

Manual editing is not efficient and prone to errors.

These pain points can be addressed by VOD's real-time image processing feature.

Aspect	Traditional Image Editing	VOD Image Processing
Process	You need to download the images, edit them, and then upload them. The process is complicated and time-consuming.	All operations are performed in the cloud, with no need for download or upload.
Operation	You need to have some knowledge of image editing to be able to use image editing software.	You can edit images in real time simply by specifying URL parameters.
Speed	It can be slow to access and download images stored in the cloud via URLs, hurting the user experience.	CDNs are used for acceleration, allowing you to get the result immediately.

The real-time image processing feature of VOD supports scaling and cropping operations.

Type	Operation
Scaling	Width: Specified; height: Auto-scaled
	Height: Specified; width: Auto-scaled
	Long side: Specified; short side: Auto-scaled
	Short side: Specified; long side: Auto-scaled
	Width: Specified; height: Specified
Cropping	Cropping to circle, with the radius specified
	Cropping to rectangle, with height and width specified

For detailed directions on how to process images with VOD, see Real-Time Image Processing.

Preset Templates

Last updated：2023-03-07 11:47:21

VOD offers a series of preset templates for different image processing scenarios. Instead of setting the parameters one by one, you can use the ready-made templates to initiate image processing tasks.

Real-Time Image Processing

Each template represents a set of image processing operations.

Image processing templates

Template ID	Description	Parameters
10	Crop	Type: Crop to rectangle Width: 360 px Height: 200 px
20	Crop	Type: Crop to rectangle Width: 200 px Height: 400 px
30	Scale	Type: Specify the short side and auto-scale the long side Short side: 320 px
40	Scale	Type: Specify the height and width Width: 200 px Height: 200 px
50	Scale and crop	Scale: Type: Specify the short side and auto-scale the long side Short side: 320 px
		Crop: Type: Crop to rectangle Width: 200 px Height: 200 px

Image Moderation

Image moderation templates

Template ID	Porn	Terror
10	Yes	Yes

Video AI
Audio/Video Moderation

Last updated：2023-10-13 17:30:08

Audio/Video content moderation is an offline task that intelligently moderates audio/video content with the aid of AI. The task execution results include confidence score, moderation suggestion, and suspected audio/video segments. According to the suggestion, you can decide whether to allow an audio/video to be published, effectively avoiding potential legal risks and damage to your brand's reputation.
VOD can perform content moderation on images, text in images, speech, and sounds. Supported moderation labels include porn, terrorism, and moaning.

Content Type	Moderation Label
Images	Pornographic (`Porn`)
	Terrorist (`Terror`)
Sounds	Moaning (`Moan`)
Speech (ASR)	Pornographic (`Porn`)
	Terrorist (`Terror`)
Text in images (OCR)	Pornographic (`Porn`)
	Terrorist (`Terror`)

The table below lists some of the fields in moderation results:

Field	Type	Description
Confidence	Float	The moderation score (0-100). The higher the score, the more likely the content is non-compliant.
Suggestion	String	The suggestion. Valid values: `pass` , `review` , `block` . `pass` : The probability of the content being non-compliant is low. We recommend you allow the content to pass. `review` : The probability of the content being non-compliant is high. Manual verification is recommended. `block` : There's a high chance that the content is non-compliant. We recommend you block the content.
Form	String	The moderated content type. Valid values:

		`Image`
		`Voice`
		`OCR`
		`ASR`

Label	String	The moderation label. Valid values:
		`Porn`
		`Terror`
		`Moan`

Audio/Video Moderation Template

An audio/video moderation template represents a set of moderation parameters. You can use a template to specify which of the following moderation labels to use:
Pornographic (`Porn`)
Terrorist (`Terror`)
Moaning (`moan`)
VOD provides preset audio/video moderation templates for common parameter combinations. You can also use a server API to create and manage custom templates.

Initiating a Moderation Task

You can initiate an audio/video moderation task by calling a server API, via the console, or by specifying the task when uploading videos. For details, see Task Initiation.
You can initiate an audio/video moderation task in these ways:
Call the server-side API ReviewAudioVideo.
Initiate a task in the console. For detailed directions, see Audio/Video Moderation.
Specify a task when uploading audio/video from server: Create a task flow with moderation enabled in the console. Then, when calling ApplyUpload to upload audio/video, set `procedure` to the name of the task flow.
Specify a task when uploading audio/video from client: Create a task flow with moderation enabled in the console. Then, when generating the signature for upload from client, set `procedure` to the name of the task flow.
Specify a task when uploading audio/video from the console: In the console, create a task flow with moderation enabled. Then, when uploading audio/video files, choose Auto-processing after upload and select the task flow created.

Getting the Result

After initiating a moderation task, you can either wait for the ReviewAudioVideoComplete notification asynchronously or perform a task query synchronously to get the task execution result. Below is an example of the notification in normal callback mode (the fields with null value are omitted):


```
{
    "EventType": "ReviewAudioVideoComplete",
    "ReviewAudioVideoCompleteEvent": {
        "TaskId": "125xxxx-ReviewAudioVideo-07edbc78ba20563cdf2362cffbf4aa0ct",
        "Status": "FINISH",
        "ErrCodeExt": "",
        "Message": "SUCCESS",
        "Input":{
            "FileId": "387702130626135215"
        },
        "Output": {
            "Suggestion": "block",
            "Label": "Porn",
            "Form": "Image",
            "SegmentSet":[
                {
                    "StartTimeOffset": 0,
                    "EndTimeOffset": 1,
                    "Confidence": 99,
                    "Suggestion": "block",
                    "Label": "Porn",
                    "SubLabel": "SexyBehavior",
                    "Form": "Image",
                    "AreaCoordSet": [],
                    "Text": "",
                    "KeywordSet": [],
                    "Url": "https://251000800.vod2.myqcloud.com/1a168d62vodcq251000
                    "PicUrlExpireTime": "2023-01-16T03:06:16.039Z"
                },
                {
                    "StartTimeOffset": 1,
                    "EndTimeOffset": 2,
                    "Confidence": 99,
                    "Suggestion": "block",
                    "Label": "Porn",
                    "SubLabel": "SexyBehavior",
                    "Form": "Image",
                    "AreaCoordSet": [],
                    "Text": "",
                    "KeywordSet": [],
                    "Url": "https://251000800.vod2.myqcloud.com/1a168d62vodcq251000
                    "PicUrlExpireTime": "2023-01-16T03:06:17.039Z"
```

```
            },
            {
                "StartTimeOffset": 2,
                "EndTimeOffset": 3,
                "Confidence": 99,
                "Suggestion": "block",
                "Label": "Porn",
                "SubLabel": "SexyBehavior",
                "Form": "Image",
                "AreaCoordSet": [],
                "Text": "",
                "KeywordSet": [],
                "Url": "https://251000800.vod2.myqcloud.com/1a168d62vodcq251000
                "PicUrlExpireTime": "2023-01-16T03:06:18.039Z"
            },
            {
                "StartTimeOffset": 3,
                "EndTimeOffset": 4,
                "Confidence": 99,
                "Suggestion": "block",
                "Label": "Porn",
                "SubLabel": "SexyBehavior",
                "Form": "Image",
                "AreaCoordSet": [],
                "Text": "",
                "KeywordSet": [],
                "Url": "https://251000800.vod2.myqcloud.com/1a168d62vodcq251000
                "PicUrlExpireTime": "2023-01-16T03:06:19.039Z"
            },
            {
                "StartTimeOffset": 4,
                "EndTimeOffset": 5,
                "Confidence": 99,
                "Suggestion": "block",
                "Label": "Porn",
                "SubLabel": "SexyBehavior",
                "Form": "Image",
                "AreaCoordSet": [],
                "Text": "",
                "KeywordSet": [],
                "Url": "https://251000800.vod2.myqcloud.com/1a168d62vodcq251000
                "PicUrlExpireTime": "2023-01-16T03:06:20.039Z"
            },
            {
                "StartTimeOffset": 5,
                "EndTimeOffset": 6,
                "Confidence": 99,
```

```
                "Suggestion": "block",
                "Label": "Porn",
                "SubLabel": "SexyBehavior",
                "Form": "Image",
                "AreaCoordSet": [],
                "Text": "",
                "KeywordSet": [],
                "Url": "https://251000800.vod2.myqcloud.com/1a168d62vodcq251000
                "PicUrlExpireTime": "2023-01-16T03:06:21.039Z"
            },
            {
                "StartTimeOffset": 6,
                "EndTimeOffset": 7,
                "Confidence": 99,
                "Suggestion": "block",
                "Label": "Porn",
                "SubLabel": "SexyBehavior",
                "Form": "Image",
                "AreaCoordSet": [],
                "Text": "",
                "KeywordSet": [],
                "Url": "https://251000800.vod2.myqcloud.com/1a168d62vodcq251000
                "PicUrlExpireTime": "2023-01-16T03:06:22.039Z"
            },
            {
                "StartTimeOffset": 7,
                "EndTimeOffset": 8,
                "Confidence": 99,
                "Suggestion": "block",
                "Label": "Porn",
                "SubLabel": "SexyBehavior",
                "Form": "Image",
                "AreaCoordSet": [],
                "Text": "",
                "KeywordSet": [],
                "Url": "https://251000800.vod2.myqcloud.com/1a168d62vodcq251000
                "PicUrlExpireTime": "2023-01-16T03:06:23.039Z"
            },
            {
                "StartTimeOffset": 8,
                "EndTimeOffset": 9,
                "Confidence": 99,
                "Suggestion": "block",
                "Label": "Porn",
                "SubLabel": "SexyBehavior",
                "Form": "Image",
                "AreaCoordSet": [],
```

```
                    "Text": "",
                    "KeywordSet": [],
                    "Url": "https://251000800.vod2.myqcloud.com/1a168d62vodcq251000
                    "PicUrlExpireTime": "2023-01-16T03:06:24.039Z"
                },
                {
                    "StartTimeOffset": 9,
                    "EndTimeOffset": 10,
                    "Confidence": 99,
                    "Suggestion": "block",
                    "Label": "Porn",
                    "SubLabel": "SexyBehavior",
                    "Form": "Image",
                    "AreaCoordSet": [],
                    "Text": "",
                    "KeywordSet": [],
                    "Url": "https://251000800.vod2.myqcloud.com/1a168d62vodcq251000
                    "PicUrlExpireTime": "2023-01-16T03:06:25.039Z"
                }
            ],
            "SegmentSetFileUrl": "http://251000800.vod2.myqcloud.com/a8800b40vodtra
            "SegmentSetFileUrlExpireTime": "2022-10-12T07:01:07.695Z"
        },
        "SessionContext": "",
        "SessionId": ""
    }
}
```


In the above callback, `ReviewAudioVideoCompleteEvent.Output` is the audio/video moderation result; `Output.Suggestion=block` indicates that VOD suggests you block the content; `Output.Label=Porn` and `Output.Form=Image` indicate that the mostly likely violation is pornographic content in images.

An audio/video clip may include multiple suspected segments. `Output.SegmentSet` lists only the first 10. You can view the information of all suspected segments by visiting `Output.SegmentSetFileUrl` within the validity period.

`StartTimeOffset` and `EndTimeOffset` are the start and end time of a suspected segment, and `SubLabel` indicates the type of violation.

If text in images or speech is moderated:

`Text` is the full text recognized.

`KeywordSet` is the list of non-compliant words hit.

If images (people and objects) or text in images are moderated:

`AreaCoordSet` is the coordinates of the suspected object.

`Url` is the URL of the suspected image.

`PicUrlExpireTime` is the expiration time of `Url` .

Video Content Analysis

Last updated：2023-03-22 14:46:48

Audio/Video content analysis is an offline task that intelligently analyzes audio/video content with the aid of AI. It intelligently gives suggestions for video categorization, labeling, and thumbnail generation to help video platforms manage videos more accurately and efficiently.

Audio/Video content analysis can do the following:

Feature	Description
Intelligent categorization	Gives suggestions on classifying videos into over 10 categories, including: news, entertainment, gaming, technology, food, sports, travel, animation, dance, music, movies & TV, and automobiles.
Intelligent labeling	Gives suggestions on labeling videos. Currently, more than 3,000 labels are supported, including: gaming, transportation, musician, racing, pet, drums, bicycle, WOW, computer, school, and jacket.
Intelligent thumbnail generation	Captures one or more screenshots of a video as the recommended cover.
Intelligent labeling by frame	Gives suggestions on labeling each frame of a video. Currently, over 1,000 labels are supported, including: contemporary dance, water sports, steak, baby, kitten, annual plant, destroyer, comics, lawn, wedding dress, function room, and passport.

Audio/Video Analysis Template

You can use audio/video analysis parameters (templates) to specify the operations an audio/video analysis task performs:

Whether to enable intelligent categorization.

Whether to enable intelligent labeling.

Whether to enable intelligent thumbnail generation.

Whether to enable intelligent labeling by frame.

VOD provides preset audio/video analysis templates for common parameter combinations. You can also use a server API to create and manage custom templates.

Initiating a Task

You can initiate an audio/video analysis task by calling a server API, via the console, or by specifying the task when uploading videos. For details, see Task Initiation.
Below are the details:
Initiate a task by calling a server API: Call ProcessMedia, setting `Definition` in the request parameter `AiAnalysisTask` to the ID of the audio/video content analysis template.
Initiate a task via the console: Call the server API CreateProcedureTemplate to create an audio/video analysis task flow (`MediaProcessTask.AiAnalysisTask`), and use it to process videos in the console.
Specify a task when uploading videos from the server: Call the server API CreateProcedureTemplate to create an audio/video analysis task flow (`MediaProcessTask.AiAnalysisTask`). When calling ApplyUpload, set the parameter `procedure` to the task flow.
Specify a task when uploading videos from a client: Call the server API CreateProcedureTemplate to create an audio/video analysis task flow (`MediaProcessTask.AiAnalysisTask`). When generating a signature for upload, set the parameter `procedure` to the task flow.
Specify a task when uploading videos via the console: Call the server API CreateProcedureTemplate to create an audio/video analysis task flow (`MediaProcessTask.AiAnalysisTask`). When uploading videos via the console, select Auto-processing after upload and choose the task flow.

Getting the Result

After initiating an audio/video content analysis task, you can wait for the result notification asynchronously or perform a task query synchronously to get the task execution result. Below is an example of getting the result notification in normal callback mode after a content analysis task is initiated (the fields with null value are omitted):

```
{
    "EventType":"ProcedureStateChanged",
    "ProcedureStateChangeEvent":{
        "TaskId":"1256768367-Procedure-2e1af2456351812be963e309cc133403t0",
        "Status":"FINISH",
        "FileId":"5285890784246869930",
        "FileName":"Animal World",
        "FileUrl":"http://1256768367.vod2.myqcloud.com/xxx/xxx/AtUCmy6gmIYA.mp4",
        "MetaData":{
            "AudioDuration":60,
            "AudioStreamSet":[
                {
                    "Bitrate":383854,
                    "Codec":"aac",
                    "SamplingRate":48000
```

```
                }
            ],
            "Bitrate":1021028,
            "Container":"mov,mp4,m4a,3gp,3g2,mj2",
            "Duration":60,
            "Height":480,
            "Rotate":0,
            "Size":7700180,
            "VideoDuration":60,
            "VideoStreamSet":[
                {
                    "Bitrate":637174,
                    "Codec":"h264",
                    "Fps":23,
                    "Height":480,
                    "Width":640
                }
            ],
            "Width":640
        },
        "AiAnalysisResultSet":[
            {
                "Type":"Classification",
                "ClassificationTask":{
                    "Status":"SUCCESS",
                    "ErrCode":0,
                    "Message":"",
                    "Input":{
                        "Definition":10
                    },
                    "Output":{
                        "ClassificationSet":[
                            {
                                "Classification":"Animals",
                                "Confidence":80
                            },
                            {
                                "Classification":"Travel",
                                "Confidence":34
                            }
                        ]
                    }
                }
            },
            {
                "Type":"Cover",
                "CoverTask":{
```

```
                "Status":"SUCCESS",
                "ErrCode":0,
                "Message":"",
                "Input":{
                    "Definition":10
                },
                "Output":{
                    "CoverSet":[
                        {
                            "CoverUrl":"http://1256768367.vod2.myqcloud.com/xxx
                            "Confidence":79
                        },
                        {
                            "CoverUrl":"http://1256768367.vod2.myqcloud.com/xxx
                            "Confidence":70
                        },
                        {
                            "CoverUrl":"http://1256768367.vod2.myqcloud.com/xxx
                            "Confidence":66
                        }
                    ]
                }
            }
        },
        {
            "Type":"Tag",
            "TagTask":{
                "Status":"SUCCESS",
                "ErrCode":0,
                "Message":"",
                "Input":{
                    "Definition":10
                },
                "Output":{
                    "TagSet":[
                        {
                            "Tag":"Horse",
                            "Confidence":34
                        },
                        {
                            "Tag":"Bird",
                            "Confidence":27
                        },
                        {
                            "Tag":"Plant",
                            "Confidence":13
                        },
```

```
                                {
                                    "Tag":"Beach",
                                    "Confidence":11
                                }
                            ]
                        }
                    }
                }
            ],
            "TasksPriority":0,
            "TasksNotifyMode":""
        }
    }
```


In the callback result, `ProcedureStateChangeEvent.AiAnalysisResultSet` contains three types of analysis results, which are video categorization (`Classification`), thumbnail generation (`Cover`), and labeling (`Tag`).

For video categorization (`Classification`), the category with the highest confidence score (`Output.ClassificationSet`) is `Travel` .

For thumbnail generation (`Cover`), three thumbnails (`CoverSet`) are recommended. `CoverUrl` indicates the download URL of each thumbnail.

For labeling (`Tag`), four labels (`Output.TagSet`) are recommended, which are listed by confidence score in descending order.

Video Content Recognition

Last updated：2024-11-04 10:11:11

Since August 1, 2022, the audio/video content recognition feature of VOD is paid feature. For more information, see Video Recognition to Become Paid Feature.

Audio/Video content recognition is an offline task that intelligently recognizes audio/video content with the aid of AI. It recognizes faces, text, opening and closing segments, and speech in the video, helping you accurately and efficiently manage your audio/videos. Specifically, it includes the following features:

Feature	Description	Use Cases
Face recognition	Recognizes faces in video images.	Marks where celebrities appear in video images Checks for particular people in video images
Full speech recognition	Recognizes all words that occur in speech	Generates subtitles for speech content Performs data analysis on video speech content
Full text recognition	Recognizes all text that occurs in video images	Performs data analysis on text in video images
Speech keyword recognition	Recognizes keywords in speech	Checks for sensitive words in speech Retrieves specific keywords in speech
Text keyword recognition	Recognizes keywords in video images	Checks for sensitive words in video images Retrieves specific keywords in video images
Opening and closing segment recognition	Recognizes opening and closing segments in videos	Marks the positions of the opening segment, and closing segment, and feature presentation in the progress bar Removes the opening and closing segments of multiple videos at a time
Speech translation recognition	Recognizes all words that occur in speech and translates them into the specified language.	Generates translated subtitles for short dramas. Generates multilingual subtitles for recorded files of cross-border audio and video conferences.

Some content recognition features depend on a material library. There are two types of libraries: public library and custom library.

Public library: VOD's preset material library.

Custom library: Your own library

Recognition Type	Public Library	Custom Library
Face recognition	Supported. The library includes celebrities in the sports and entertainment industries, as well as other people.	Supported. You can use a server API to manage the custom face library.
Speech recognition	Not supported yet	Supported. You can use a server API to manage the custom keyword library.
Text recognition	Not supported yet	Supported. You can use a server API to manage the custom keyword library.

Audio/Video Content Recognition Template

Audio/Video content recognition integrates a number of recognition features. You can use parameters to control the following:

Which content recognition features to enable

Whether to use the public library or custom library for face recognition

The confidence score threshold to return face recognition results

The labels of the faces to return

VOD provides preset video content recognition templates for common parameter combinations. You can also use a server API to create and manage custom templates.

Initiating a Task

You can initiate an audio/video recognition task by calling a server API, via the console, or by specifying the task when uploading videos. For details, see Task Initiation.

Below are the details:

Initiate a task by calling a server API: Call ProcessMedia, setting `Definition` in the request parameter `AiRecognitionTask` to the ID of the audio/video content recognition template.

Initiate a task via the console: Call the server API CreateProcedureTemplate to create an audio/video recognition task flow (`MediaProcessTask.AiRecognitionTask`), and use it to process videos in the console.

Specify a task when uploading videos from the server: Call the server API CreateProcedureTemplate to create an audio/video recognition task flow (`MediaProcessTask.AiRecognitionTask`). When calling ApplyUpload, set the parameter `procedure` to the task flow.

Specify a task when uploading videos from a client: Call the server API CreateProcedureTemplate to create an audio/video recognition task flow (`MediaProcessTask.AiRecognitionTask`). When generating a signature

for upload, set the parameter `procedure` to the task flow.
Specify a task when uploading videos via the console: Call the server API CreateProcedureTemplate to create an audio/video recognition task flow (`MediaProcessTask.AiRecognitionTask`). When uploading videos via the console, select Auto-processing after upload and choose the task flow.

Getting the Result

After initiating an audio/video content recognition task, you can wait for the result notification asynchronously or perform a task query synchronously to get the task execution result. Below is an example of getting the result notification in normal callback mode after a content recognition task is initiated (the fields with null value are omitted):


```
{
    "EventType":"ProcedureStateChanged",
    "ProcedureStateChangeEvent":{
        "TaskId":"1400155958-Procedure-2e1af2456351812be963e309cc133403t0",
        "Status":"FINISH",
        "FileId":"5285890784363430543",
        "FileName":"Collection",
        "FileUrl":"http://1400155958.vod2.myqcloud.com/xxx/xxx/aHjWUx5Xo1EA.mp4",
        "MetaData":{
            "AudioDuration":243,
            "AudioStreamSet":[
                {
                    "Bitrate":125599,
                    "Codec":"aac",
                    "SamplingRate":48000
                }
            ],
            "Bitrate":1459299,
            "Container":"mov,mp4,m4a,3gp,3g2,mj2",
            "Duration":243,
            "Height":1080,
            "Rotate":0,
            "Size":44583593,
            "VideoDuration":243,
            "VideoStreamSet":[
                {
                    "Bitrate":1333700,
                    "Codec":"h264",
                    "Fps":29,
                    "Height":1080,
                    "Width":1920
                }
            ],
```

```
            "Width":1920
        },
        "AiRecognitionResultSet":[
            {
                "Type":"FaceRecognition",
                "FaceRecognitionTask":{
                    "Status":"SUCCESS",
                    "ErrCode":0,
                    "Message":"",
                    "Input":{
                        "Definition":10
                    },
                    "Output":{
                        "ResultSet":[
                            {
                                "Id":183213,
                                "Type":"Default",
                                "Name":"John Smith",
                                "SegmentSet":[
                                    {
                                        "StartTimeOffset":10,
                                        "EndTimeOffset":12,
                                        "Confidence":97,
                                        "AreaCoordSet":[
                                            830,
                                            783,
                                            1030,
                                            599
                                        ]
                                    },
                                    {
                                        "StartTimeOffset":12,
                                        "EndTimeOffset":14,
                                        "Confidence":97,
                                        "AreaCoordSet":[
                                            844,
                                            791,
                                            1040,
                                            614
                                        ]
                                    }
                                ]
                            },
                            {
                                "Id":236099,
                                "Type":"Default",
                                "Name":"Jane Smith",
```

```
                                "SegmentSet":[
                                    {
                                        "StartTimeOffset":120,
                                        "EndTimeOffset":122,
                                        "Confidence":96,
                                        "AreaCoordSet":[
                                            579,
                                            903,
                                            812,
                                            730
                                        ]
                                    }
                                ]
                            }
                        ]
                    }
                }
            ],
            "TasksPriority":0,
            "TasksNotifyMode":""
        }
    }
```


In the callback result, `ProcedureStateChangeEvent.AiRecognitionResultSet` contains the result of face recognition (`Type` is `FaceRecognition`).

According to the content of `Output.ResultSet` , two people are recognized: `John Smith` and `Jane Smith` . `SegmentSet` indicates when (from `StartTimeOffset` to `EndTimeOffset`) and where (coordinates specified by `AreaCoordSet`) the two people appear in the video.

Image Moderation

Last updated：2023-04-17 15:01:47

VOD's image moderation feature detects non-compliant information in images with the help of AI. The moderation results generated include a score and a suggestion. You can decide whether to publish an image based on the results. This helps you avoid potential legal risks and damage to your brand's reputation.
VOD can moderate images as well as the text in images. The supported moderation labels include porn, terrorism, politically sensitive, illegal, abuse, and ads.

Content Type	Moderation Label
Images	Pornographic (`Porn`)
	Terrorist (`Terror`)
	Politically sensitive (`Polity`)
	Ads (`Ad`)
	Illegal (`Illegal`)
Text in images (OCR)	Pornographic (`Porn`)
	Terrorist (`Terror`)
	Politically sensitive (`Polity`)
	Ads (`Ad`)
	Illegal (`Illegal`)
	Abuse (`Abuse`)

The moderation results include the following fields:

Field	Type	Description
Confidence	Float	The moderation score (0-100). The higher the score, the more likely the content is non-compliant.
Suggestion	String	The suggestion. Valid values: `pass` , `review` , `block` . `pass` : The probability of the content being non-compliant is low. We recommend you allow the content to pass. `review` : The probability of the content being non-compliant is high. Manual verification is recommended.

		`block` : There's a high chance that the content is non-compliant. We recommend you block the content.

Initiating a Moderation Task

You can start an image moderation task either via the console or by calling a server API.

Obtaining the Result

Moderation results are returned immediately, regardless of how you start a task.
If a task is created in the console, get the result in the console. If a task is created using the ReviewImage API, the result will be returned by the API. For the structure of the data returned, see Review Image - 3. Output Parameters.

Event Notification Overview

Last updated：2022-10-27 12:14:59

An operation such as uploading, deleting, or video processing initiated on a video in VOD can be referred to as an event. The execution of an event takes a certain amount of time. Upon completion of the event, VOD will immediately notify the application service of the execution result, i.e., sending an event notification.
VOD supports the following types of event notifications:

Categorization	Event notification
Upload and deletion	Video upload completion
	Video pull from URL completion
	Video deletion completion
Video processing	Task flow status change
	Video editing completion
	Video composing completion

Event notification modes include "normal callback" and "reliable callback". You can log in to the VOD Console to set the callback mode and select the events for which you want to receive callbacks. For detailed directions, please see Callback Settings.
Normal callback: configure a callback URL in the console. After an event is completed, the system will send an HTTP request to this URL, which contains the notification content.
Reliable callback: after an event is completed, the VOD system will put the notifications into a built-in message queue, and then the application service will consume the notifications in the queue through a server API.

Normal Callback

Normal callback is a mode in which the application service passively receives event notifications. After the callback URL is configured and the normal callback mode is selected, VOD will initiate a callback to the callback URL after an event is completed.
A normal callback initiated by VOD is an HTTP request, where the request body is in JSON format and the content is the `EventContent` structure excluding the `EventHandle` parameter.

Take task status change notification as an example. The `EventType` parameter in the callback is `ProcedureStateChanged`, and the information is represented by the `ProcedureStateChangeEvent` parameter (`ProcedureTask` structure).

Reliable Callback

Reliable callback is a mode in which the application service actively pulls event notifications to VOD. After the reliable callback mode is selected, the VOD system will put event notifications into a queue, and the application service will consume the notifications in the queue through a server API.
After the application service gets a message through the PullEvents API, the ConfirmEvents API needs to be called for confirmation. The message must be confirmed for receipt before it can be removed from the queue in VOD, so the reliability of "reliable callback" is higher than that of "normal callback". **If the requirement for event notification reliability is high, you are recommended to use the "reliable callback" mode.**

Getting Started

Last updated：2024-12-02 18:45:34

This tutorial guides you through how to use VOD event notifications in "normal callback" and "reliable callback" modes.

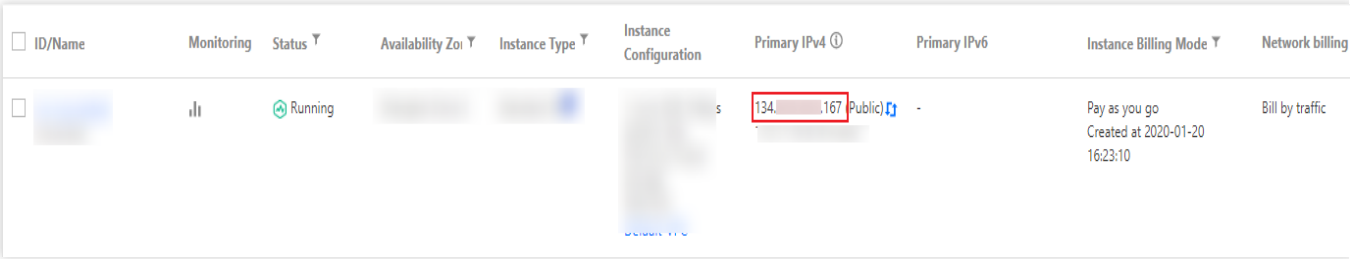
Prerequisites

Sign up for a Tencent Cloud account and verify your identity.
Python 2.7 runtime environment for normal callback.

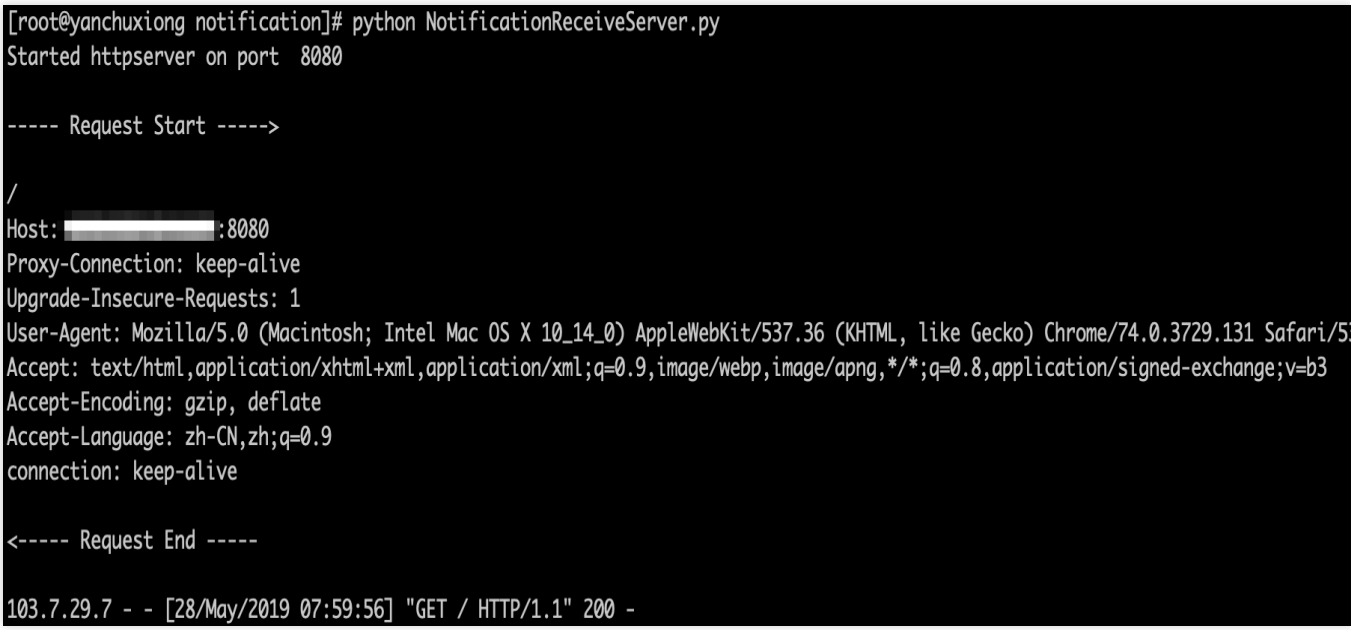
Normal Callback

Deploying callback receiving service

To get event notifications through normal callback, you need to deploy a callback receiving service on a server with a public IP. Below describes how to deploy such a service on a CVM instance as an example:
1. Enter the Instance List page in the CVM Console and click **Create**.
2. Select the **Quick Configuration** menu, select **Ubuntu Server** or **CentOS** for **Image** and **1 Mbps** for **Public Network Bandwidth**, check **Allocate Free Public IP**, and then click **Buy Now**.
3. Enter the Instance List page again, find the CVM instance successfully created, and copy the public IP in **Primary IP Address** (**134.XXX.XXX.167** in this example).



4. Log in to the purchased CVM instance, download the source code package, extract it to your working directory, and run the following command:
python NotificationReceiveServer.py
After the command is executed, the standard output of the CVM instance should print `Started httpserver on port 8080` , indicating that the service process has started and is listening on port 8080.
5. Enter `http://134.XXX.XXX.167:8080` in a browser, and the standard output of the CVM instance should print the following HTTP request information.



Configuring normal callback

1. Log in to the VOD Console and click **Callback Settings** on the left sidebar.
2. Click **Settings**:

Event Notification Method: Select **Normal Callback**.

Callback URL: enter `http://134.XXX.XXX.167:8080` .

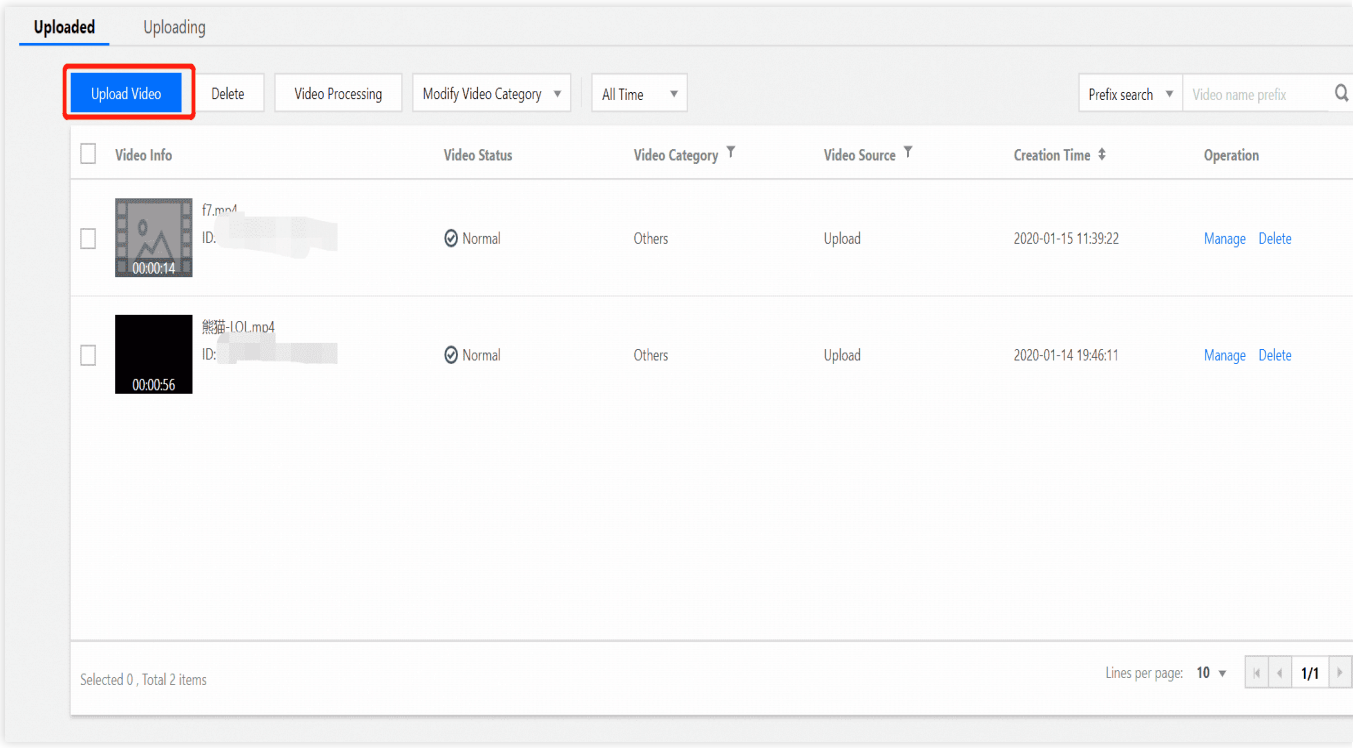
Event Notification: Select **Finished video uploading**.

3. Click **Confirm**.

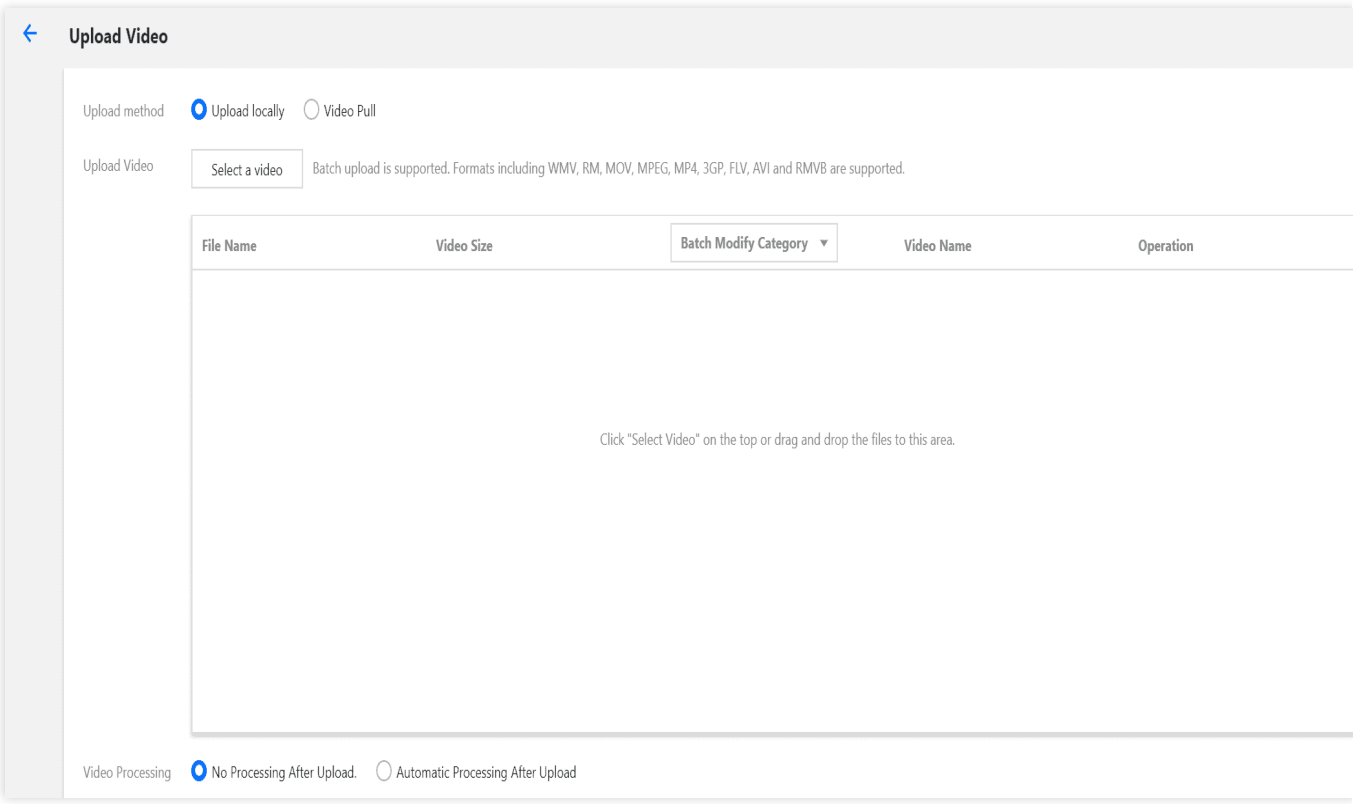
Initiating and receiving normal callback

Please download the demo video to your local file system for getting started.

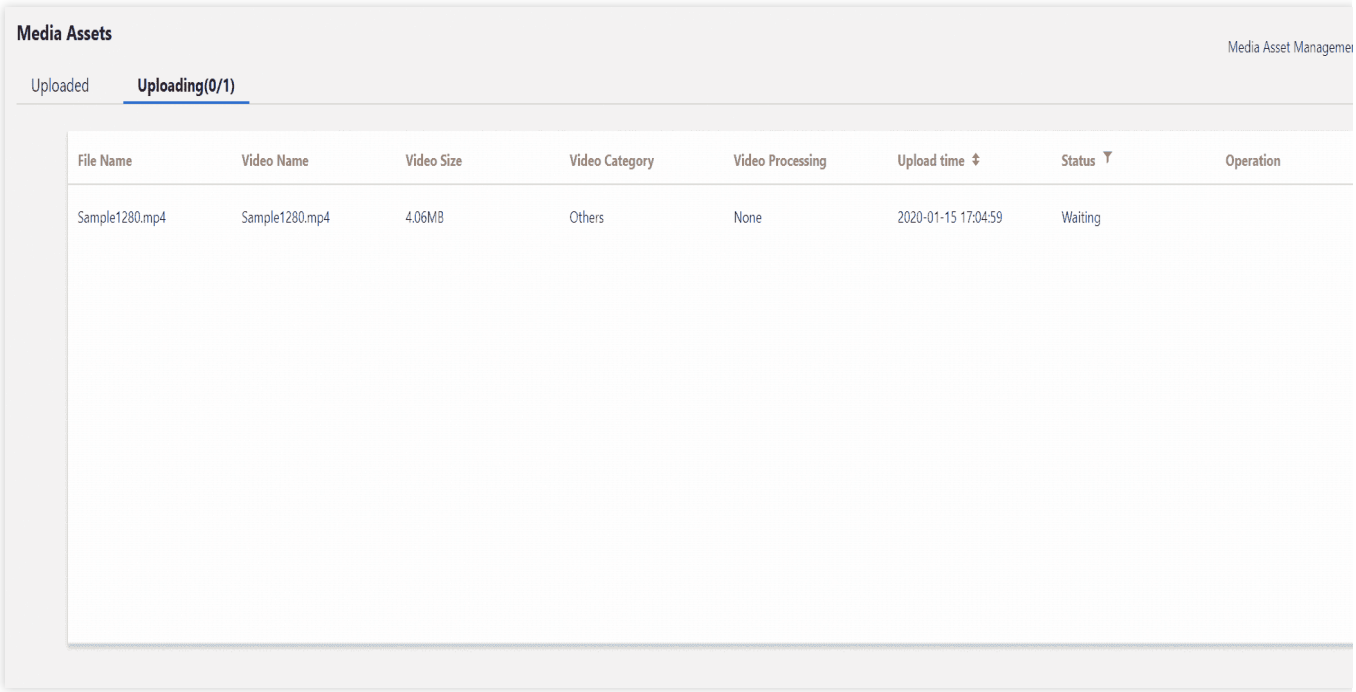
1. Select **Media Assets > Video Management** on the left sidebar.



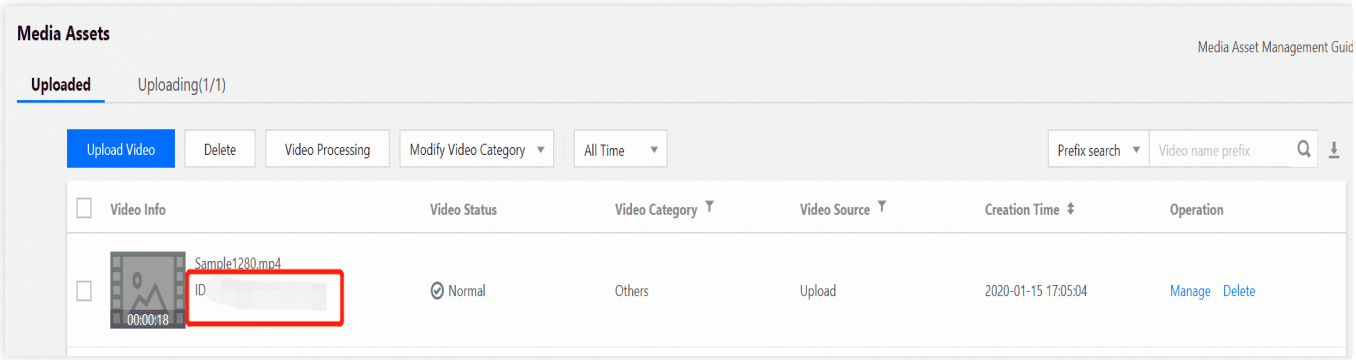
2. In the **Upload Video** dialog box that pops up, select **Local Upload**, click **Select Video**, and upload the **demo video** to the VOD platform.



After performing the upload operation, you will see the video upload progress in the **Uploading** column.



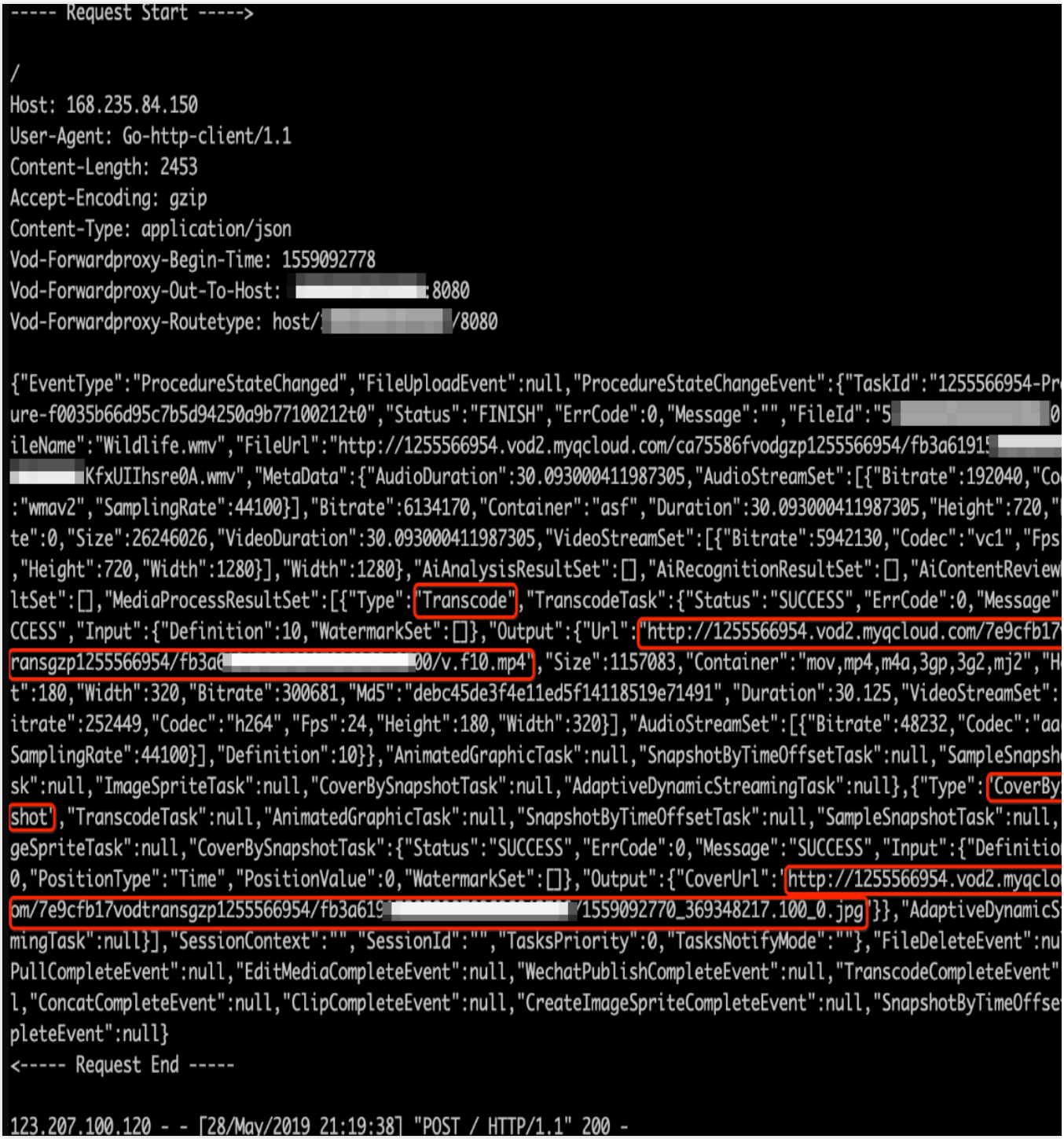
After the upload is completed, you will see the uploaded video and its corresponding ID (i.e., `FileId`) in the video list in the **Uploaded** column.



3. Check the CVM instance. The standard output should print the content of the notification for video upload completion.

4. In the **Uploaded** column in **Media Assets**, select the video just uploaded and click **Video Processing**. Select **Manually select transcoding template** for **Processing Type**, check **MP4-LD-FLU (10)** in **Transcoding Template**, keep **Video Cover** checked, and click **OK**.

5. After waiting for 10 minutes, check the CVM instance, and its standard output should print the content of the notification for task flow status change, including the results of transcoding (where `Type` is `Transcode`) and time point screencapturing for cover generation (where `Type` is `CoverBySnapshot`).



At this point, you have uploaded a video and performed a transcoding task on it. After the upload and transcoding were completed, your callback receiving service received notifications for **video upload completion** and **task flow status change**.

Reliable Callback

1. Log in to the VOD Console and click **Callback Settings** on the left sidebar.

2. Click **Settings**:

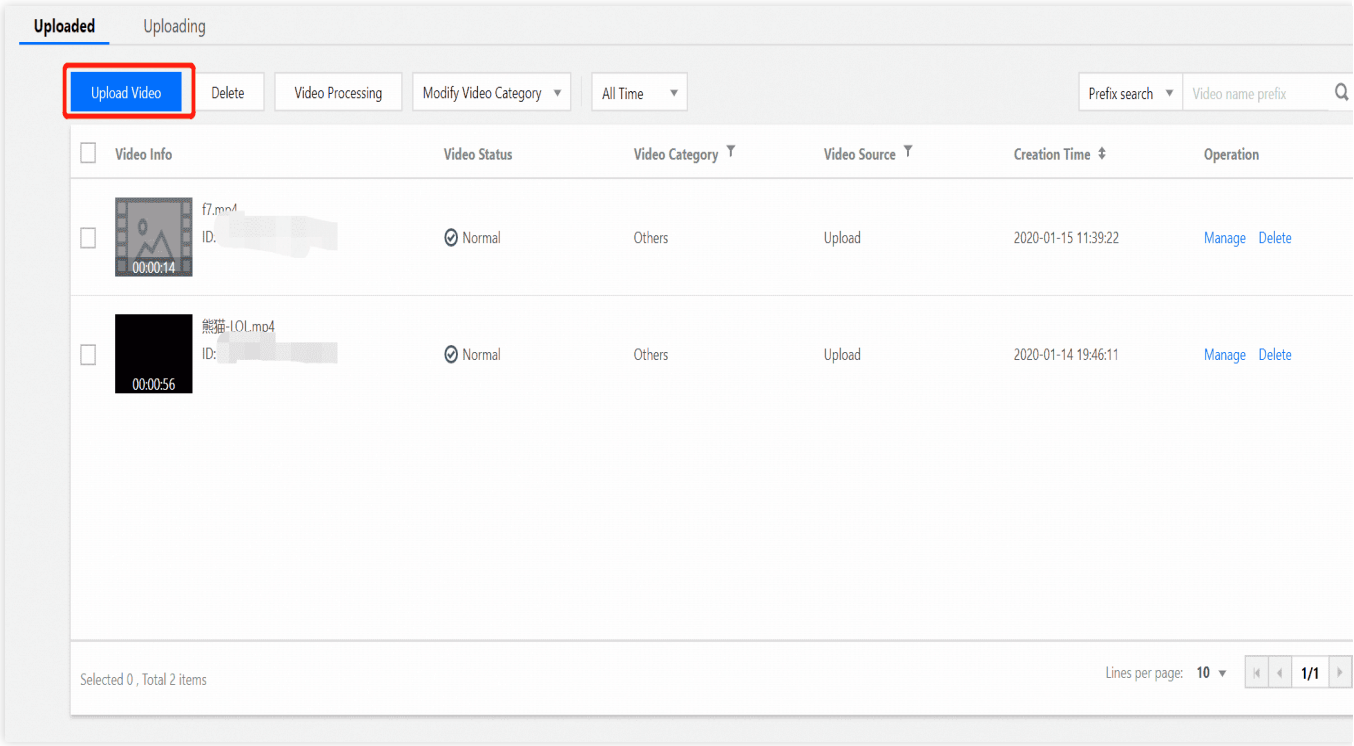
Callback Mode: select **Reliable Callback**.

Callback Event: check **video upload completion callback**.

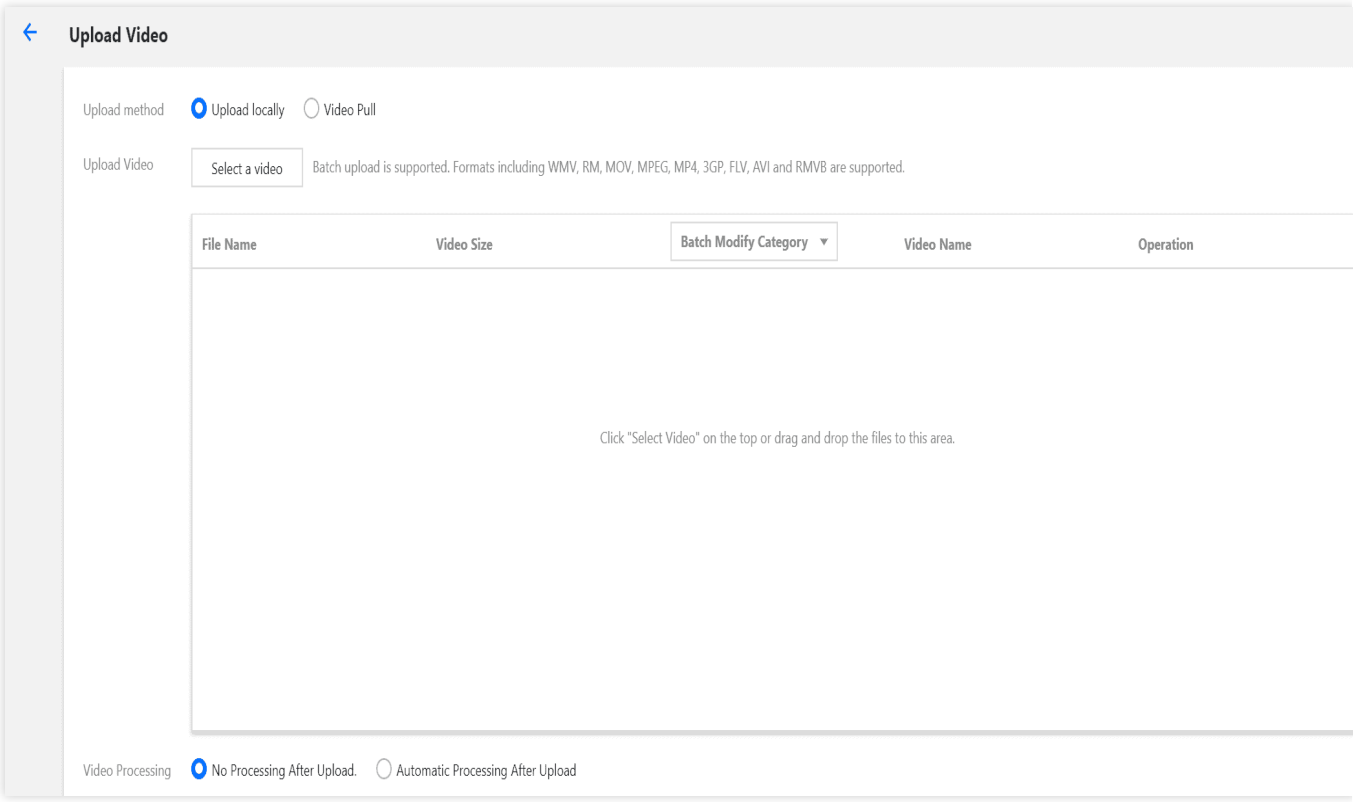
3. Click **OK**.

Initiating reliable callback

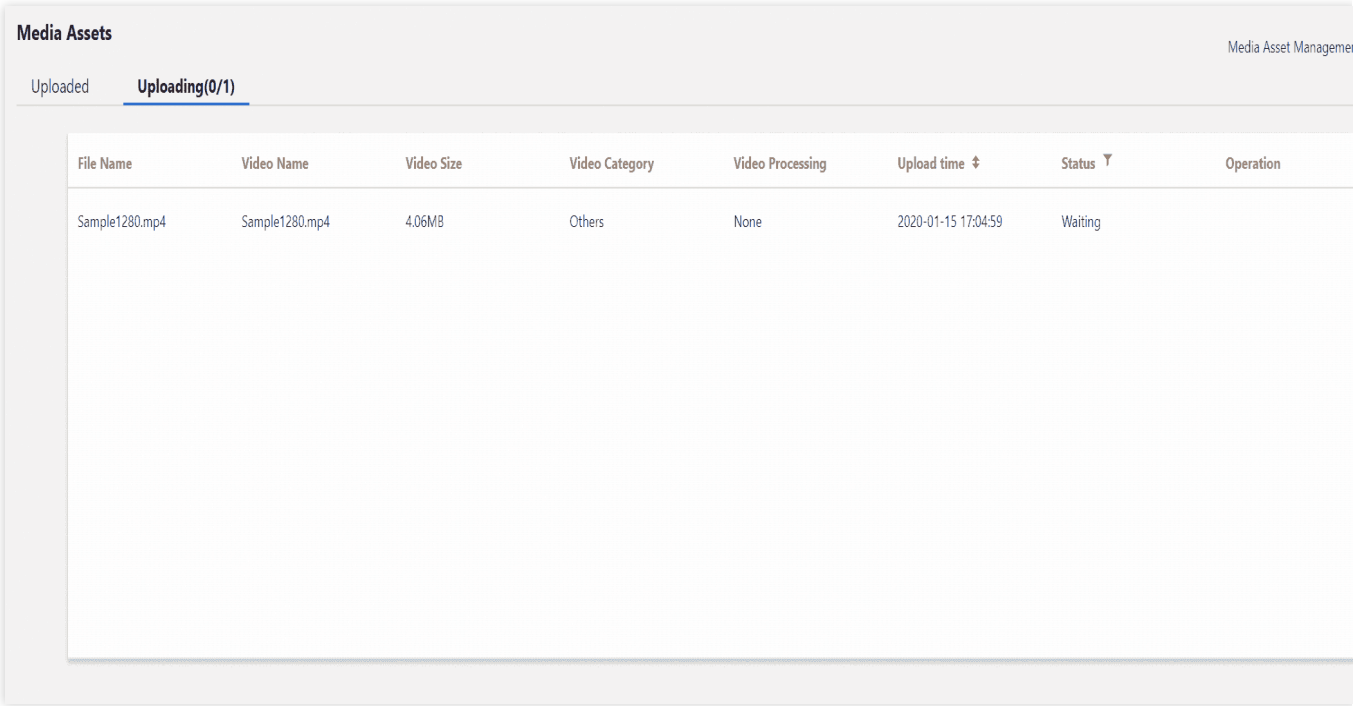
1. Select **Media Assets > Video Management** on the left sidebar, select **Uploaded**, and click **Upload Video**.



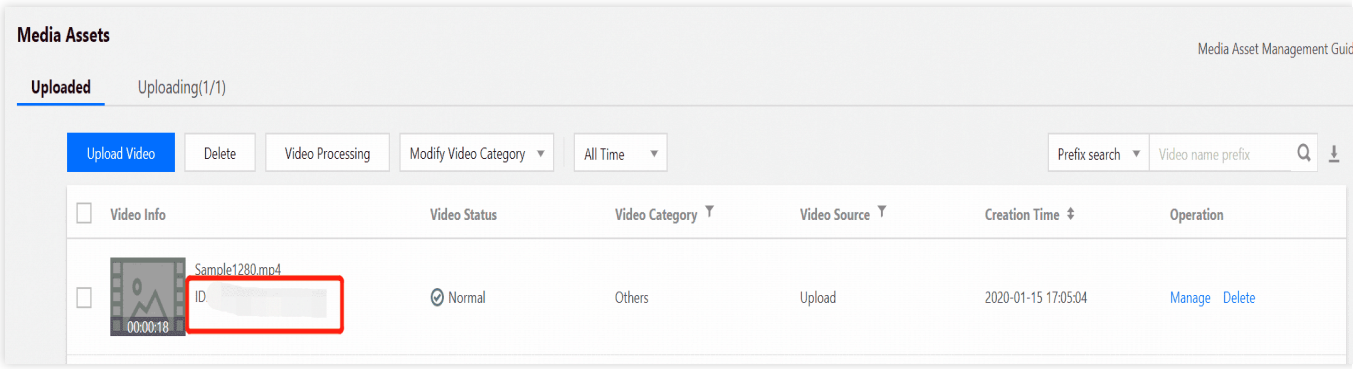
2. In the **Upload Video** dialog box that pops up, select **Local Upload**, click **Select Video**, and upload the **demo video** to the VOD platform.



After performing the upload operation, you will see the video upload progress in the **Uploading** column.



After the upload is completed, you will see the uploaded video and its corresponding ID (i.e., `FileId`) in the video list in the **Uploaded** column.



3. In the **Uploaded** column in **Media Assets**, select the video just uploaded and click **Video Processing**. Select **Manually select transcoding template** for **Processing Type**, check **MP4-LD-FLU (10)** in **Transcoding Template**, keep **Video Cover** checked, and click **OK**.

At this point, you have uploaded a video again and initiated a transcoding task for it. These operations have triggered event notifications.

Video Upload Completion

Last updated：2023-03-13 11:41:15

Event Name

NewFileUpload

Event Description

If you have configured event notifications for your application, after a video is uploaded from a client or the server, your application backend will be notified either by a “normal callback” or a “reliable callback”. For the content of the callback, see FileUploadTask.

Normal Callback

In the normal callback mode, your callback URL will receive an HTTP POST request from VOD. The content of the callback is included in the request body, as shown below (fields with null values are omitted):


```
{
    "EventType":"NewFileUpload",
    "FileUploadEvent":{
        "FileId":"5285890784273533167",
        "MediaBasicInfo":{
            "Name":"Animal World",
            "Description":"",
            "CreateTime":"2019-01-09T16:36:22Z",
            "UpdateTime":"2019-01-09T16:36:24Z",
            "ExpireTime":"9999-12-31T23:59:59Z",
            "ClassId":0,
            "ClassName":"Other",
            "ClassPath":"Other",
            "CoverUrl":"",
            "Type":"mp4",
            "MediaUrl":"http://125676836723.vod2.myqcloud.com/xxx/xxx/q1BORBPQH1IA.
            "TagSet":[
            ],
            "StorageRegion":"ap-guangzhou-2",
            "SourceInfo":{
                "SourceType":"Upload",
```

```
                "SourceContext":""
            },
            "Vid":"5285890784273533167"
        },
        "ProcedureTaskId":"",
        "ReviewAudioVideoTaskId":""
    }
}
```

Reliable Callback

In the reliable callback mode, after calling the PullEvents API, you will receive an HTTP response in the following format (fields with null values are omitted):

```
{
    "Response": {
        "EventSet":[
            {
                "EventHandle": "EventHandle.N",
                "EventType": "NewFileUpload",
                "FileUploadEvent":{
                    "FileId": "5285890784273533167",
                    "MediaBasicInfo":{
                        "Name": "Animal World",
                        "Description": "",
                        "CreateTime": "2019-01-09T16:36:22Z",
                        "UpdateTime": "2019-01-09T16:36:24Z",
                        "ExpireTime": "9999-12-31T23:59:59Z",
                        "ClassId": 0,
                        "ClassName": "Other",
                        "ClassPath": "Other",
                        "CoverUrl": "",
                        "Type": "mp4",
                        "MediaUrl": "http://125676836723.vod2.myqcloud.com/xxx/xxx/
                        "TagSet": [],
                        "StorageRegion": "ap-guangzhou-2",
                        "SourceInfo":{
                            "SourceType": "Upload",
                            "SourceContext": ""
                        },
                        "Vid": "5285890784273533167"
                    },
                    "ProcedureTaskId": "",
                    "ReviewAudioVideoTaskId":""
```

```
                    }
                }
            ],
            "RequestId": "335bdaa3-db0e-46ce-9946-51941d9cb0f5"
        }
    }
```

Video Pull from URL Completion

Last updated：2023-03-13 11:41:15

Event Name

PullComplete

Event Description

If you have configured event notifications for your application, after a video pull and upload task is completed, your application backend will be notified either by a “normal callback” or a “reliable callback”. For the content of the callback, see PullComplete.

Example

Normal callback

In the normal callback mode, your callback URL will receive an HTTP POST request from VOD. The content of the callback is included in the request body, as shown below (fields with null values are omitted):


```
{
    "EventType": "PullComplete",
    "PullCompleteEvent": {
        "TaskId": "125676836723-Pull-f5ac8127b3b6b85cdc13f237c6005d8",
        "Status": "FINISH",
        "ErrCode": 0,
        "Message": "SUCCESS",
        "FileId": "14508071098244959037",
        "MediaBasicInfo":{
            "Name": "Animal World",
            "Description": "",
            "CreateTime": "2019-01-09T16:36:22Z",
            "UpdateTime": "2019-01-09T16:36:24Z",
            "ExpireTime": "9999-12-31T23:59:59Z",
            "ClassId": 0,
            "ClassName": "Other",
            "ClassPath": "Other",
            "CoverUrl": "",
            "Type": "mp4",
```



```
            "MediaUrl": "http://125676836723.vod2.myqcloud.com/xxx/xxx/xxx.mp4",
            "TagSet": [ ],
            "StorageRegion": "ap-guangzhou-2",
            "SourceInfo":{
                "SourceType": "Upload",
                "SourceContext": ""
            },
            "Vid": ""
        },
        "FileUrl": "http://125676836723.vod2.myqcloud.com/xxx/xxx/xxx.mp4",
        "ProcedureTaskId": "",
        "ReviewAudioVideoTaskId":"",
        "SessionContext": "",
        "SessionId": ""
    }
}
```


Reliable callback

In the reliable callback mode, after calling the PullEvents API, you will receive an HTTP response in the following format (fields with null values are omitted):


```
{
    "Response": {
        "EventSet":[
            {
                "EventHandle": "EventHandleX",
                "EventType": "PullComplete",
                "PullCompleteEvent": {
                    "TaskId": "125676836723-Pull-f5ac8127b3b6b85cdc13f237c6005d8",
                    "Status": "FINISH",
                    "ErrCode": 0,
                    "Message": "SUCCESS",
                    "FileId": "14508071098244959037",
                    "MediaBasicInfo":{
                        "Name": "Animal World",
                        "Description": "",
                        "CreateTime": "2019-01-09T16:36:22Z",
                        "UpdateTime": "2019-01-09T16:36:24Z",
                        "ExpireTime": "9999-12-31T23:59:59Z",
                        "ClassId": 0,
                        "ClassName": "Other",
                        "ClassPath": "Other",
                        "CoverUrl": "",
                        "Type": "mp4",
                        "MediaUrl": "http://125676836723.vod2.myqcloud.com/xxx/xxx/
```

```
                        "TagSet": [ ],
                        "StorageRegion": "ap-guangzhou-2",
                        "SourceInfo":{
                            "SourceType": "Upload",
                            "SourceContext": ""
                        },
                        "Vid": ""
                    },
                    "FileUrl": "http://125676836723.vod2.myqcloud.com/xxx/xxx/xxx.m
                    "ProcedureTaskId": "",
                    "ReviewAudioVideoTaskId":"",
                    "SessionContext": "",
                    "SessionId": ""
                }
            }
        ]
    }
}
```

Video Deletion Completion

Last updated：2023-03-13 11:41:15

Event Name

FileDeleted

Event Description

If you have configured event notifications for your application, after a video is deleted, your application backend will be notified either by a “normal callback” or a “reliable callback”. For the content of the callback, see FileDeleteTask.

Example

Normal callback

In the normal callback mode, your callback URL will receive an HTTP POST request from VOD. The content of the callback is included in the request body, as shown below (fields with null values are omitted):

```
{
    "EventType":"FileDeleted",
    "FileDeleteEvent":{
        "FileIdSet":[
            "24961954183381008"
        ],
        "FileDeleteResultInfo":[
            {
                "FileId":"24961954183381008",
                "DeleteParts":[
                    {
                        "Type":"TranscodeFiles",
                        "Definition":0
                    }
                ]
            }
        ]
    }
}
```

Reliable callback

In the reliable callback mode, after calling the PullEvents API, you will receive an HTTP response in the following format (fields with null values are omitted):

```
{
    "Response":{
        "EventSet":[
            {
                "EventHandle":"EventHandle.N",
                "EventType":"FileDeleted",
                "FileDeleteEvent":{
                    "FileIdSet":[
                        "24961954183381008"
                    ],
                    "FileDeleteResultInfo":[
                        {
                            "FileId":"24961954183381008",
                            "DeleteParts":[
                                {
                                    "Type":"TranscodeFiles",
                                    "Definition":0
                                }
                            ]
                        }
                    ]
                }
            }
        ],
        "RequestId":"335bdaa3-db0e-46ce-9946-51941d9cb0f5"
    }
}
```

Task Flow Status Change

Last updated：2023-03-13 11:41:15

Event Name

ProcedureStateChanged

Event Description

If you have configured event notifications for your application, after the status of a task flow is changed, your application backend will be notified either by a “normal callback” or a “reliable callback”. For the content of the callback, see ProcedureTask.

Example

Normal callback

In the normal callback mode, your callback URL will receive an HTTP POST request from VOD. The content of the callback is included in the request body, as shown below (fields with null values are omitted):

```
{
    "EventType":"ProcedureStateChanged",
    "ProcedureStateChangeEvent":{
        "TaskId":"1256768367-Procedure-475b72xxxcb177t1",
        "Status":"FINISH",
        "ErrCode":0,
        "Message":"",
        "FileId":"5285890784246869930",
        "FileName":"Animal World",
        "FileUrl":"https://1256768367.vod2.myqcloud.com/xxx/xxx/xxx.mp4",
        "MetaData":{
            "AudioDuration":59.990001678467,
            "AudioStreamSet":[
                {
                    "Bitrate":383854,
                    "Codec":"aac",
                    "SamplingRate":48000
                }
            ],
```

```
            "Bitrate":1021028,
            "Container":"mov,mp4,m4a,3gp,3g2,mj2",
            "Duration":60,
            "Height":480,
            "Rotate":0,
            "Size":7700180,
            "VideoDuration":60,
            "VideoStreamSet":[
                {
                    "Bitrate":637174,
                    "Codec":"h264",
                    "Fps":23,
                    "Height":480,
                    "Width":640
                }
            ],
            "Width":640
        },
        "MediaProcessResultSet":[
            {
                "Type":"Transcode",
                "TranscodeTask":{
                    "Status":"SUCCESS",
                    "ErrCode":0,
                    "Message":"SUCCESS",
                    "Input":{
                        "Definition":20
                    },
                    "Output":{
                        "Url":"https://1256768367.vod2.myqcloud.com/xxx/xxx/v.f20.m
                        "Size":4189073,
                        "Container":"mov,mp4,m4a,3gp,3g2,mj2",
                        "Height":480,
                        "Width":640,
                        "Bitrate":552218,
                        "Md5":"eff7031ad7877865f9a3240e9ab165ad",
                        "Duration":60.04700088501,
                        "VideoStreamSet":[
                            {
                                "Bitrate":503727,
                                "Codec":"h264",
                                "Fps":24,
                                "Height":480,
                                "Width":640
                            }
                        ],
                        "AudioStreamSet":[
```

```
                                {
                                    "Bitrate":48491,
                                    "Codec":"aac",
                                    "SamplingRate":44100
                                }
                            ],
                            "Definition":0
                        }
                    }
                },
                {
                    "Type":"CoverBySnapshot",
                    "CoverBySnapshotTask":{
                        "Status":"SUCCESS",
                        "ErrCode":0,
                        "Message":"SUCCESS",
                        "Input":{
                            "Definition":10,
                            "PositionType":"Time",
                            "PositionValue":0
                        },
                        "Output":{
                            "CoverUrl":"http://1256768367.vod2.myqcloud.com/xxx/xxx/xxx
                        }
                    }
                }
            ]
        }
}
```


Reliable callback

In the reliable callback mode, after calling the PullEvents API, you will receive an HTTP response in the following format (fields with null values are omitted):


```
{
    "Response": {
        "EventSet":[
            {
                "EventHandle": "EventHandleX",
                "EventType": "ProcedureStateChanged",
                "ProcedureStateChangeEvent":{
                    "TaskId": "1256768367-Procedure-475b72xxxcb177t1",
                    "Status": "FINISH",
                    "FileId": "5285890784246869930",
                    "FileName": "Animal World",
```

```
                    "FileUrl": "https://1256768367.vod2.myqcloud.com/xxx/xxx/xxx.mp
                    "MetaData":{
                        "AudioDuration": 59.990001678467,
                        "AudioStreamSet": [{
                            "Bitrate": 383854,
                            "Codec": "aac",
                            "SamplingRate": 48000
                        }],
                        "Bitrate": 1021028,
                        "Container": "mov,mp4,m4a,3gp,3g2,mj2",
                        "Duration": 60,
                        "Height": 480,
                        "Rotate": 0,
                        "Size": 7700180,
                        "VideoDuration": 60,
                        "VideoStreamSet": [{
                            "Bitrate": 637174,
                            "Codec": "h264",
                            "Fps": 23,
                            "Height": 480,
                            "Width": 640
                        }],
                        "Width": 640
                    },
                    "MediaProcessResultSet": [{
                            "Type": "Transcode",
                            "TranscodeTask":{
                                "Status": "SUCCESS",
                                "ErrCode": 0,
                                "Message": "SUCCESS",
                                "Input":{
                                    "Definition": 20
                                },
                                "Output": {
                                    "Url": "https://1256768367.vod2.myqcloud.com/xx
                                    "Size": 4189073,
                                    "Container": "mov,mp4,m4a,3gp,3g2,mj2",
                                    "Height": 480,
                                    "Width": 640,
                                    "Bitrate": 552218,
                                    "Md5": "eff7031ad7877865f9a3240e9ab165ad",
                                    "Duration": 60.04700088501,
                                    "VideoStreamSet": [{
                                        "Bitrate": 503727,
                                        "Codec": "h264",
                                        "Fps": 24,
                                        "Height": 480,
```

```
                                    "Width": 640
                                }],
                                "AudioStreamSet": [{
                                    "Bitrate": 48491,
                                    "Codec": "aac",
                                    "SamplingRate": 44100
                                }],
                                "Definition": 0
                            }
                        }
                    },
                    {
                        "Type": "CoverBySnapshot",
                        "CoverBySnapshotTask":{
                            "Status": "SUCCESS",
                            "ErrCode": 0,
                            "Message": "SUCCESS",
                            "Input":{
                                "Definition": 10,
                                "PositionType": "Time",
                                "PositionValue":0
                            },
                            "Output": {
                                "CoverUrl": "http://1256768367.vod2.myqcloud.co
                            }
                        }
                    }
                ]
            }
        }
    ],
    "RequestId": "335bdaa3-db0e-46ce-9946-51941d9cb0f5"
}
}
```

Video Editing Completion

Last updated：2023-01-05 11:11:53

Event Name

EditMediaComplete

Event Description

If the application is configured with event notification, after a video is edited, the application backend can get an event notification through "normal callback" or "reliable callback". The content of the event notification is the `EditMediaTask` structure.

Samples

Normal callback

If you choose the normal callback mode, the callback URL will receive an HTTP POST request from VOD, whose content is in the `BODY` as shown below (the fields with null value are omitted):

```
{
    "EventType":"EditMediaComplete",
    "EditMediaCompleteEvent":{
        "TaskId":"1256768367-EditMedia-f5ac8127b3b6b85cdc13f237c6005d8",
        "Status":"FINISH",
        "ErrCode":0,
        "Message":"SUCCESS",
        "Input":{
            "InputType":"File",
            "FileInfoSet":[
                {
                    "FileId":"2496195418338100 8",
                    "StartTimeOffset":0,
                    "EndTimeOffset":0
                },
                {
                    "FileId":"2496195418338100 9",
                    "StartTimeOffset":0,
                    "EndTimeOffset":0
```

```
                },
                {
                    "FileId":"24961954183381010",
                    "StartTimeOffset":0,
                    "EndTimeOffset":0
                }
            ]
        },
        "Output":{
            "FileType":"mp4",
            "FileId":"24961954183923290",
            "FileUrl":"http://125676836723.vod2.myqcloud.com/xxx/xxx/f0.mp4"
        },
        "ProcedureTaskId":"",
        "ReviewAudioVideoTaskId":""
    }
}
```

Reliable callback

If you choose the reliable callback mode, after the PullEvents API is called, an HTTP response in the following format will be received (the fields with null value are omitted).

```
{
    "Response": {
        "EventSet": [
            {
                "EventHandle": "EventHandle.N",
                "EventType": "EditMediaComplete",
                "EditMediaCompleteEvent": {
                    "TaskId": "EditMedia-f5ac8127b3b6b85cdc13f237c6005d8",
                    "Status": "FINISH",
                    "ErrCode": 0,
                    "Message": "SUCCESS",
                    "Input": {
                        "InputType": "File",
                        "FileInfoSet": [
                            {
                                "FileId": "24961954183381008",
                                "StartTimeOffset": 0,
                                "EndTimeOffset": 0
                            },
                            {
                                "FileId": "24961954183381009",
                                "StartTimeOffset": 0,
                                "EndTimeOffset": 0
```

```
                            },
                            {
                                "FileId": "24961954183381010",
                                "StartTimeOffset": 0,
                                "EndTimeOffset": 0
                            }
                        ]
                    },
                    "Output": {
                        "FileType": "mp4",
                        "FileId": "24961954183923290",
                        "FileUrl": "http://125676836723.vod2.myqcloud.com/xxx/xxx/f
                    },
                    "ProcedureTaskId": "",
                "ReviewAudioVideoTaskId": ""
                }
            }
        ],
        "RequestId": "335bdaa3-db0e-46ce-9946-51941d9cb0f5"
    }
}
```

Video Compositing Completion

Last updated：2022-10-26 17:20:53

Event Name

ComposeMediaComplete

Event Description

If the application is configured with event notification, after a video is composed, the application backend can get an event notification through "normal callback" or "reliable callback". The content of the event notification is the `ComposeMediaTask` structure.

Samples

Normal callback

If you choose the normal callback mode, the callback URL will receive an HTTP POST request from VOD, whose content is in the `BODY` as shown below (the fields with null value are omitted):

```
{
    "EventType": "ComposeMediaComplete",
    "ComposeMediaCompleteEvent": {
        "TaskId": "1256768367-ComposeMedia-f5ac8127b3b6b85cdc13f237c6005d8",
        "Status": "FINISH",
        "ErrCode": 0,
        "Message": "SUCCESS",
        "Input": {
            "Tracks": [{
                    "Type": "Video",
                    "TrackItems": [{
                        "Type": "Video",
                        "SourceMedia": "5285485487985271487",
                        "AudioOperations": [{
                            "Type": "Volume",
                            "VolumeParam": {
                                "Mute": 1
                            }
                        }]
```

```
                    }]
                },
                {
                    "Type": "Audio",
                    "TrackItems": [{
                            "Type": "Empty",
                            "EmptyItem": {
                                "Duration": 5
                            }
                        },
                        {
                            "Type": "Audio",
                            "AudioItem": {
                                "SourceMedia": "5285485487985271488",
                                "Duration": 15
                            }
                        },
                        {
                            "Type": "Audio",
                            "AudioItem": {
                                "SourceMedia": "5285485487985271489",
                                "SourceMediaStartTime": 2,
                                "Duration": 14
                            }
                        }
                    ]
                }
            ],
            "Output": {
                "FileName": "Video composing effect test",
                "Container": "mp4"
            }
        },
        "Output": {
            "FileType": "mp4",
            "FileId": 5285485487985271490,
            "FileUrl": "http://125676836723.vod2.myqcloud.com/xxx/xxx/xxx.mp4"
        }
    }
}
```

Reliable callback

If you choose the reliable callback mode, after the PullEvents API is called, an HTTP response in the following format will be received (the fields with null value are omitted).

```
{
    "Response":{
        "EventSet":[
            {
                "EventHandle":"EventHandle.N",
                "ComposeMediaCompleteEvent":{
                    "TaskId":"1256768367-ComposeMedia-f5ac8127b3b6b85cdc13f237c6005
                    "Status":"FINISH",
                    "ErrCode":0,
                    "Message":"SUCCESS",
                    "Input":{
                        "Tracks":[
                            {
                                "Type":"Video",
                                "TrackItems":[
                                    {
                                        "Type":"Video",
                                        "SourceMedia":"5285485487985271487",
                                        "AudioOperations":[
                                            {
                                                "Type":"Volume",
                                                "VolumeParam":{
                                                    "Mute":1
                                                }
                                            }
                                        ]
                                    }
                                ]
                            },
                            {
                                "Type":"Audio",
                                "TrackItems":[
                                    {
                                        "Type":"Empty",
                                        "EmptyItem":{
                                            "Duration":5
                                        }
                                    },
                                    {
                                        "Type":"Audio",
                                        "AudioItem":{
                                            "SourceMedia":"5285485487985271488",
                                            "Duration":15
                                        }
                                    },
                                    {
```

```
                                        "Type":"Audio",
                                        "AudioItem":{
                                            "SourceMedia":"5285485487985271489",
                                            "SourceMediaStartTime":2,
                                            "Duration":14
                                        }
                                    }
                                ]
                            }
                        ],
                        "Output":{
                            "FileName":"Video composing effect test",
                            "Container":"mp4"
                        }
                    },
                    "Output":{
                        "FileType":"mp4",
                        "FileId":5285485487985271490,
                        "FileUrl":"http://125676836723.vod2.myqcloud.com/xxx/xxx/xx
                    }
                }
            }
        ],
        "RequestId":"335bdaa3-db0e-46ce-9946-51941d9cb0f5"
    }
}
```

Video Retrieval Completion

Last updated：2023-03-13 11:41:15

Event Name

RestoreMediaComplete

Event Description

If you have configured event notifications for your application, after a media file is retrieved from ARCHIVE or DEEP ARCHIVE, your application backend will be notified either by a “normal callback” or a “reliable callback”. For the content of the callback, see RestoreMediaTask.

Example

Normal callback

In the normal callback mode, your callback URL will receive an HTTP request in the following format.

```
{
    "EventType":"RestoreMediaComplete",
    "RestoreMediaCompleteEvent":{
        "FileId":"24961954183381008",
        "OriginalStorageClass":"ARCHIVE",
        "TargetStorageClass":"STANDARD",
        "RestoreTier":"Standard",
        "RestoreDay":0,
        "Status":0,
        "Message":"Restore success!"
    }
}
```

Reliable callback

In the reliable callback mode, after calling the PullEvents API, you will receive an HTTP response in the following format:

```
{
    "Response":{
```

```
        "EventSet":[
            {
                "EventHandle":"EventHandle.N",
                "EventType":"RestoreMediaComplete",
                "RestoreMediaCompleteEvent":{
                    "FileId":"24961954183381008",
                    "OriginalStorageClass":"ARCHIVE",
                    "TargetStorageClass":"STANDARD",
                    "RestoreTier":"Standard",
                    "RestoreDay":0,
                    "Status":0,
                    "Message":"Restore success!"
                }
            }
        ],
        "RequestId":"335bdaa3-db0e-46ce-9946-51941d9cb0f5"
    }
}
```

Moderation Completion

Last updated：2022-10-13 15:08:47

Event Name

ReviewAudioVideoComplete

Event Description

If you have configured event notifications for your application, after a moderation task is completed, your application backend will be notified either by a “normal callback” or a “reliable callback”. For the content of the callback, see ReviewAudioVideoTask.

Examples

Normal callback

In the normal callback mode, your callback URL will receive an HTTP POST request from VOD. The content of the callback is included in the request body, as shown below (fields with null values are omitted):

```
{
    "EventType": "ReviewAudioVideoComplete",
    "ReviewAudioVideoCompleteEvent": {
        "TaskId": "125xxxx-ReviewAudioVideo-07edbc78ba20563cdf2362cffbf4aa0ct",
        "Status": "FINISH",
        "ErrCodeExt": "",
        "Message": "SUCCESS",
        "Input":{
            "FileId": "387702130626135215"
        },
        "Output":{
            "Suggestion": "block",
            "Label": "porn",
            "Form": "Image",
            "SegmentSet": [
                {
                    "StartTimeOffset": 0,
                    "EndTimeOffset": 1,
                    "Confidence": 99,
```

```
                "Suggestion": "block",
                "Label": "Porn",
                "SubLabel": "porn",
                "Form": "Image",
                "AreaCoordSet": [],
                "Text": "",
                "KeywordSet": []
            },
            {
                "StartTimeOffset": 1,
                "EndTimeOffset": 2,
                "Confidence": 99,
                "Suggestion": "block",
                "Label": "Porn",
                "SubLabel": "porn",
                "Form": "Image",
                "AreaCoordSet": [],
                "Text": "",
                "KeywordSet": []
            },
            {
                "StartTimeOffset": 2,
                "EndTimeOffset": 3,
                "Confidence": 99,
                "Suggestion": "block",
                "Label": "Porn",
                "SubLabel": "porn",
                "Form": "Image",
                "AreaCoordSet": [],
                "Text": "",
                "KeywordSet": []
            },
            {
                "StartTimeOffset": 3,
                "EndTimeOffset": 4,
                "Confidence": 99,
                "Suggestion": "block",
                "Label": "Porn",
                "SubLabel": "porn",
                "Form": "Image",
                "AreaCoordSet": [],
                "Text": "",
                "KeywordSet": []
            },
            {
                "StartTimeOffset": 4,
                "EndTimeOffset": 5,
```

```
                    "Confidence": 99,
                    "Suggestion": "block",
                    "Label": "Porn",
                    "SubLabel": "porn",
                    "Form": "Image",
                    "AreaCoordSet": [],
                    "Text": "",
                    "KeywordSet": []
                },
                {
                    "StartTimeOffset": 5,
                    "EndTimeOffset": 6,
                    "Confidence": 99,
                    "Suggestion": "block",
                    "Label": "Porn",
                    "SubLabel": "porn",
                    "Form": "Image",
                    "AreaCoordSet": [],
                    "Text": "",
                    "KeywordSet": []
                },
                {
                    "StartTimeOffset": 6,
                    "EndTimeOffset": 7,
                    "Confidence": 99,
                    "Suggestion": "block",
                    "Label": "Porn",
                    "SubLabel": "porn",
                    "Form": "Image",
                    "AreaCoordSet": [],
                    "Text": "",
                    "KeywordSet": []
                },
                {
                    "StartTimeOffset": 7,
                    "EndTimeOffset": 8,
                    "Confidence": 99,
                    "Suggestion": "block",
                    "Label": "Porn",
                    "SubLabel": "porn",
                    "Form": "Image",
                    "AreaCoordSet": [],
                    "Text": "",
                    "KeywordSet": []
                },
                {
                    "StartTimeOffset": 8,
```

```
                    "EndTimeOffset": 9,
                    "Confidence": 99,
                    "Suggestion": "block",
                    "Label": "Porn",
                    "SubLabel": "porn",
                    "Form": "Image",
                    "AreaCoordSet": [],
                    "Text": "",
                    "KeywordSet": []
                },
                {
                    "StartTimeOffset": 9,
                    "EndTimeOffset": 10,
                    "Confidence": 99,
                    "Suggestion": "block",
                    "Label": "Porn",
                    "SubLabel": "porn",
                    "Form": "Image",
                    "AreaCoordSet": [],
                    "Text": "",
                    "KeywordSet": []
                }
            ],
            "SegmentSetFileUrl": "http://251000800.vod2.myqcloud.com/a8800b40vodtra
            "SegmentSetFileUrlExpireTime": "2022-10-12T07:01:07.695Z"
        },
        "SessionContext": "",
        "SessionId": ""
    }
}
```


Reliable callback

In the reliable callback mode, after calling the PullEvents API, you will receive an HTTP response in the following format (fields with null values are omitted):


```
{
    "Response": {
        "EventSet":[
            {
                "EventHandle": "EventHandle.N",
                "EventType": "ReviewAudioVideoComplete",
                "ReviewAudioVideoCompleteEvent": {
                    "TaskId": "125xxxx-ReviewAudioVideo-07edbc78ba20563cdf2362cffbf
                    "Status": "FINISH",
                    "ErrCodeExt": "",
```

```
    "Message": "SUCCESS",
    "Input":{
        "FileId": "387702130626135215"
    },
    "Output":{
        "Suggestion": "block",
        "Label": "porn",
        "Form": "Image",
        "SegmentSet": [
            {
                "StartTimeOffset": 0,
                "EndTimeOffset": 1,
                "Confidence": 99,
                "Suggestion": "block",
                "Label": "Porn",
                "SubLabel": "porn",
                "Form": "Image",
                "AreaCoordSet": [],
                "Text": "",
                "KeywordSet": []
            },
            {
                "StartTimeOffset": 1,
                "EndTimeOffset": 2,
                "Confidence": 99,
                "Suggestion": "block",
                "Label": "Porn",
                "SubLabel": "porn",
                "Form": "Image",
                "AreaCoordSet": [],
                "Text": "",
                "KeywordSet": []
            },
            {
                "StartTimeOffset": 2,
                "EndTimeOffset": 3,
                "Confidence": 99,
                "Suggestion": "block",
                "Label": "Porn",
                "SubLabel": "porn",
                "Form": "Image",
                "AreaCoordSet": [],
                "Text": "",
                "KeywordSet": []
            },
            {
                "StartTimeOffset": 3,
```

```
                "EndTimeOffset": 4,
                "Confidence": 99,
                "Suggestion": "block",
                "Label": "Porn",
                "SubLabel": "porn",
                "Form": "Image",
                "AreaCoordSet": [],
                "Text": "",
                "KeywordSet": []
            },
            {
                "StartTimeOffset": 4,
                "EndTimeOffset": 5,
                "Confidence": 99,
                "Suggestion": "block",
                "Label": "Porn",
                "SubLabel": "porn",
                "Form": "Image",
                "AreaCoordSet": [],
                "Text": "",
                "KeywordSet": []
            },
            {
                "StartTimeOffset": 5,
                "EndTimeOffset": 6,
                "Confidence": 99,
                "Suggestion": "block",
                "Label": "Porn",
                "SubLabel": "porn",
                "Form": "Image",
                "AreaCoordSet": [],
                "Text": "",
                "KeywordSet": []
            },
            {
                "StartTimeOffset": 6,
                "EndTimeOffset": 7,
                "Confidence": 99,
                "Suggestion": "block",
                "Label": "Porn",
                "SubLabel": "porn",
                "Form": "Image",
                "AreaCoordSet": [],
                "Text": "",
                "KeywordSet": []
            },
            {
```

```
                                "StartTimeOffset": 7,
                                "EndTimeOffset": 8,
                                "Confidence": 99,
                                "Suggestion": "block",
                                "Label": "Porn",
                                "SubLabel": "porn",
                                "Form": "Image",
                                "AreaCoordSet": [],
                                "Text": "",
                                "KeywordSet": []
                            },
                            {
                                "StartTimeOffset": 8,
                                "EndTimeOffset": 9,
                                "Confidence": 99,
                                "Suggestion": "block",
                                "Label": "Porn",
                                "SubLabel": "porn",
                                "Form": "Image",
                                "AreaCoordSet": [],
                                "Text": "",
                                "KeywordSet": []
                            },
                            {
                                "StartTimeOffset": 9,
                                "EndTimeOffset": 10,
                                "Confidence": 99,
                                "Suggestion": "block",
                                "Label": "Porn",
                                "SubLabel": "porn",
                                "Form": "Image",
                                "AreaCoordSet": [],
                                "Text": "",
                                "KeywordSet": []
                            }
                        ],
                        "SegmentSetFileUrl": "http://251000800.vod2.myqcloud.com/a8
                        "SegmentSetFileUrlExpireTime": "2022-10-12T07:01:07.695Z"
                    },
                    "SessionContext": "",
                    "SessionId": ""
                }
            }
        ],
        "RequestId": "335bdaa3-db0e-46ce-9946-51941d9cb0f5"
    }
}
```

Video Playback
Video Playback Overview

Last updated：2024-12-04 10:19:48

VOD supports multiple methods to play back uploaded and transcoded videos, and video playback mainly involves three scenarios: short video, long video, and encrypted video playback.

Short video playback

Short videos generally refer to videos of less than 5 minutes in length, mainly including:
Videos shared on UGSV social media sites (such as TikTok).
Product promotion videos shared on ecommerce platforms.
Videos shared on WeChat Official Account and we media.

Long video playback

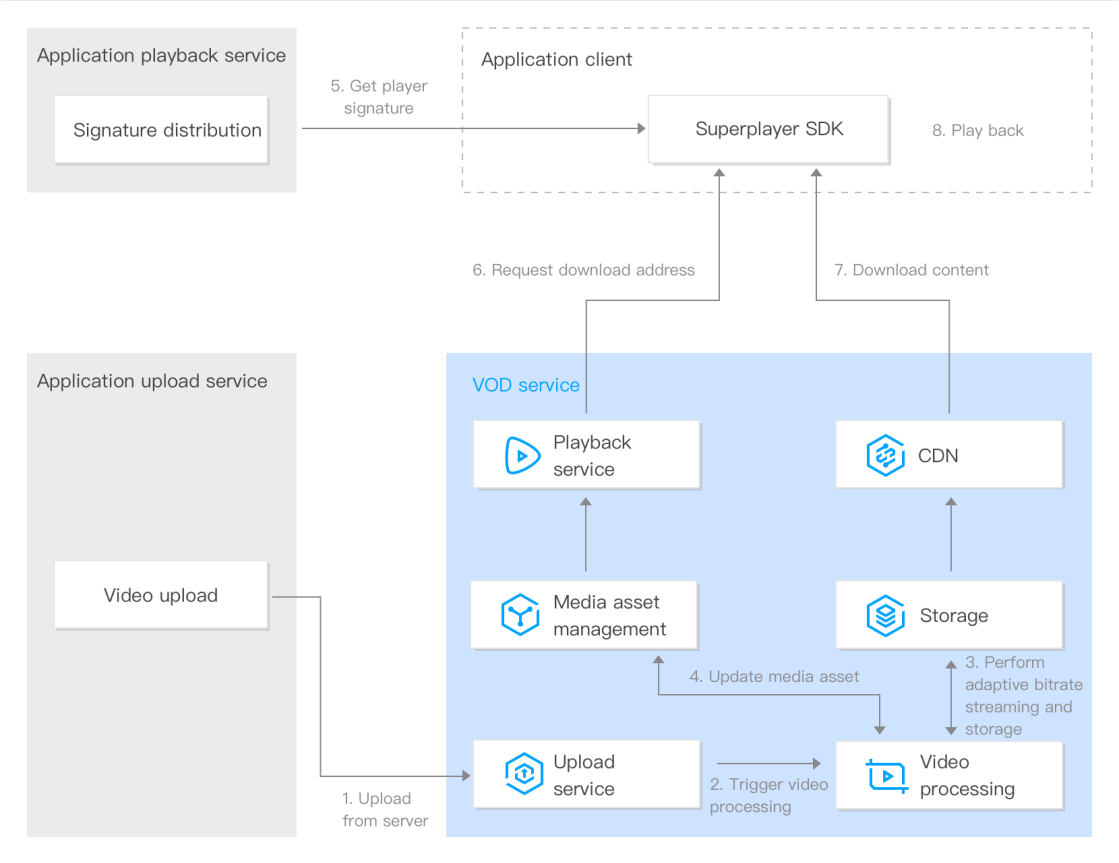
Long videos generally refer to videos produced by professional organizations and published on video websites, mainly including:
Exclusive TV series and variety shows published on video social media platforms (such as Tencent Video, Youku, and iQIYI).
Course videos published on online education websites (such as Tencent Class and Penguin Tutoring).
TV programs replayed on online TV platforms (such as CNTV and Mongo TV).

Encrypted video playback

Video encryption is a specific scenario of long video playback scenarios, where copyrighted videos such as exclusive TV series and online courses are encrypted to avoid unauthorized download and distribution.

Playback Architecture

For various video playback scenarios, we recommend you use the **Player SDK** to play back the output video of adaptive bitrate streaming in VOD. The overall playback architecture is as follows:



1. **Upload from server**: The business backend uploads a video to VOD through the console, server API, or other means.
2. **Trigger video processing**: When the video is uploaded, adaptive bitrate streaming is specified. After the video is uploaded, video processing begins.
3. **Transcode to adaptive bitstream and write to storage**: After the video is transcoded to adaptive bitstream, the output video content is written to the VOD storage.
4. **Update the media asset**: The output video information is written into the media asset management module.
5. **Distribute the signature**: The business backend distributes the playback signature generated according to the player signature calculation rule.
6. **Request the download address**: The player gets the download address of the video from VOD's playback service after the video's `FileId` is specified.
7. **Download the content**: The player downloads the content from VOD CDN at the download address.
8. **Play back the video**: The player plays back the output adaptive bitstream.

Documentation

For the features supported by the Player SDK, see Feature Description. For the integration method, see SDK Download.

To help you quickly integrate the VOD Player, we provide an integration guide for the Player SDK to describe the integration steps by way of demos.
For more information on how video encryption works and the integration methods in video encryption and playback scenarios, see Overview and Stage 4. Play back an encrypted video.

Adding a Domain

Last updated：2023-08-31 16:45:24

Why do you need to add a domain?

When you are still using the default distribution domain name of VOD to accelerate the distribution of media content, add your own domain name for distribution to ensure more flexible business and avoid the impact of business distribution caused by the ban of the default distribution domain name of VOD risk.

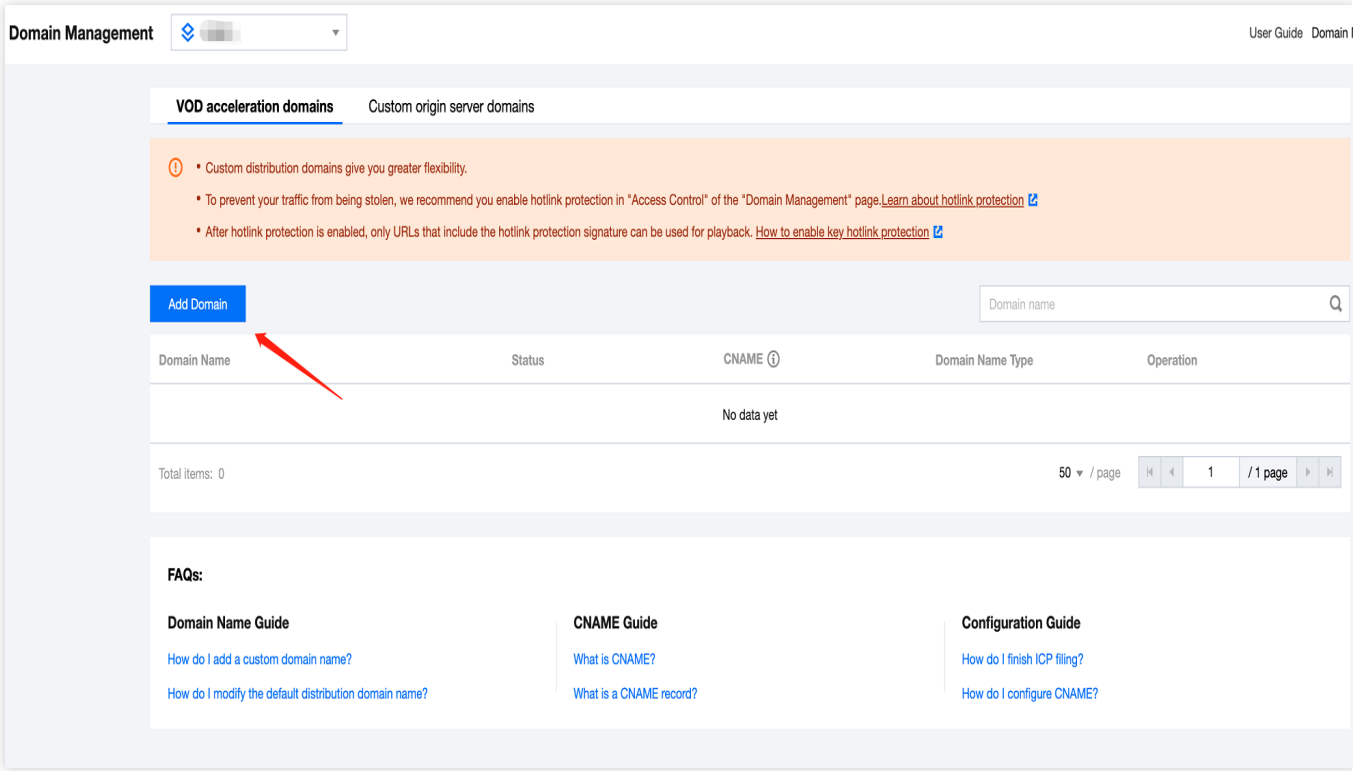
Preparation

Prepare a domain that has been registered and can be used for video on demand acceleration, for example: example.com

Method 1: Add a domain name through the cloud Video on Demand Console

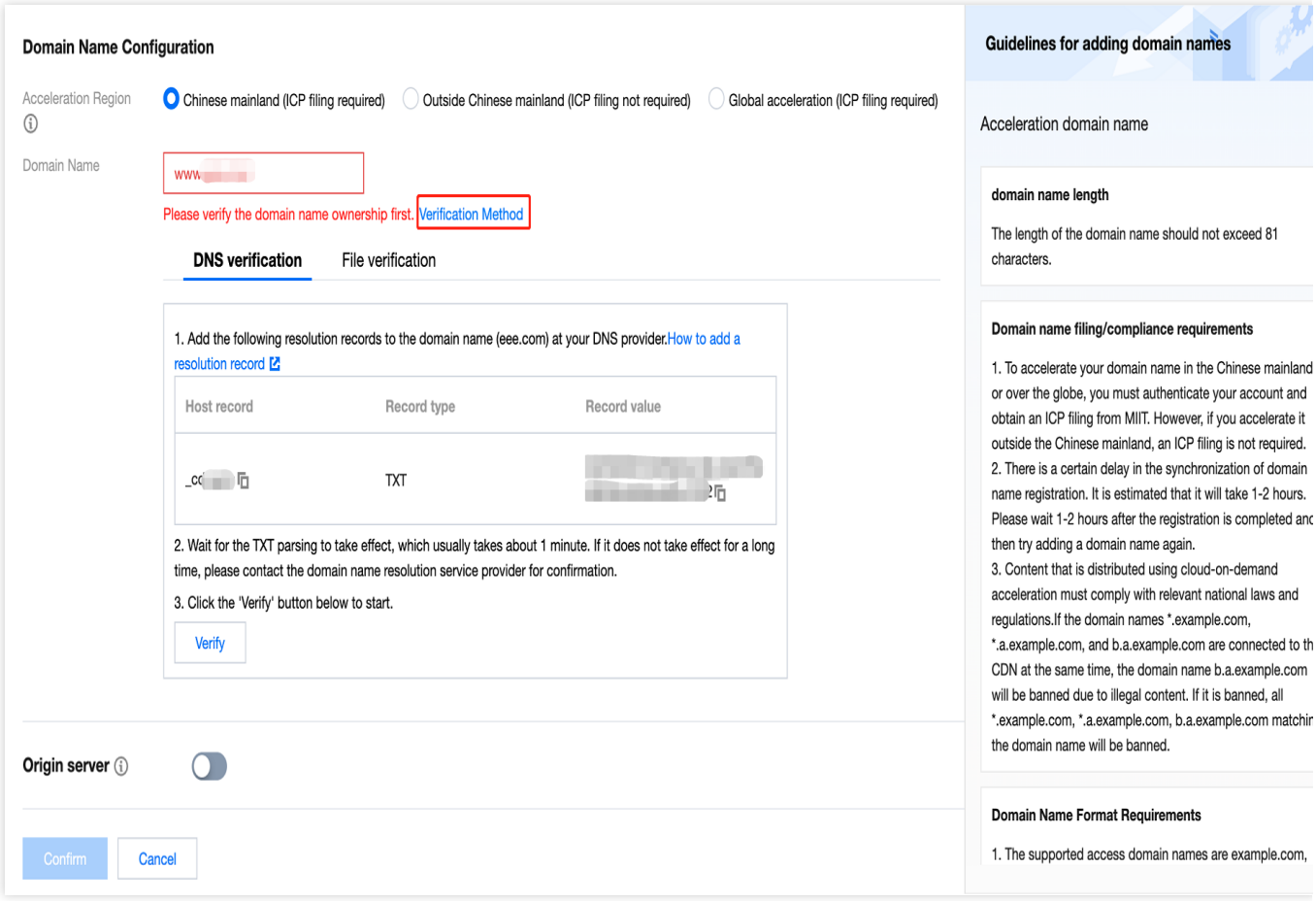
1.Add domain

Video on Demand Console > Distribution playback settings > Aomain management > Add domain or Add a custom origin site acceleration domain. After entering the domain name, domain attribution resolution verification is required.



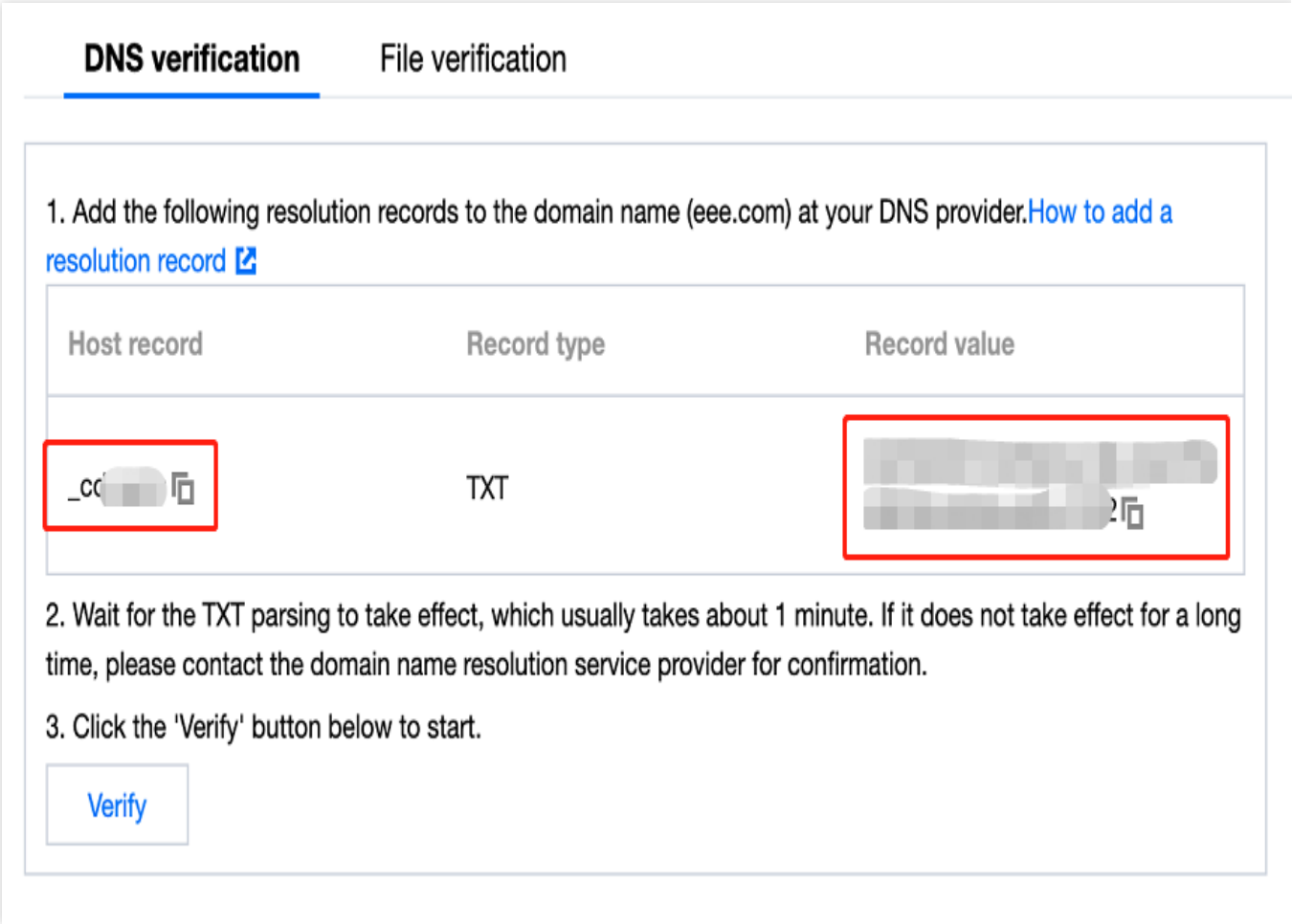
2. DNS Verification

（1）Click Verification Method



（2）Use the default verification method, which is DNS verification

To start DNS verification, add the " `_cdnauth` " host record of the TXT type for the domain name at your DNS provider.



Note：

In scenarios where multi-level domain names are used, host records must be added only for the domain name regardless of the level of the added domain name, such as `c.b.a.example.com`, `*.example.com`, and `test.example.com`. For example, if you add the `c.b.a.example.com` subdomain name, you must add the `_cdnauth.example.com` resolution record.

To add a resolution record of Tencent Cloud DNS, perform the following operations:

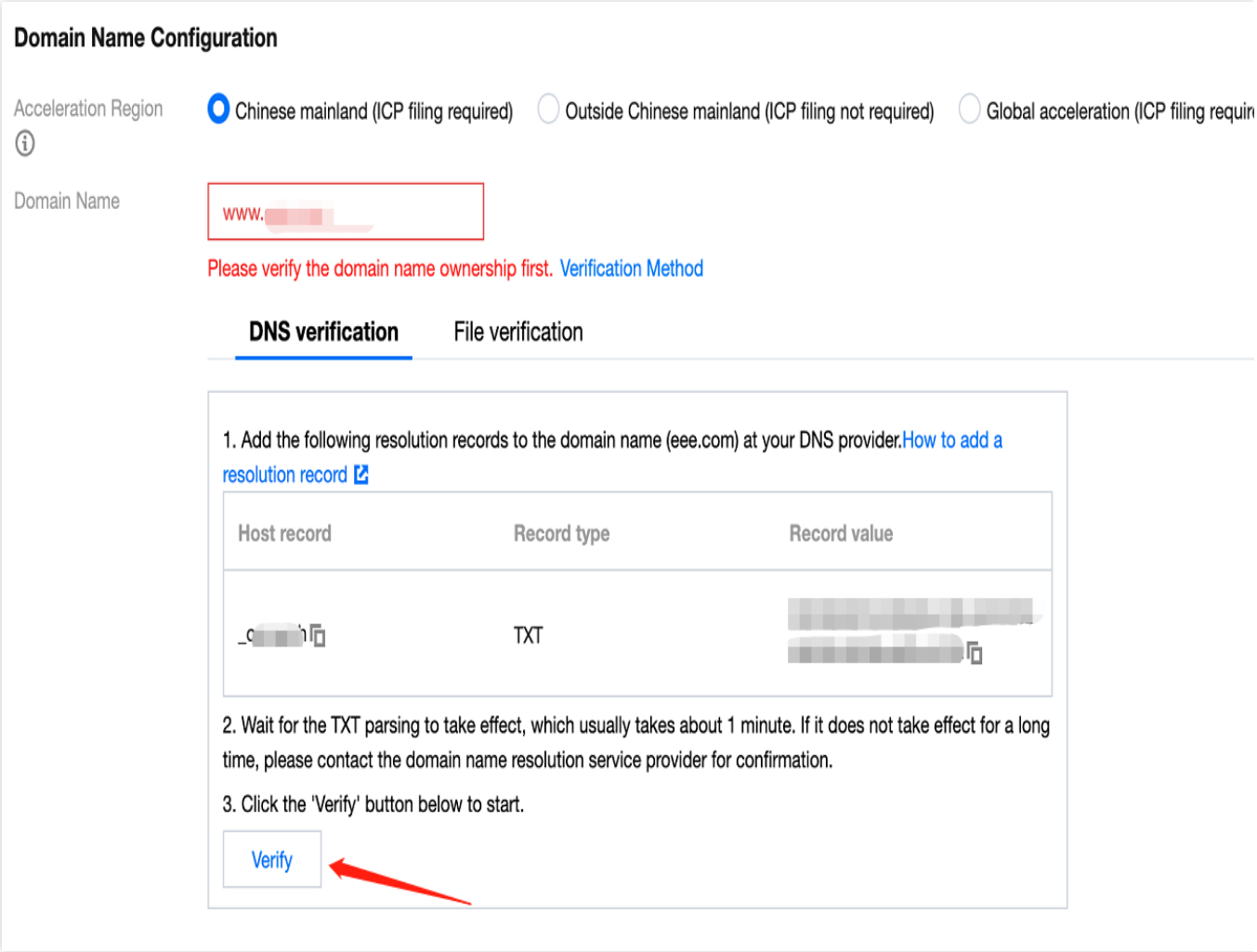
If your DNS provider is Tencent Cloud, log in to the DNSPod console, find the target domain name, click **DNS**, and add a TXT record. Set the **Host** parameter to `_cdnauth`, the **Record Type** parameter to TXT, and the **Record Value** parameter to the record value provided by Tencent Cloud CDN. Use the default settings for other parameters.

To add a resolution record of Alibaba Cloud DNS, perform the following operations:

If your DNS provider is Alibaba Cloud, log in to the DNS console of Alibaba Cloud, find the target domain name, and click **DNS Settings** in the **Actions** column. Set the **Record Type** parameter to `TXT`, configure the **Hostname** and **Record Value** parameters, and use the default settings for other parameters.

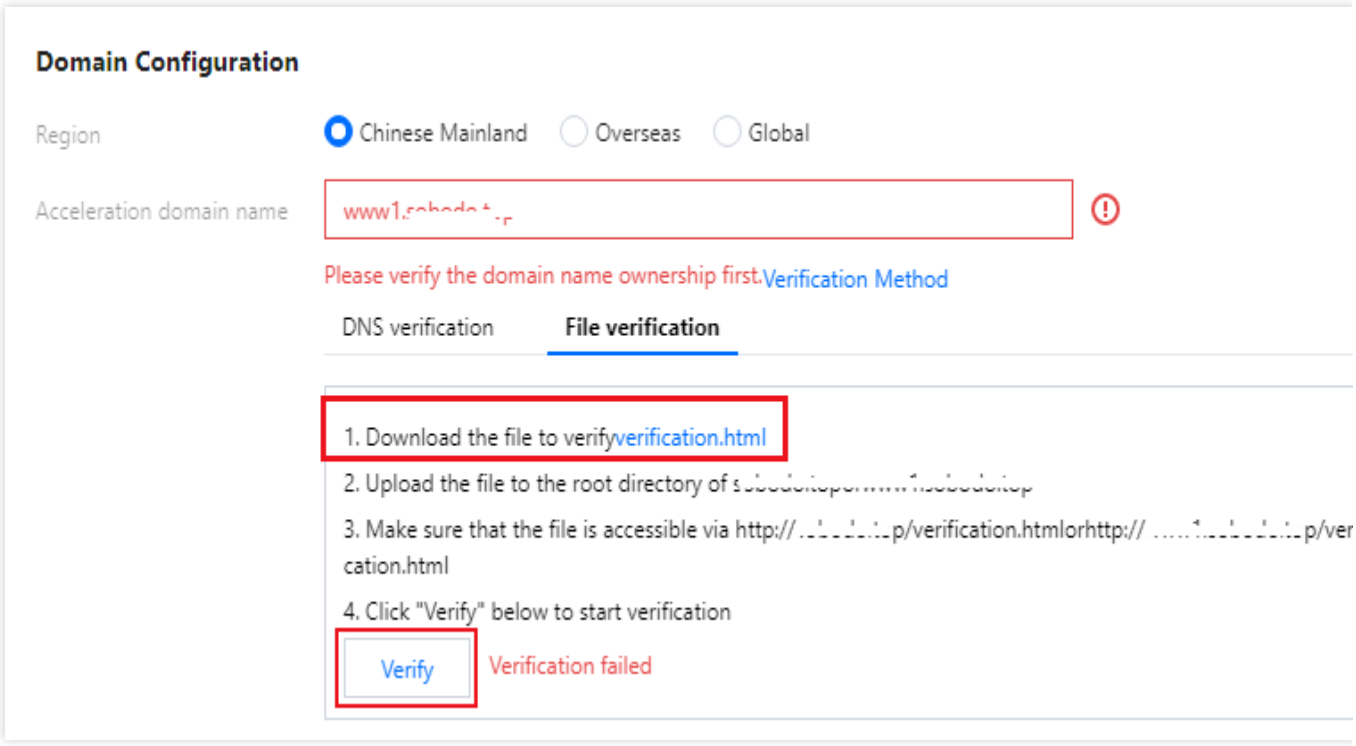
（3）Complete domain name attribution verification

Wait for the TXT record to take effect before you click the verification button to start verification. If the domain name fails to be verified, make sure that the TXT record is valid and has taken effect at the DNS provider. How do I know whether a TXT record takes effect?



3. File Verification

（1）Click the File verification tab.



（2）Click verification.html to download the file for verification.

Upload the file to the root directory on the server of your domain name, such as a Tencent Cloud Cloud Virtual Machine (CVM) instance, a Tencent Cloud Object Storage (COS) bucket, an Alibaba Cloud Elastic Compute Service (ECS) instance, or an Alibaba Cloud Object Storage Service (OSS) bucket. For example, if your domain name is `test.example.com`, you must upload the file to the `example.com` / or `test.example.com` / root directory.

Note：

You can perform verification by uploading the file to a subdomain name only if you use the file verification method.

（3）Complete domain name attribution verification

Make sure that the file is accessible via `http://example.com/verification.html` or `http://test.example.com/verification.html` before you click **Verify**. Your domain name will be successfully verified if the record you added is consistent with the content of the file. If the domain name cannot be verified, check whether the record and the content of the file are consistent.

Example:

In this example, the acceleration domain name is `a.test.com` and the origin server is a COS biucket.

1. Upload the verification.html file to the root directory of COS.

2. Add a CNAME record for the acceleration domain name at your DNS provider. Set the **Record value** parameter to the COS domain name.

3. Check whether the verification.html file is accessible via http(https)://Acceleration domain name/verification.html. Click **Verify**.

Method 2:API Operation Verification

1.Call the CreateVerifyRecord operation to generate a TXT resolution record for an acceleration domain name.

```
{
  "Response": {
    "DNSVerifyInfo": {
      "Record": "2023082515502104ad6d69c54862dcc99e226349af3440",
      "RecordType": "TXT",
      "SubDomain": "_cdnauth"
    },
    "FileVerifyInfo": {
      "FileVerifyDomains": [
        "123.com"
      ],
      "FileVerifyName": "verification.html",
      "FileVerifyUrl": "http://123.com/verification.html"
    },
    "RequestId": "10645a01-c728-4fb5-baa8-09d21e1090e3"
  }
}
```

2.Add the TXT resolution record at your DNS provider, such as DNSPod.

3.Call the VerifyDomainRecord operation to check whether the resolution record takes effect.

```
{
  "Response": {
    "RequestId": "48d4442e-cda6-4404-af2a-467cc5891079",
    "Result": true
  }
}
```

4.If the resolution record takes effect, call the AddCdnDomain operation to add the domain name.

FAQs

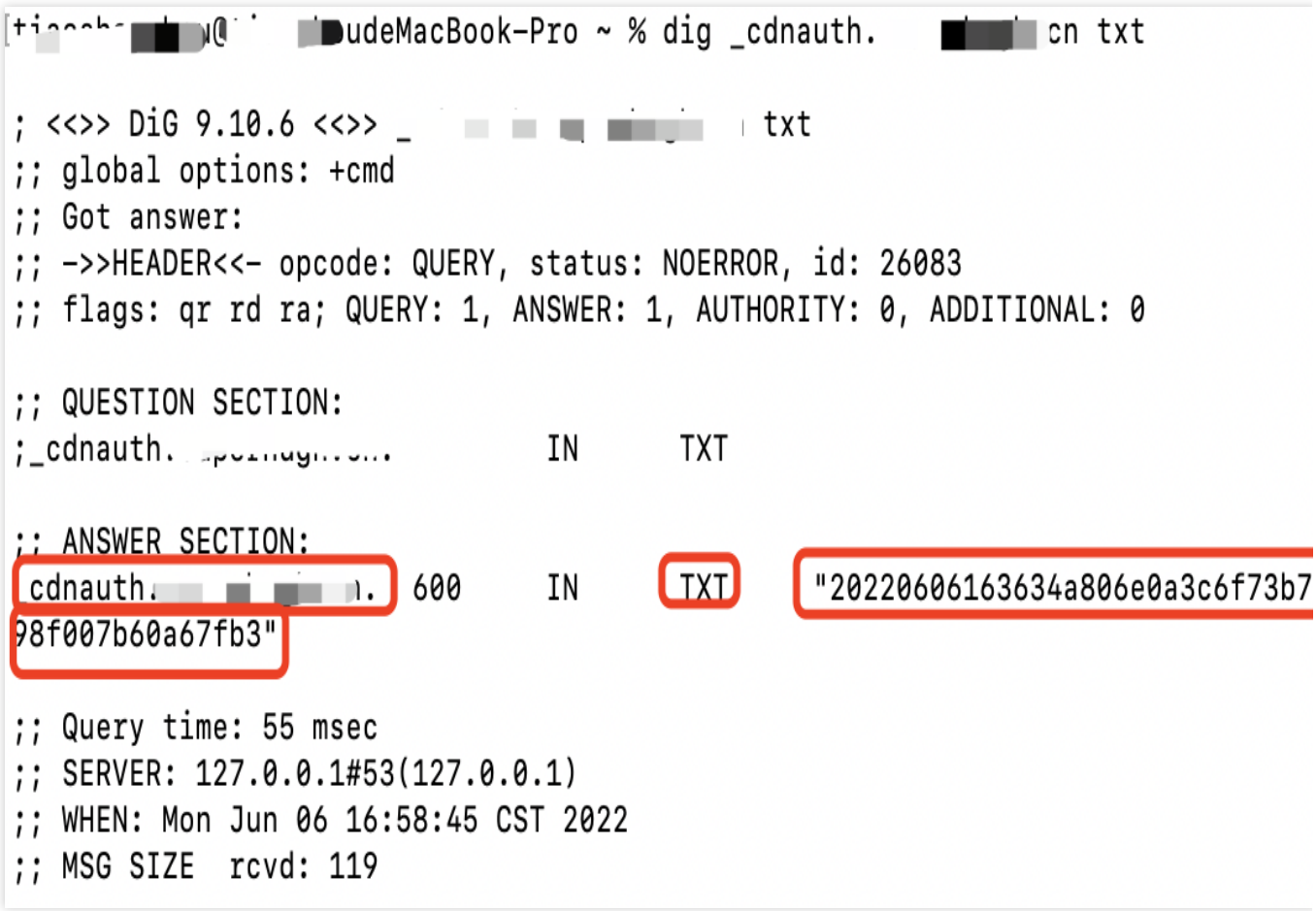
How do I know whether a TXT record takes effect?

Windows:

If the domain name that you connected is `test.example.com`, open the command prompt and run the `nslookup -qt=txt _cdnauth.example.com` command. Check whether the TXT record takes effect or is valid based on the output.

Linux or macOS:

If the domain name that you connected is `test.example.com`, open the command prompt and run the `dig _cdnauth.example.com` txt command. Check whether the TXT record takes effect or is valid based on the output.

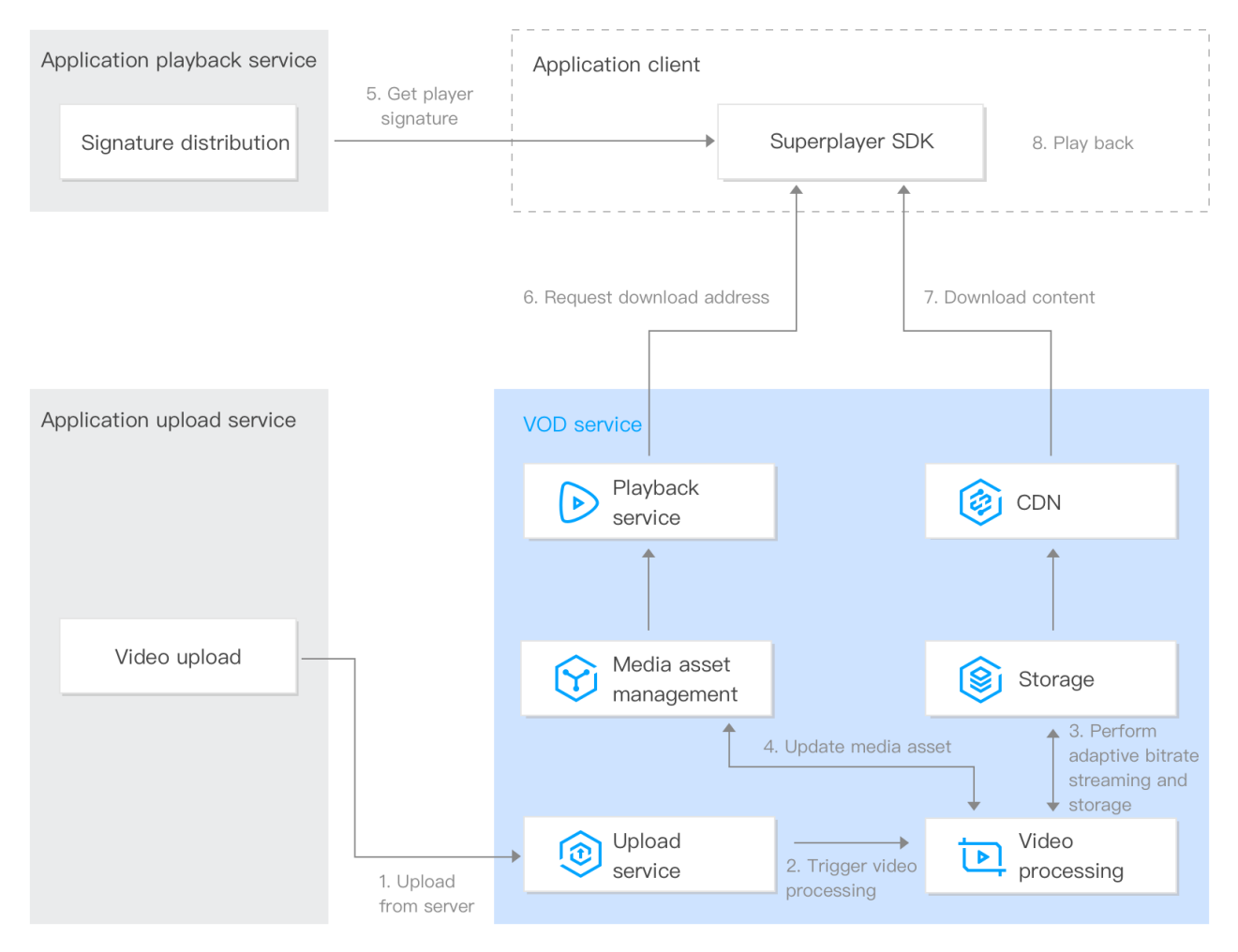


```
[t[redacted]@[redacted]udeMacBook-Pro ~ % dig _cdnauth.[redacted]cn txt

; <<>> DiG 9.10.6 <<>> _[redacted] txt
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26083
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;_cdnauth.[redacted]          IN      TXT

;; ANSWER SECTION:
cdnauth.[redacted]. 600     IN      TXT     "20220606163634a806e0a3c6f73b7
98f007b60a67fb3"

;; Query time: 55 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Mon Jun 06 16:58:45 CST 2022
;; MSG SIZE  rcvd: 119
```

Player Signature

Last updated：2025-02-19 17:53:22

A player signature is used to authorize a playback device to play videos. If your application playback service issues a valid signature to a device (step 6 in the figure below), the device will be able to play the video within the signature's validity period.



This document describes the parameters of a player signature and how to generate a signature.

Signature Parameters

Parameter	Required	Type	Description
appId	Yes	Integer	The VOD application AppID.
fileId	Yes	String	The VOD file ID.

contentInfo	Yes	Object	The content type of the specified file ID. For the structure of this parameter, see ContentInfo. Three types of content are supported: Adaptive bitrate audio/video, which may or may not be encrypted Transcoded audio/video Uploaded original audio/video
currentTimeStamp	Yes	Integer	The current time (Unix timestamp).
expireTimeStamp	No	Integer	The expiration time (Unix timestamp) of the distributed signature. If this parameter is left empty, the signature will never expire.
urlAccessInfo	No	Object	The access parameters of the playback URL, including key hotlink protection parameters, the playback domain, and the protocol. For the structure of this parameter, see UrlAccessInfo.
drmLicenseInfo	No	Object	The DRM configuration. For the structure of this parameter, see DrmLicenseInfo.

ContentInfo

Parameter	Required	Type	Description
audioVideoType	Yes	String	The type of audio/video played. Valid values: `RawAdaptive` : Unencrypted adaptive bitrate output `ProtectedAdaptive` : Private protocol- or DRM-encrypted adaptive bitrate output `Transcode` : Transcoding output `Original` : Uploaded original audio/video
rawAdaptiveDefinition	No	Integer	The ID of the unencrypted adaptive bitrate template allowed. This parameter is valid and required if `audioVideoType` is `RawAdaptive` .
drmAdaptiveInfo	No	Object	The ID of the encrypted adaptive bitrate template allowed. This parameter is valid and required if `audioVideoType` is `ProtectedAdaptive` . For its structure, see DRMAdaptiveInfo.
transcodeDefinition	No	Integer	The ID of the transcoding template allowed. This parameter is valid and required if `audioVideoType` is `Transcode` .

imageSpriteDefinition	No	Integer	The ID of the image sprite template, which is used to generate thumbnail previews.
resolutionNames	No	Array of Object	The names of different streams (different resolutions) displayed in the player. For its structure, see ResolutionNameInfo. If you do not specify this parameter or leave it empty, the following will be used: `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

DRMAdaptiveInfo

Parameter	Required	Type	Description
privateEncryptionDefinition	No	Integer	The ID of the adaptive bitrate template used when DrmType is `SimpleAES` .
widevineDefinition	No	Integer	The ID of the adaptive bitrate template used when DrmType is `Widevine` .
fairPlayDefinition	No	Integer	The ID of the adaptive bitrate template used when DrmType is `FairPlay` .

ResolutionNameInfo

Parameter	Required	Type	Description
MinEdgeLength	Yes	Integer	The video short side (px).
Name	Yes	String	The stream name.

UrlAccessInfo

Parameter	Required	Type	Description
t	No	String	The expiration time of the URL, which must be a hexadecimal string. For the valid values and other information, see the `t` parameter of hotlink protection.

			If this parameter is left empty, the URL will never expire.
exper	No	Integer	The preview duration in decimal seconds. The preview duration cannot be shorter than 30 seconds. For the valid values and other information, see the `exper` parameter of hotlink protection.
rlimit	No	Integer	The maximum number (decimal) of IP addresses allowed for playback. For the valid values and other information, see the `rlimit` parameter of hotlink protection.
us	No	String	The URL ID, which uniquely identifies a link. For the valid values and other information, see the `us` parameter of hotlink protection.
domain	No	String	The playback domain. If this is not specified or `Default` is passed in, the default distribution domain will be used.
scheme	No	String	The playback scheme. If this is not specified or `Default` is passed in, the default distribution configuration will be used. Other valid values: HTTP HTTPS

DrmLicenseInfo

Parameter	Required	Type	Description
persistent	No	String	Whether to allow persistent storage of DRM playback licenses by playback devices. Valid values: `ON` : Allow `OFF` : Do not allow The default value is `OFF` .
rentalDuration	No	Integer	The allowed storage time (seconds) of DRM playback licenses when `persistent` is `ON` . If this is not specified, there will be no limit on the storage time.
forceL1TrackTypes	No	Array of String	The track type that must use the L1 security level when Widevine is used. For other track

			types, Widevine L3 will be used. Valid values: `AUDIO` : Audio tracks `SD` : Video tracks whose short side is smaller than 720 px `HD` : Video tracks whose short side is equal to or larger than 720 px and smaller than 2160 px `UHD1` : Video tracks whose short side is equal to or larger than 2160 px and smaller than 4320 px `UHD2` : Video tracks whose short side is equal to or larger than 4320 px
minimumProtectionLevel	No	String	Whether to allow the terminal to use the basic mode for playback. Value options: BASIC: Allow playback using the basic mode STANDARD: Do not allow playback using the basic mode The default value is BASIC

Note:

If you use a subapplication, set `appId` to the ID of the subapplication.

The meanings and valid values of the signature parameters `t` , `exper` , `rlimit` , and `us` are the same as those of the hotlink protection.

Signature Calculation

The VOD player signature is a JSON Web Token (JWT), which consists of a header, a payload, and a key.

Header

The header is in JSON format and indicates the algorithm information used. Its content is fixed as follows:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Payload

The payload is in JSON format and includes the player signature parameters. Below is an example:

```
{
```

```
    "appId": 1255566655,
    "fileId": "4564972818519602447",
    "contentInfo": {
      "audioVideoType": "RawAdaptive",
      "rawAdaptiveDefinition": 10,
      "imageSpriteDefinition": 10
    },
    "currentTimeStamp": 1663064276,
    "expireTimeStamp": 1663294210,
    "urlAccessInfo": {
      "t": "6323e6b0",
      "rlimit": 3,
      "us": "72d4cd1101"
    }
  }
```

Key

The key is what's used to calculate the signature. In the example below, the default playback key is used.

Calculation formula

1. Calculate the signature:

`Signature = HMACSHA256(base64UrlEncode(Header) + "." + base64UrlEncode(Payload), Key)`

2. Calculate the token:

`Token = base64UrlEncode(Header) + '.' + base64UrlEncode(Payload) + '.' + base64UrlEncode(Signature)`

The token generated is the VOD player signature.

Note:

For more information about the HMACSHA256 algorithm, see RFC 4868 - Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec. For more information about `base64UrlEncode`, see Base 64 Encoding with URL and Filename Safe Alphabet.

VOD offers a signature generation tool and a signature verification tool:

Player signature tools

Calculation example

Suppose your `appId` is `1255566655` and you want to generate a player signature for a video whose file ID is `4564972818519602447`. The other parameters are as follows:

The playback key is `TxtyhLlgo7J3iOADIron`.

The distribution time of the player signature is 18:17:56 on September 13, 2022, which converted to Unix timestamp is `1663064276`.

The expiration time of the player signature is 10:10:10 on September 16, 2022, which converted to Unix timestamp is `1663294210`.

The expiration time of hotlink protection is 11:00:00 on September 16, 2022, which converted to Unix timestamp is `6323e6b0`.

Up to three IP addresses are allowed to play the video using the playback URL.

The random string generated for the URL ID is `72d4cd1101`.

Calculate the signature as follows:

1. Determine the content of the header:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

The result generated after `base64UrlEncode` is:

`eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9`.

2. Determine the content of the payload:

```
{
  "appId": 1255566655,
  "fileId": "4564972818519602447",
  "contentInfo": {
    "audioVideoType": "RawAdaptive",
    "rawAdaptiveDefinition": 10,
    "imageSpriteDefinition": 10
  },
  "currentTimeStamp": 1663064276,
  "expireTimeStamp": 1663294210,
  "urlAccessInfo": {
    "t": "6323e6b0",
    "rlimit": 3,
    "us": "72d4cd1101"
  }
}
```

The result generated after `base64UrlEncode` is:

`eyJhcHBJZCI6MTI1NTU2NjY1NSwiZmlsZUlkIjoiNDU2NDk3MjgxODUxOTYwMjQ0NyIsImNvbnRlbnRJbmZvMSI6eyJhdWRpb1ZpZGVvVHlwZSI6IlJhd0FkYXB0`

`aXZlIiwicmF3QWRhcHRpdmVEZWZpbml0aW9uIjoxMCwiaW1hZ2VTcHJpdGVEZWZpbml0aW9uIjoxMH0sImN1cnJlbnRUaW1lU3RhbXAiOjE2NjMwNjQyNzYsImV4`

`cGlyZVRpbWVTdGFtcCI6MTY2MzI5NDIxMCwidXJsQWNjZXNzSW5mbyI6eyJ0IjoiNjMyM2U2YjAiLCJybGltaXQiOjMsInVzIjoiNzJkNGNkMTEwMSJ9fQ`。

3. Use the playback key (`TxtyhLlgo7J3iOADIron`) to generate an HMAC signature: `QFcBX9830ysTzJIyZxoOlRmNb2Gqy2fns9yOfriaDI8` .
4. The token generated is:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcHBJZCI6MTI1NTU2NjY1NSwiZmlsZUlkIjoiNDU2NDk3MjgxODUxOTYwMjQ0NyIsImNvbnRlbnRJbmZvMSI6eyJhdWRpb1ZpZGVvVHlwZSI6IlJhd0FkYXB0aXZlIiwicmF3QWRhcHRpdmVEZWZpbml0aW9uIjoxMCwiaW1hZ2VTcHJpdGVEZWZpbml0aW9uIjoxMH0sImN1cnJlbnRUaW1lU3RhbXAiOjE2NjMwNjQyNzYsImV4cGlyZVRpbWVTdGFtcCI6MTY2MzI5NDIxMCwidXJsQWNjZXNzSW5mbyI6eyJ0IjoiNjMyM2U2YjAiLCJybGltaXQiOjMsInVzIjoiNzJkNGNkMTEwMSJ9fQ.QFcBX9830ysTzJIyZxoOlRmNb2Gqy2fns9yOfriaDI8 。
```

Sample Code

VOD provides Python, Java, Go, C#, PHP, and Node.js sample code for calculating player signatures. For details, see Player Signature Sample Codes.

Common Errors

If you use the player signature and the player SDK returns an error code, common causes include:
Incorrect signature calculation key**. The** playback key in the default distribution configuration rather than the `KEY` parameter in the key hotlink protection configuration should be used.
Incorrect signature parameters:
Incorrect parameter type. For example, `appId` must be an integer, but the value entered is `appId:"125000123"` (string); the transcoding template parameter in `contentInfo` must be an integer, but the value entered is `transcodeDefinition: "14011"` (string).
Incorrect parameter value. For example, the `audioVideoType` parameter in `contentInfo` is set to `Transocde` (typo).

Player Signature Sample Codes

Last updated：2023-07-14 16:25:37

Python Sample Code for Signature Calculation

Use the pyjwt library to calculate the signature. You can install it by running the `pip install pyjwt` command.

```
#!/usr/bin/python
#coding=utf-8

import jwt

AppId = 1255566655
FileId = "4564972818519602447"
AudioVideoType = "RawAdaptive"
RawAdaptiveDefinition = 10
ImageSpriteDefinition = 10
CurrentTime = 1546340400
PsignExpire = 1546344000
UrlTimeExpire = "5c2b5640"
PlayKey = "TxtyhLlgo7J3iOADIron"

Original = {
    "appId": AppId,
    "fileId": FileId,
    "contentInfo": {
        "audioVideoType": AudioVideoType,
        "rawAdaptiveDefinition": RawAdaptiveDefinition,
        "imageSpriteDefinition": ImageSpriteDefinition
    },
    "currentTimeStamp": CurrentTime,
    "expireTimeStamp": PsignExpire,
    "urlAccessInfo": {
        "t": UrlTimeExpire
    }
}

Signature = jwt.encode(Original, PlayKey, algorithm='HS256')

print("Original: ", Original)
```

```
print("Signature: ", Signature)
```

Java Sample Code for Signature Calculation

Use the java-jwt library to calculate the signature.

```
import java.util.*;
import com.auth0.jwt.algorithms.Algorithm;
import com.auth0.jwt.exceptions.JWTCreationException;
import com.auth0.jwt.JWT;

class Main {
    public static void main(String[] args) {
        Integer AppId = 1255566655;
        String FileId = "4564972818519602447";
        String AudioVideoType = "RawAdaptive";
        Integer RawAdaptiveDefinition = 10;
        Integer ImageSpriteDefinition = 10;
        Integer CurrentTime = 1589448067;
        Integer PsignExpire = 1589548067;
        String UrlTimeExpire = "5ebe9423";
        String PlayKey = "TxtyhLlgo7J3iOADIron";
        HashMap<String, Object> urlAccessInfo = new HashMap<String, Object>();
        urlAccessInfo.put("t", UrlTimeExpire);
        HashMap<String, Object> contentInfo = new HashMap<String, Object>();
        contentInfo.put("audioVideoType", AudioVideoType);
        contentInfo.put("rawAdaptiveDefinition", RawAdaptiveDefinition);
        contentInfo.put("imageSpriteDefinition", ImageSpriteDefinition);

        try {
            Algorithm algorithm = Algorithm.HMAC256(PlayKey);
            String token = JWT.create().withClaim("appId", AppId).withClaim("fileId
                    .withClaim("contentInfo", contentInfo)
                    .withClaim("currentTimeStamp", CurrentTime).withClaim("expireTi
                    .withClaim("urlAccessInfo", urlAccessInfo).sign(algorithm);
            System.out.println("token:" + token);
        } catch (JWTCreationException exception) {
            // Invalid Signing configuration / Couldn't convert Claims.
        }
    }
}
```

Go Sample Code for Signature Calculation

Use the jwt-go library to calculate the signature. You can install it by running the `go get github.com/dgrijalva/jwt-go` command.

```
package main

import (
        "fmt"
        "time"
        "strconv"
        "github.com/dgrijalva/jwt-go"
)

func main(){
        appId := 1255566655 // Your `appid`
        fileId := "4564972818519602447" // The target `FileId`
        audioVideoType := "RawAdaptive" // The type of audio/video played
        rawAdaptiveDefinition := 10 // The ID of the unencrypted adaptive bitrate t
        imageSpriteDefinition := 10 // The ID of the image sprite template, which i
        currentTime := time.Now().Unix()
        psignExpire := currentTime + 3600 // The signature expiration time, which i
        urlTimeExpire := strconv.FormatInt(psignExpire, 16) // The URL expiration t
        playKey := []byte("TxtyhLlgo7J3iOADIron")

        // Create a new token object, specifying signing method and the claims
        // you would like it to contain.
        token := jwt.NewWithClaims(jwt.SigningMethodHS256, jwt.MapClaims{
                "appId":            appId,
                "fileId":           fileId,
                "contentInfo": {
                        "audioVideoType": audioVideoType,
                        "rawAdaptiveDefinition": rawAdaptiveDefinition,
                        "imageSpriteDefinition": imageSpriteDefinition,
                },
                "currentTimeStamp": currentTime,
                "expireTimeStamp":  psignExpire,
                "urlAccessInfo": map[string]string{
                        "t": urlTimeExpire,
                },
```

```
        })

        // Sign and get the complete encoded token as a string using the secret
        tokenString, err := token.SignedString(playKey)

        fmt.Println(tokenString, err)
}
```

C# Sample Code for Signature Calculation

Use the jose-jwt library to calculate the signature. You can install it by running the `Install-Package jose-jwt` command of NuGet.

```
using System;
using System.Text;
using System.Collections.Generic;
using Jose;

public class Program
{
        public static void Main()
        {
                var appId = 1255566655; // Your `appid`
                var fileId = "4564972818519602447"; // The target `FileId`
                var audioVideoType = "RawAdaptive"; // The type of audio/video play
                var rawAdaptiveDefinition = 10; // The ID of the unencrypted adapti
                var imageSpriteDefinition = 10; // The ID of the image sprite templ
                var currentTime = DateTimeOffset.UtcNow.ToUnixTimeSeconds();
                var psignExpire = currentTime + 3600; // The signature expiration t
                var urlTimeExpire = psignExpire.ToString("X4"); // The URL expirati
                var playKey = "TxtyhLlgo7J3iOADIron";
                var playKeyBytes = Encoding.ASCII.GetBytes(playKey);
                var payload = new Dictionary<string, object>()
                {
                        {"appId", appId},
                        {"fileId", fileId},
                        {"contentInfo", new Dictionary<string, object>()
                                {
                                        {"audioVideoType", audioVideoType},
                                        {"rawAdaptiveDefinition", rawAdaptiveDefini
                                        {"imageSpriteDefinition", imageSpriteDefini
                                }
```

```
                        },
                        {"currentTimeStamp", currentTime},
                        {"expireTimeStamp", psignExpire},
                        {"urlAccessInfo", new Dictionary<string, object>()
                                {
                                        {"t", urlTimeExpire}
                                }
                        }
                };
                string token = Jose.JWT.Encode(payload, playKeyBytes, JwsAlgorithm.
                Console.WriteLine(token);
        }
}
```

PHP Sample Code for Signature Calculation

Use the php-jwt library to calculate the signature. You can install it by running the `composer require firebase/php-jwt` command.

```
<?php
require 'vendor/autoload.php';
use \\Firebase\\JWT\\JWT;

$appId = 1255566655; // Your `appid`
$fileId = "4564972818519602447"; // The target `FileId`
$audioVideoType = "RawAdaptive"; // The type of audio/video played
$rawAdaptiveDefinition = 10; // The ID of the unencrypted adaptive bitrate template
$imageSpriteDefinition = 10; // The ID of the image sprite template, which is used
$currentTime = time();
$psignExpire = $currentTime + 3600; // The signature expiration time, which is set
$urlTimeExpire = dechex($psignExpire); // The URL expiration time, which is a hexad
$playKey = "TxtyhLlgo7J3iOADIron";

$payload = array(
    "appId" => $appId,
    "fileId" => $fileId,
    "contentInfo" => array(
        "audioVideoType"=> $audioVideoType,
                "rawAdaptiveDefinition"=> $rawAdaptiveDefinition,
                "imageSpriteDefinition"=> $imageSpriteDefinition
    ),
    "currentTimeStamp" => $currentTime,
```

```
    "expireTimeStamp" => $psignExpire,
    "urlAccessInfo" => array(
        "t" => $urlTimeExpire
    )
);

$jwt = JWT::encode($payload, $playKey, 'HS256');
print_r($jwt);
?>
```

Node.js Sample Code for Signature Calculation

Use the jsonwebtoken library to calculate the signature. You can install it by running the `npm install jsonwebtoken` command.

```
var jwt = require('jsonwebtoken');

var appId = 1255566655 // Your `appid`
var fileId = "4564972818519602447" // The target `FileId`
var audioVideoType = "RawAdaptive" // The type of audio/video played
var rawAdaptiveDefinition = 10 // The ID of the unencrypted adaptive bitrate templa
var imageSpriteDefinition = 10 // The ID of the image sprite template, which is use
var currentTime = Math.floor(Date.now()/1000)
var psignExpire = currentTime + 3600 // The signature expiration time, which is set
var urlTimeExpire = psignExpire.toString(16) // The URL expiration time, which is a
var playKey = 'TxtyhLlgo7J3iOADIron'

var payload = {
        appId: appId,
        fileId: fileId,
        contentInfo: {
                audioVideoType: audioVideoType,
                rawAdaptiveDefinition: rawAdaptiveDefinition,
                imageSpriteDefinition: imageSpriteDefinition
        },
        currentTimeStamp: currentTime,
        expireTimeStamp: psignExpire,
        urlAccessInfo: {
                t: urlTimeExpire
        }
}
var token = jwt.sign(payload, playKey);
console.log(token);
```

Hotlink Protection Settings Overview

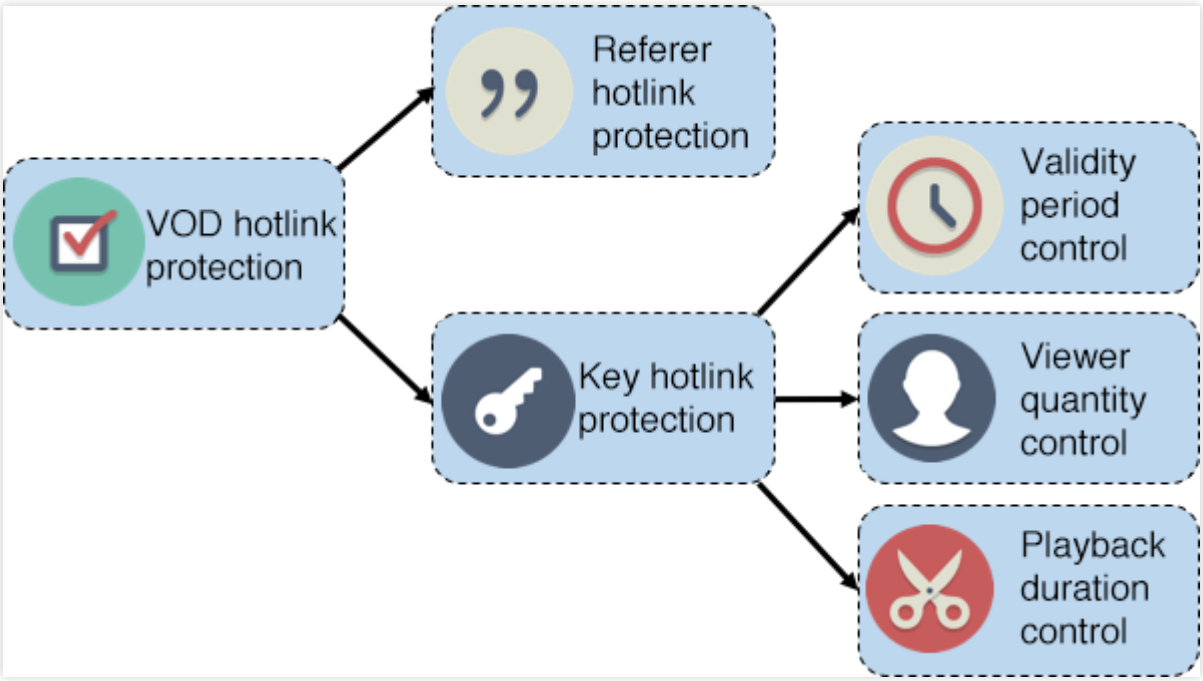
Last updated：2021-03-17 10:36:54

Overview

Tencent Cloud VOD offers hotlink protection to control the video playback permissions. After hotlink protection is enabled, a Tencent Cloud CDN node will check key information in the playback requests and return the video data only to approved requests. This scheme has no special requirements for players, that is, it is applicable to both the player SDK of VOD and common players.

Types and Capabilities

VOD hotlink protection can prevent hotlinking based on referer and key.



Referer hotlink protection

The referer mechanism based on HTTP identifies the request source through the referer field in the playback request header. You can add specified domain names to a blocklist or allowlist, based on which the CDN node will authenticate to allow or deny the playback requests accordingly.

Key hotlink protection

It allows you to splice a video's playback control parameters into the video URL in the form of `QueryString`. The CDN node will check the playback control parameters in the URL and control video playback accordingly. At present, key hotlink protection supports controlling "validity period", "viewer quantity", and "video playback duration" through corresponding parameters, i.e., "expiration time", "number of IPs allowed for playback", and "preview duration".

Validity period control

It specifies the expiration time of a video URL. If the requested video URL has expired, the video cannot be played back. In this way, you can set a validity period for the video URL to prevent malicious users from transferring the URL to other websites for long-term use.

Viewer quantity control

It specifies the number of viewers that can access the video URL. Devices that are not in the same private network generally have different public IPs. You can specify how many viewers are allowed to access a URL by limiting the number of IPs allowed for playback on the URL. This helps prevent malicious users from transferring the URL to other websites for unrestricted distribution.

Video playback duration control

It specifies the preview duration in a video URL (e.g., the first five minutes of a video) to implement preview for non-paying users.

Note:

For more information on referer hotlink protection, please see Referer Hotlink Protection.

For more information on key hotlink protection, please see Key Hotlink Protection.

Referer Hotlink Protection

Last updated：2021-03-17 10:39:26

Feature Overview

Based on the referer mechanism supported by the HTTP protocol, the source of a request can be identified through the referer field in the HTTP header. You can configure a referer blocklist or allowlist to identify and authenticate the sources of video requests.

Blocklist and allowlist modes are supported. When a video playback request reaches a CDN node, the node will authenticate the request source according to the configured referrer blocklist or allowlist. If a request meets the rule, CDN will return video data; otherwise, it will return a 403 response code and reject the playback request.

Note:

For more information on enabling referer hotlink protection, please see Setting Hotlink Protection.

Precautions

This feature is optional and not enabled by default.

After enabling this feature, select and configure the blocklist or allowlist. The two modes are mutually exclusive, and only one is supported at the same time.

1–10 domain names can be entered in the blocklist or allowlist (one entry per line).

Do not put a protocol name (`http://` or `https://`) before a domain name. Domain name matching is based on the prefix (for example, if you enter `abc.com` , then both `abc.com/123` and `abc.com.cn` will be matched), and wildcard (e.g., `*.abc.com)` is supported.

Key Hotlink Protection

Last updated：2024-08-26 16:43:28

Overview

You can specify the expiration time in a playback URL to prevent malicious users from transferring the URL to other websites for long-term unauthorized viewing.
You can specify the maximum number of IP addresses allowed to access a playback URL to prevent malicious users from distributing the video to a large number of viewers.
You can specify the preview duration in a playback URL to allow viewers to watch only the start of your video.
You can add a region allowlist/blocklist to a playback URL.
You can add a referer allowlist/blocklist to a playback URL.
You can use a key (`KEY`) to generate a signature and add it to a playback URL. As long as the key is not disclosed, the URL cannot be forged.
A CDN node determines whether to allow a playback request by verifying the parameters and signature in the playback URL. If a request fails to pass the verification, a 403 response code will be returned.
Supported file formats include MP4, TS, M3U8, FLV, AAC, MOV, WMV, AVI, MP3, RMVB, MKV, MPG, 3GP, WEBM, M4V, ASF, F4V, WAV, MPEG, VOB, RM, WMA, DAT, M4A, MPD, and M4S.
Note:
For detailed directions on configuring key hotlink protection, see Setting Hotlink Protection.
Currently, the preview feature is not supported for audio files.

Generating a Hotlink Protection URL

All your videos in VOD have an **original video URL**. If hotlink protection is not enabled, the original video URL can be used to play back the video.
After key hotlink protection is enabled, your videos will no longer be playable via the original URLs. You need to generate **hotlink protection URLs** for them.
You can generate a hotlink protection URL for a video by adding hotlink protection parameters at the end of the original URL in the form of a query string. Below is an example:

```
http://example.vod2.myqcloud.com/dir1/dir2/myVideo.mp4?t=[t]&exper=[exper]&rlimit=[
```

Below are descriptions and values of the parameters in a hotlink protection URL.

Hotlink protection parameters

Parameter	Required	Description
KEY	Yes	The key used when key hotlink protection is enabled. It can contain 8–20 letters (a–Z) or digits (0–9). We recommend you generate this key in the console. For detailed directions, see Setting Hotlink Protection.
Dir	Yes	The part of the original video URL with the filename removed. For example, if the original URL is `http://example.vod2.myqcloud.com/dir1/dir2/myVideo.mp4`, then the playback path is `/dir1/dir2/`.
t	Yes	The expiration time of the playback URL, which is a Unix hexadecimal timestamp in lowercase. Once expired, the URL will become invalid, and a 403 error will be returned. Given the potential time differences between devices, we recommend you set the expiration time five minutes (300 seconds) later than the actual time you want the URL to expire. The validity period of the URL should be longer than the video length so that the full video can be played.
exper	No	The preview duration in decimal seconds. If this parameter is left empty or set to 0, preview is disabled (the full video will be returned). The preview duration must be shorter than the original video duration; otherwise, playback may fail.
rlimit	No	The maximum number (decimal) of IP addresses allowed to play the video. The maximum value is 9. If this parameter is left empty, there is no restriction. Suppose you want your video to be played by only one user. We do not recommend setting `rlimit` to 1 (instead, set it to 3, for example). This is because the IP of a mobile device may change after a reconnection.
us	No	The URL ID, which randomizes a hotlink protection URL and improves its uniqueness. We recommend you use a random `us` value every time you generate a hotlink protection URL.
whreg	No	A list of regions allowed to play the video. You can specify 1-10 three-letter region codes. Separate the codes with commas.
bkreg	No	A list of regions banned from playing the video. You can specify 1-10 three-letter region codes. Separate the codes with commas.
whref	No	A list of domains allowed to play the video. You can specify 1-10 domains. Leave out the `http://` and `https://` prefix, and separate the domains with commas. If you pass in `abc.com`, it will cover lower-level domains such as `abc.com/123` and `abc.com.cn`. Wildcards are supported. For example, you can pass in `*.abc.com`.

bkref	No	A list of domains banned from playing the video. You can specify 1-10 domains. Leave out the `http://` and `https://` prefix, and separate the domains with commas. If you pass in `abc.com`, it will cover lower-level domains such as `abc.com/123` and `abc.com.cn`. Wildcards are supported. For example, you can pass in `*.abc.com`.
sign	Yes	The hotlink protection signature, which verifies the validity of a hotlink protection URL. It must be a 32-character hexadecimal number. A 403 error will be returned if a URL fails to pass the signature verification. For how to generate the signature, see below.

Signature calculation formula

```
sign = md5(KEY + Dir + t + exper + rlimit + us + whref + bkref + whreg + bkreg)
```

`+` in the formula is used to concatenate two strings. Optional parameters can be empty strings.

Examples of Hotlink Protection URL Generation

Assume that you have a video in VOD and its original playback URL is `http://example.vod2.myqcloud.com/dir1/dir2/myVideo.mp4`. You have enabled key hotlink protection. The generated key is `24FEQmTzro4V5u3D5epW`, and the generated random string is `72d4cd1101`, and now you want to:

1. Generate a hotlink protection URL for this video and set the expiration time of the URL to 20:00 on January 31, 2018 (1517400000 in Unix time).
2. Generate a preview URL and set the preview duration to the first five minutes of the video (the original video duration is longer than five minutes).
3. Set the maximum number of IP addresses allowed to play the video to three.

The following describes how to generate hotlink protection URLs for three different example scenarios.

Example 1. Limiting the expiration time

Step 1. Determine the hotlink protection parameters

Parameter	Value	Description
KEY	24FEQmTzro4V5u3D5epW	The key you set when enabling key hotlink protection.
Dir	/dir1/dir2/	The part of the original video URL with the filename `myVideo.mp4` removed.

t	5a71afc0	The expiration timestamp (1517400000) converted to hexadecimal.
us	72d4cd1101	The generated random string.

Step 2. Calculate the signature

```
sign = md5("24FEQmTzro4V5u3D5epW/dir1/dir2/5a71afc072d4cd1101") =
"3d8488faeb37d52d6bf63b63c1b171c3"
```

Step 3. Generate a hotlink protection URL

Add the hotlink protection parameters to the original URL in the form of a query string:

```
http://example.vod2.myqcloud.com/dir1/dir2/myVideo.mp4?t=5a71afc0&us=72d4cd1101&sig
```

Example 2. Limiting the number of IP addresses

Step 1. Determine the hotlink protection parameters

Parameter	Value	Description
`KEY`	`24FEQmTzro4V5u3D5epW`	The key you set when enabling key hotlink protection.
`Dir`	`/dir1/dir2/`	The part of the original video URL with the filename `myVideo.mp4` removed.
`t`	`5a71afc0`	The expiration timestamp (1517400000) converted to hexadecimal.
`rlimit`	`3`	Allow up to three IP addresses to play the video.
`us`	`72d4cd1101`	The generated random string.

Step 2. Calculate the signature

```
sign = md5("24FEQmTzro4V5u3D5epW/dir1/dir2/5a71afc0372d4cd1101") =
"c5214f0d5961b13acd558b4957c4dfc5"
```

Step 3. Generate a hotlink protection URL

Add the hotlink protection parameters to the original URL in the form of a query string:

```
http://example.vod2.myqcloud.com/dir1/dir2/myVideo.mp4?
t=5a71afc0&rlimit=3&us=72d4cd1101&sign=c5214f0d5961b13acd558b4957c4dfc5
```

Example 3. Limiting the playback duration

Step 1. Determine the hotlink protection parameters

Parameter	Value	Description
KEY	24FEQmTzro4V5u3D5epW	The key you set when enabling key hotlink protection.
Dir	/dir1/dir2/	The part of the original video URL with the filename `myVideo.mp4` removed.
t	5a71afc0	The expiration timestamp (1517400000) converted to hexadecimal.
exper	300	Set the preview duration to five minutes (300 seconds).
us	72d4cd1101	The generated random string.

Step 2. Calculate the signature

```
sign = md5("24FEQmTzro4V5u3D5epW/dir1/dir2/5a71afc030072d4cd1101") =
"547d98c4b91e81b5ea55c95cef63223f"
```

Step 3. Generate a hotlink protection URL

Add the hotlink protection parameters to the original URL in the form of a query string:

```
http://example.vod2.myqcloud.com/dir1/dir2/myVideo.mp4?
t=5a71afc0&exper=300&us=72d4cd1101&sign=547d98c4b91e81b5ea55c95cef63223f
```

Key Hotlink Protection URL Generator and Checker

VOD provides a key hotlink protection URL generator and checker for you to quickly and accurately generate and check hotlink protection URLs.

Key hotlink protection URL generator

Key hotlink protection URL checker

Notes

This feature is optional and disabled by default.

After key hotlink protection is enabled, the original video URL can no longer be used for playback. You need to generate a hotlink protection URL according to the rules specified above.

The key (`KEY`) must contain 8–20 letters or digits.

If a hotlink protection URL expires or its signature fails to pass the verification, the video cannot be played and a 403 response code will be returned.

Make sure the parameters in the query string of a hotlink protection URL are in the order of `t` , `exper` , `rlimit` , `us` , and `sign` ; otherwise, playback will fail.

If you use the preview feature, make sure that the preview duration is shorter than the video duration; otherwise, playback will fail.

The preview feature has strict restrictions on video formats (for example, only H.264 is supported, and the metadata must be included in the header of the video file). Preview will fail if a video does not meet the requirements. We recommend you transcode your video in VOD before enabling preview (all transcoding files generated by VOD meet the preview requirements).

Media Encryption and Copyright Protection Overview

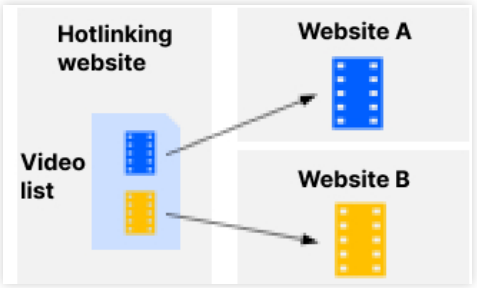
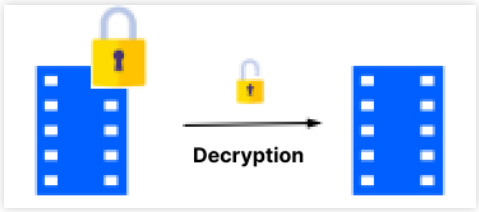
Last updated：2025-02-19 17:53:22

Online education platforms and OTT services provide their users with quality content. Viewers either buy courses or videos they are interested in or become subscribers to access the full content.
With digital content piracy on the rise, copyright protection has become a major challenge for the video on demand industry.

Main Forms of Content Piracy

Hotlinking, **decryption**, and **unauthorized recording** are three common forms of piracy.

Piracy	Hotlinking	Decryption
Note	Your content is linked by other websites.	Your encrypted content is decrypted and distribu without your consent.

Copyright Protection Capabilities

Thanks to VOD's rich experience in copyright protection, we are able to provide a full range of solutions to help you fight piracy.

Category	Capability	VOD Solution
Content protection	Encrypt your content	HLS private encryption, commercial-grade DRM
	Prevent network sniffing and decryption	HLS private encryption, commercial-grade DRM
	Prevent decryption by browser extensions	HLS private encryption, commercial-grade DRM
	Prevent screencapturing by **browser extensions**. When screenshot or screen recording behavior is detected, the player turns black, and the recorded/screenshot content is all black.	Commercial-grade DRM
	Prevent screencapturing by **system software**. When screenshot or screen recording behavior is detected, the player turns black, and the recorded/screenshot content is all black.	Commercial-grade DRM
Playback protection	Limit the countries or regions allowed to play your content	Key hotlink protection
	Limit the domains allowed to play your content	Refer hotlink protection

	Limit the validity period of a playback URL	Key hotlink protection
	Limit the number of IP addressed allowed to play your content	Key hotlink protection
	Limit the preview length	Key hotlink protection
Unauthorized distributor tracking	Identify unauthorized distributors	Floating watermarks, digital watermarks

Protection Levels and Compatibility

VOD offers two encryption schemes: HLS private encryption and commercial-grade DRM.

HLS private encryption

This is VOD's proprietary encryption scheme. The standard mode is used by default. If a device does not support the standard mode(such as WeChat Mini Programs, iOS browsers, and a few other browsers), the player will automatically switch to the basic mode.

Note:

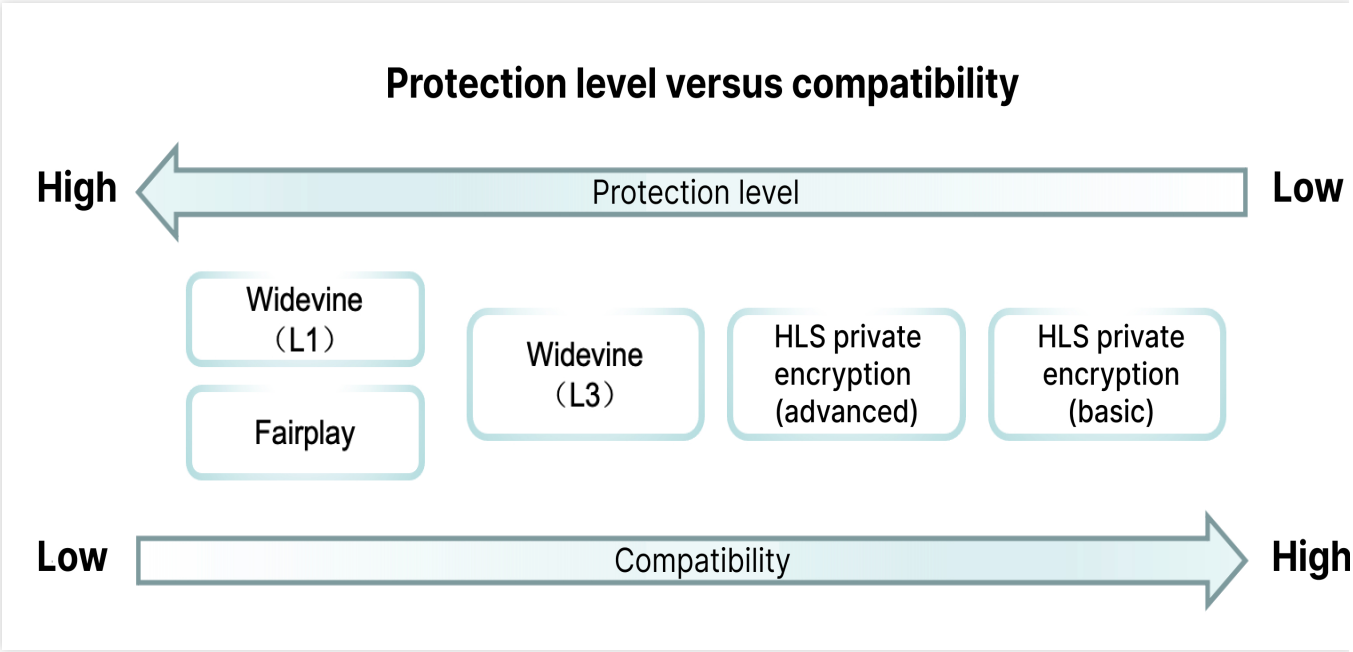
When using the basic mode for playback, the anti-piracy and screen recording prevention capabilities are weaker. If you need to disable playback in the basic mode, you can achieve this through the DrmLicenseInfo type in the Player Signature. Note that disabling the basic mode will result in playback failures on incompatible playback clients.

Commercial-grade DRM

Currently, VOD supports FairPlay DRM (Apple) and Widevine DRM (Google). For Widevine, the L1 and L3 protection levels are supported. L1 requires a hardware-based trusted execution environment (TEE) and is more demanding on the playback device. If a device does not support L1, L3 will be used automatically.

Encryption Method	Protection Level	Compatibility
HLS private encryption (standard)	**High** Content is encrypted. Strong protection against sniffing and decryption. Strong protection against decryption and screencapturing by browser extensions. Relatively weak protection against system and third-party screencapturing software	**High** Supports playback on mobile devices. Supports most PC browsers. Supports most Android browsers; does not support iOS browsers.
HLS private	**Medium**	**Very high**

encryption (basic)	Content is encrypted. Relatively weak protection against decryption and screencapturing tools.	Supports almost all platforms and devices.
FairPlay	**Very high** Hardware-based decryption is required to play the encrypted content. Very strong protection against decryption tools and extensions. Very strong protection against system and third-party screencapturing software and screencapturing browser extensions.	**Relatively high** Supports playback on iOS applications. Supports playback on iOS and macOS Safari.
Widevine (L1)	**Very high** Hardware-based decryption is required to play the encrypted content. Very strong protection against decryption tools and extensions. Very strong protection against system and third-party screencapturing software and screencapturing browser extensions.	**Relatively low** Does not support playback on browsers. Supports playback on applications on some Android devices.
Widevine (L3)	**High** Software-based decryption is required to play the encrypted content. Strong protection against decryption tools and extensions. Relatively strong protection against screencapturing tools.	**Medium** Supports playback on Chrome and some Chromium-based browsers. Supports playback on applications on some Android devices.



Basically, the protection level and compatibility of an encryption method are inversely related. That is, the higher the protection level, the lower the compatibility. Therefore, you will need to find a balance between the two.
The guide below shows you how to use VOD's content protection schemes.

Best Practices

In this section, we will talk about the three aspects of copyright protection – content protection, unauthorized distributor tracking, and playback protection – and discuss how you can best use them to protect your content.

Content protection

As mentioned above, the protection level and compatibility of an encryption method are inversely related. In order to find a balance between the two, we recommend you choose your protection schemes according to video resolution.
HLS private encryption only supports playback resolutions lower than 720p.
Widevine and FairPlay DRM supports all playback resolutions.
The VOD Player SDK will try to play Widevine- or FairPlay-encrypted content first. If a device does not support commercial-grade DRM, the Player SDK will automatically switch to the output of HLS private encryption.
This gives your high-value content (720p or above) strong protection (implemented by commercial-grade DRM) against decryption and screencapturing. Meanwhile, the HLS private encryption scheme allows you to improve the compatibility of your service. Even if decryption or unauthorized recording occurs, it would only affect your less valuable content (below 720p).

Unauthorized distributor tracking

Due to the relatively low compatibility of commercial-grade DRM, you may need to use DRM solutions together with HLS private encryption. In such cases, you can use floating watermarks or digital watermarks to complement the protection offered by private encryption. When unauthorized recording occurs, the watermarks help you identify the distributor.
A floating watermark is an overlay added to a video when it is played on the viewer's device. A typical floating watermark displays the ID of the viewer. This offers an extremely low-cost way for you to deter piracy.
A digital watermark is encoded into images and audio. If your content is recorded without authorization, you will be able to extract the ID of the distributor from the cloud. Compared with floating watermarks, digital watermarks have less impact on viewing experience (invisible to viewers) and offer higher protection levels (the watermark is encoded into your content and cannot be removed or covered).
We recommend you add both floating and digital watermarks to your content so as to effectively track unauthorized distributors when piracy occurs.

Playback protection

Hotlink protection offers protection for a playback URL. We recommend the following configurations:
Enable referer hotlink protection and add your domains to the allowlist so that other domains cannot access your content.
Enable key hotlink protection and set the validity period of a playback URL to 30 minutes longer than the video duration.
Enable key hotlink protection and set the maximum number of IP addresses allowed to access your playback URL to three.
Enable key hotlink protection and set the region allowed to access your playback URL to the country or region of your domain.

Best practices

This section provides detailed directions on how to use the above-mentioned content protection schemes.

Obtaining a FairPlay certificate

In order to use the FairPlay encryption scheme, you must first request a FairPlay certificate and submit the certificate information to VOD.

Encrypting content and adding digital watermarks

Upload your content to VOD and generate adaptive bitrate outputs that are encrypted by the HLS private encryption scheme, the FairPlay DRM encryption scheme, and the Widevine DRM encryption scheme.
1. Log in to the VOD console, select **Media Assets > Video/Audio Management** on the left sidebar, and click **Upload** to upload your media file.
2. After upload, select your media file, and click **Task Flow**. In the pop-up window, select the "MultiDRMPreset" task flow, and click **Confirm**.

The task will generate a file that is encrypted by the HLS private encryption scheme (only 480p and lower resolutions), as well as files encrypted by FairPlay and Widevine DRM. It will also add digital watermarks to all the files.

Note:

Because the "MultiDRMPreset" task flow will add digital watermarks to the outputs of HLS private encryption, please make sure your video is longer than six minutes. Otherwise, the task will fail.

Enabling hotlink protection

Configuring referer and key hotlink protection:

1. Log in to the VOD console and select **Distribution and Playback > Domain Name** on the left sidebar. Find your domain, and click **Set**.
2. Select the **Access Control** tab. Toggle **Referer hotlink protection** on. In the pop-up window, select **Allowlist** and enter the domains you want to allow to access your content, and click **Confirm**.
3. Toggle **Key hotlink protection** on. In the pop-up window, enter or generate a hotlink protection key, and click **Confirm**.

Now, you have enabled referer and key hotlink protection for your domain.

Generating a player signature

A player signature is required to play encrypted videos using VOD's player. Specify the fields as follows:

1. For `appId` and `fileId`, pass in your account APPID and the file ID of the content to play respectively.
2. Set `audioVideoType` in `contentInfo` to `ProtectedAdaptive`.
3. For `drmAdaptiveInfo`, set `privateEncryptionDefinition`, `widevineDefinition`, and `fairPlayDefinition` to `14`, `21`, and `12` respectively.
4. For `urlAccessInfo`, set `t` to the expiration time of the playback URL (current time + video length + 30 minutes is recommended); set `rlimit` to `3`; set `us` to a random string (randomly generated for each signature), and set `uv` to the viewer ID.

Playing an encrypted video

1. Integrate the Player SDK. For detailed directions, see Web Integration, iOS Integration Guide, and Android Integration Guide.
2. Configure the floating watermark displayed when the video is played. For detailed directions, see Web Integration, Android Integration Guide, and iOS Integration Guide.
3. The player obtains the signature from your server and starts playing the video.

On devices that support FairPlay and Widevine, the outputs of commercial-grade DRM will be played. On devices that do not support FairPlay and Widevine, the output of HLS private encryption will be played.

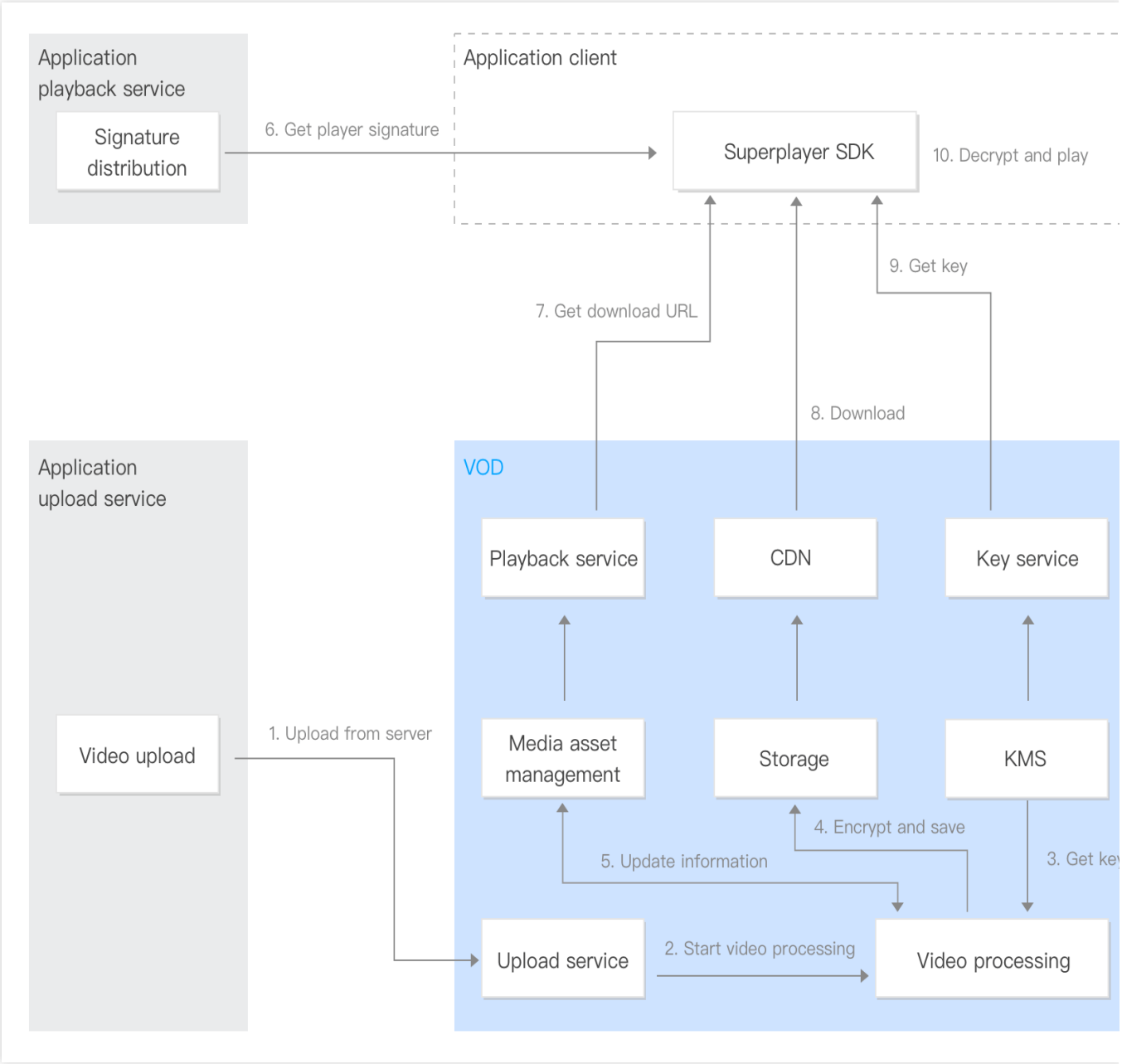
HLS Private Encryption

Last updated：2022-05-27 10:50:23

HLS private encryption is VOD's proprietary video encryption solution. It uses a private protocol to prevent key leakage and provides better protection against cracking by browser extensions and other tools.

Workflow

The figure below shows the workflow of HLS private encryption.



1. **Upload from server**: Videos are uploaded to VOD via the console or using server-side APIs.
2. **Start video processing**: After upload, video encryption starts (videos are encrypted according to the adaptive bitrate streaming parameters specified).
3. **Get the key**: VOD gets the encryption key from the KMS module.
4. **Encrypt and save the videos**: VOD encodes and encrypts the videos and saves the outputs.
5. **Update the information**: The information of the encrypted videos is updated to the media asset management module.
6. **Get player signatures**: The VOD superplayer, which is integrated into your project, requests player signatures from your server.
7. **Get the download URLs**: The superplayer gets the download URLs of the videos from VOD.
8. **Download**: The superplayer downloads the encrypted content via the URLs from the CDN of VOD.

9. **Get the key**: The superplayer sends a request that carries the signature for the key, which is protected by VOD's private protocol against leakage.
10. **Decrypt and play the content**: The superplayer uses the private protocol to get the key and decrypt and play the content.

Directions

For detailed directions on how to use VOD's encryption feature, see Stage 4. Play back an encrypted video.

FAQs

1. How do I encrypt uploaded videos using HLS private encryption?
VOD's adaptive bitrate streaming feature allows you to convert a video into multiple resolutions and encrypt the content. For detailed directions, see Stage 4. Play back an encrypted video.
2. How to play encrypted videos?
A client must be integrated with the superplayer SDK in order to play encrypted videos. You also need to build a signature generation tool. For detailed directions, see Stage 4. Play back an encrypted video.

DRM Encryption
DRM Overview

Last updated：2025-04-14 17:26:24

Copyright infringement has grown alongside the rapid development of the online video industry, making copyright protection a major concern of content owners.
Established DRM solutions use playback licenses to offer high-level content protection. Before a device can play a DRM-encrypted video, it must obtain a license (which includes information such as the decryption key, validity period of the key, and device information) to decrypt the video.
The strengths of established DRM solutions are as follows:
The key can only be read by the content decryption module (CDM).
Each license can be used for only one device.
You can set the validity period of a license.
Support hardware-based TEE and decoding.
Widevine and FairPlay are two mainstream DRM solutions.

DRM Solution	Adaptive Bitrate Streaming Protocol	Player and Browser
Widevine	HLS, DASH	Android player, Chrome, Firefox, Edge, Opera
FairPlay	HLS	iOS player, Safari

Currently, VOD supports two DRM licensing schemes:
VOD DRM: The licensing service is provided by Tencent Cloud VOD.
Third-party DRM: The licensing service is provided by SMDC.
You can choose the scheme that fits your needs.

VOD DRM Scheme

Established DRM solutions provide high-level protection for your video content, but may be difficult to implement from scratch. VOD offers an easy-to-use DRM scheme that is built on established DRM solutions and integrates a full range of features including DRM encryption, license management, license distribution, decryption, and playback.
The encryption and decryption process is as follows:

We offer a tutorial that uses an example to show you how to quickly implement the scheme.

Third-Party DRM Scheme

If you use this scheme, VOD will offer services including transcoding, encryption, storage, and CDNs, while the third-party DRM service provider SDMC will offer certificate management and license distribution services.
The encryption and decryption process is as follows:

We offer a tutorial that uses an example to show you how to quickly implement the scheme.

Billing

The following fees may be incurred for using the DRM feature:
Transcoding fees: Videos are transcoded during DRM encryption, which incurs transcoding fees.
Storage fees: The videos generated after transcoding take up storage space, which incurs storage fees.
DRM licensing fee: For a device to play a DRM-encrypted video, you must supply it with a license. This will incur DRM licensing fees. If you use the third-party DRM scheme, this fee will be charged by the third-party DRM service provider.
For the pricing details, see Daily Pay-As-You-Go.

Obtaining FairPlay Certificate Information

Last updated：2022-12-30 16:58:31

To use the FairPlay streaming (FPS) technology, you need to request from Apple an FPS deployment package. This document shows you how to obtain this package as well as the following information:

FairPlay Streaming (FPS) certificate (format: CER)

Private key file (format: PEM)

Private key password

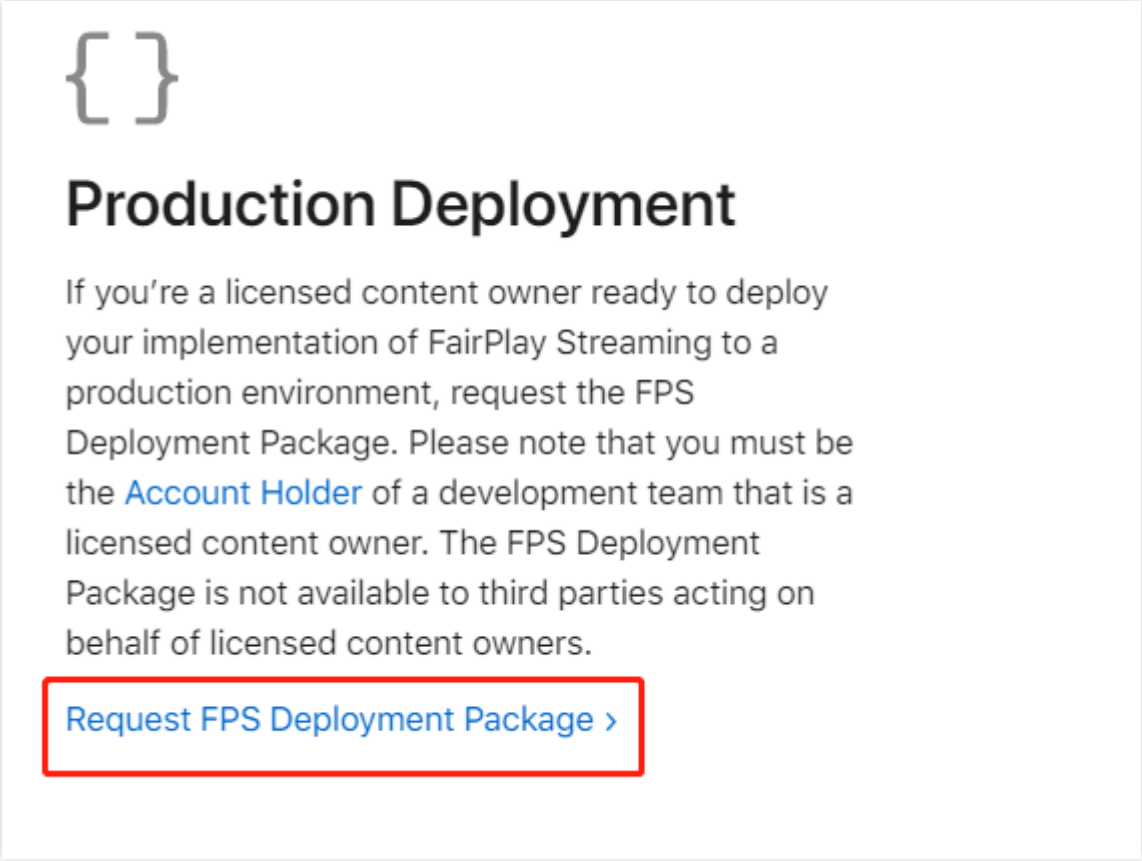
Application secret key (ASK)

Step 1. Request FairPlay Streaming Deployment Package

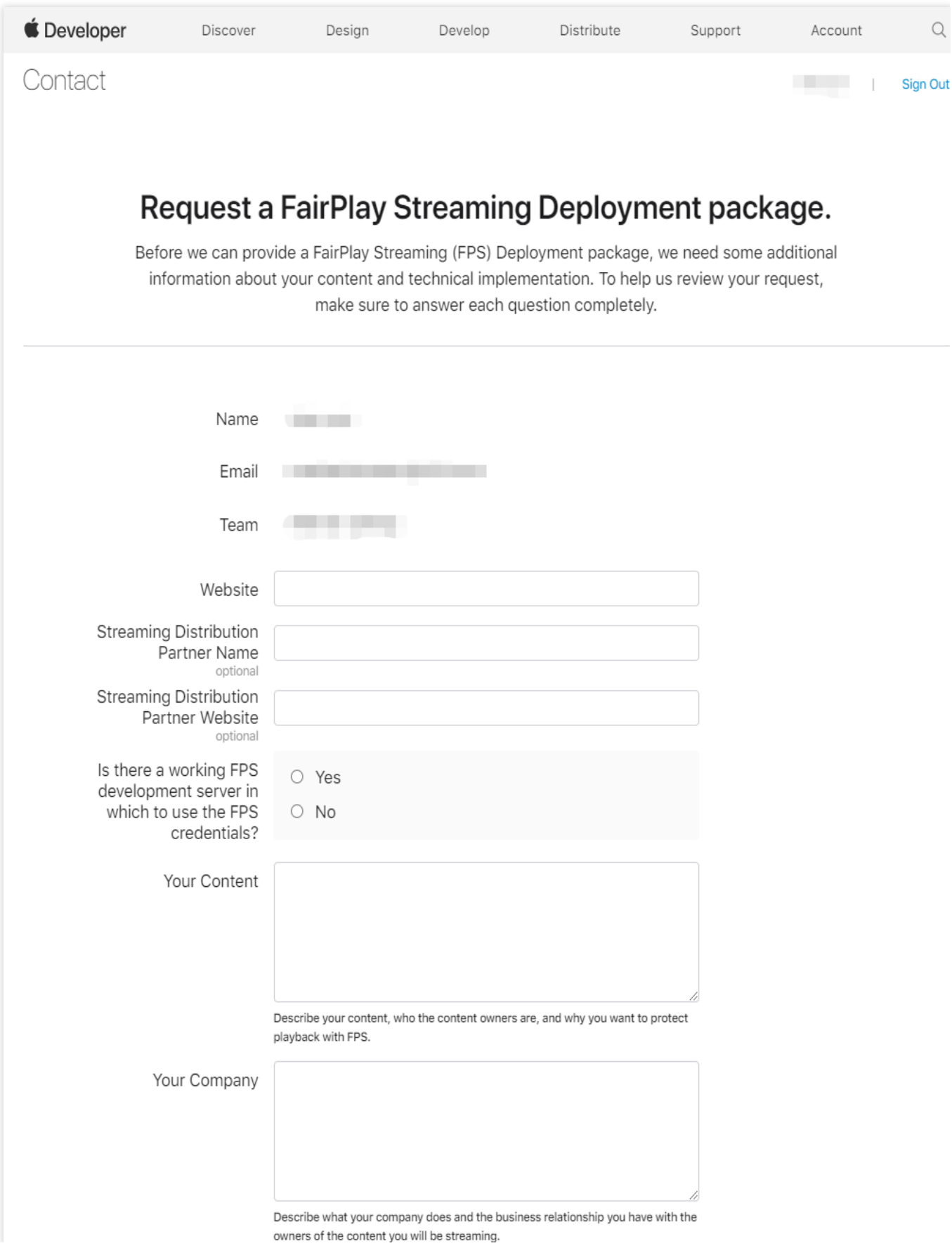
1. Go to the Apple FairPlay page, scroll to the bottom, and click **Request FPS Deployment Package**. A form will pop up.

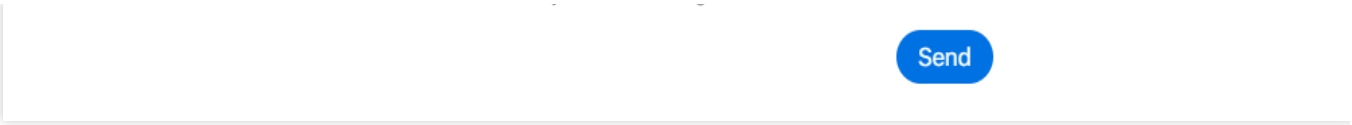
Note:

You need to log in with an Apple developer account.



2. Fill out the form and submit it.





3. After your request is approved, you will be issued an `FPS_Deployment_Package.zip` package.

Note:

When asked if you have implemented and tested Key Security Module (KSM), you can paste the answer below:

```
> I am using a 3rd party DRM company and the company has already built and tested KSM
```

Step 2. Create a Private Key and a Certificate Signing Request (CSR)

Unzip `FPS_Deployment_Package.zip` and create a password-protected private key file and a CSR file as instructed in the guide document (PDF) in the package.

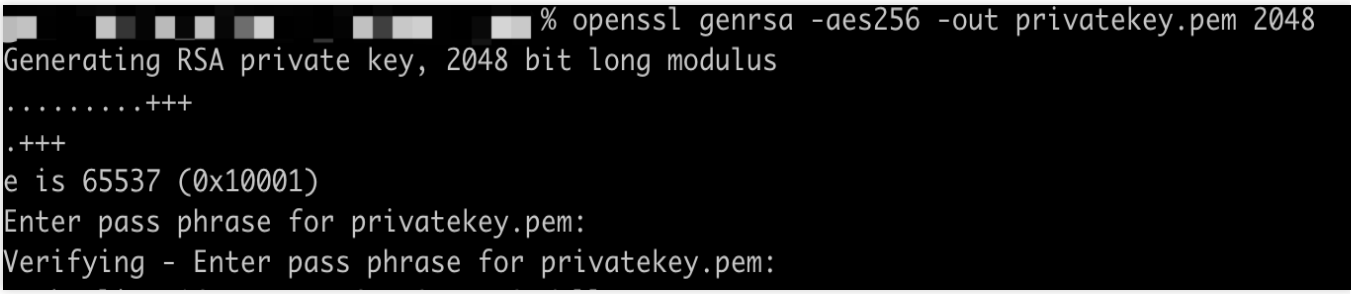
Note:

Make sure OpenSSL is installed on the computer or server environment where this process is performed.

1. Run the command below to create a private key file (`privatekey.pem`):

```
openssl genrsa -aes256 -out privatekey.pem 2048
```

You need to set a password (preferably not longer than 32 characters) for the private key. Note the password for later use.



2. Run the command below to create a CSR file (certreq.csr):

```
openssl req -new -sha1 -key privatekey.pem -out certreq.csr -subj "/CN=SubjectName/
```

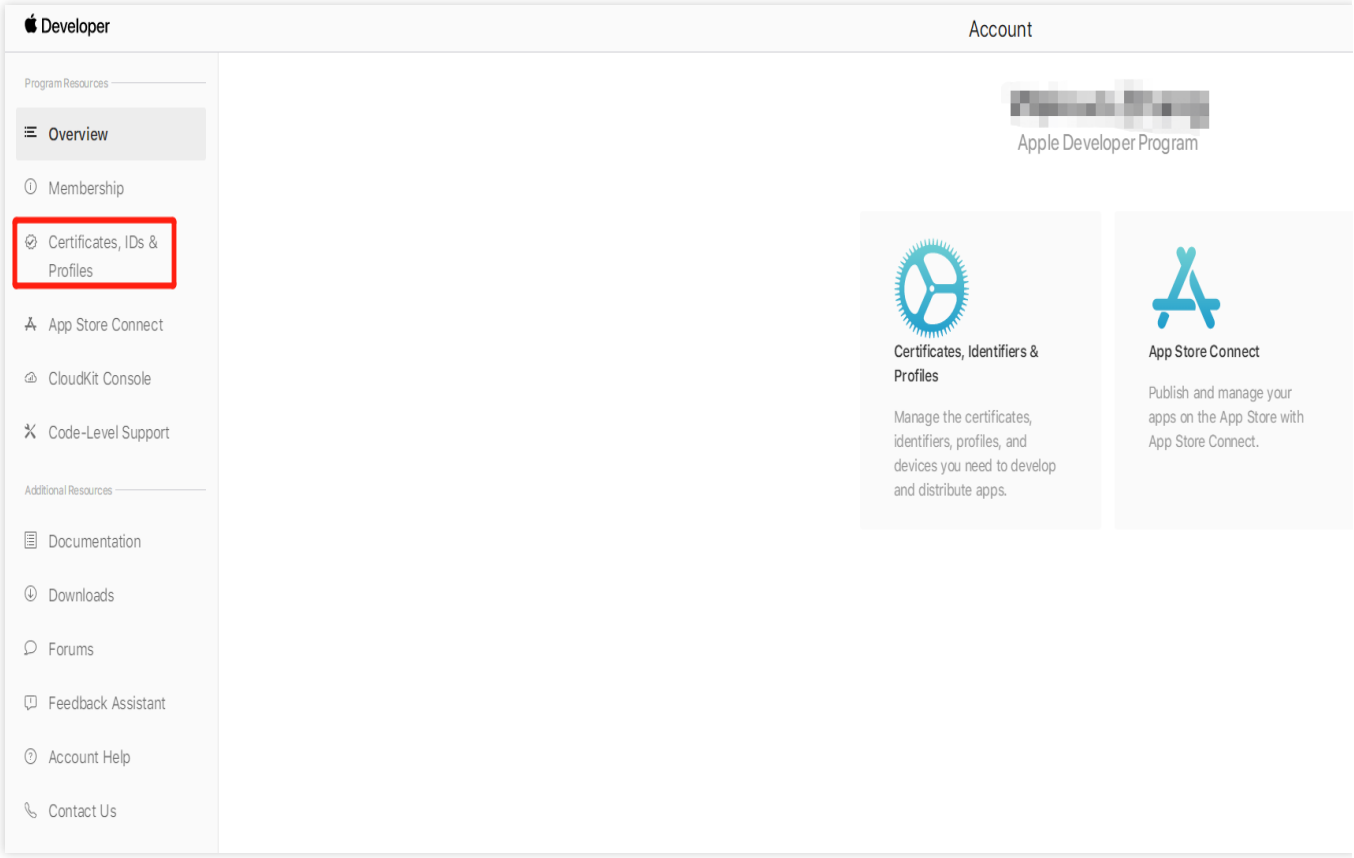
You need to enter the private key password.

```
[redacted] % openssl req -new -sha1 -key privatekey.pem -out certreq.csr -subj "/CN=SubjectName/OU=OrganizationalUnit/O=Organizatior
Enter pass phrase for privatekey.pem:
```

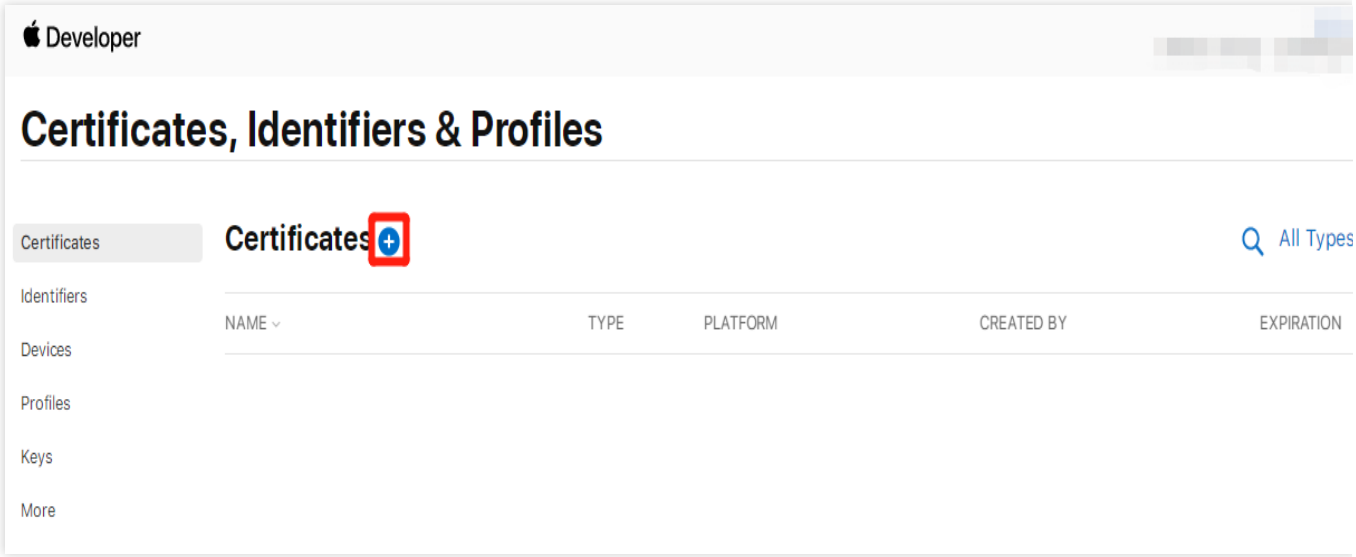
Step 3. Generate the FPS Certificate

Get the FPS certificate and ASK from the Apple developer page.

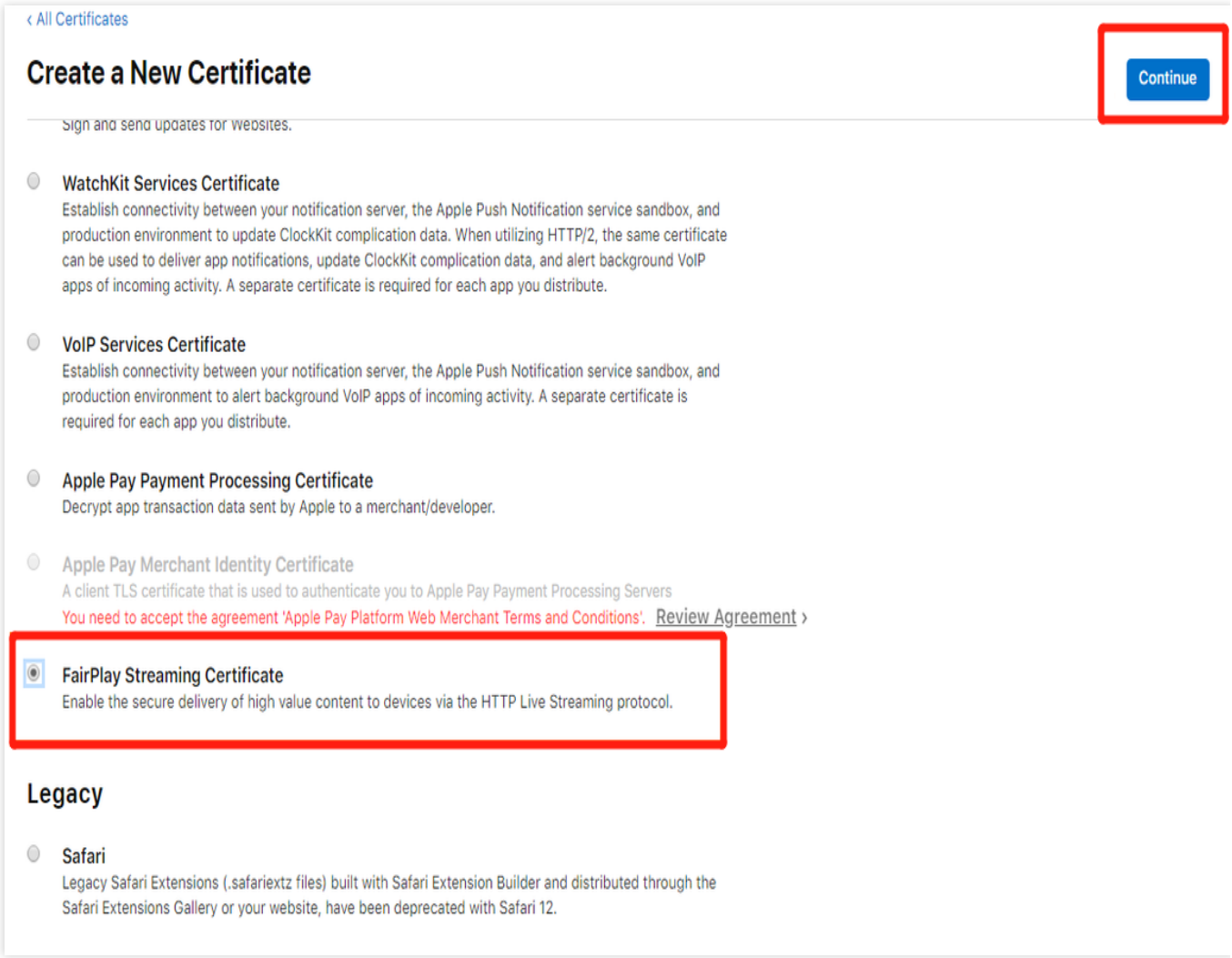
1. Go to the Apple developer page and click **Certificates, IDs & Profiles** in the left sidebar.



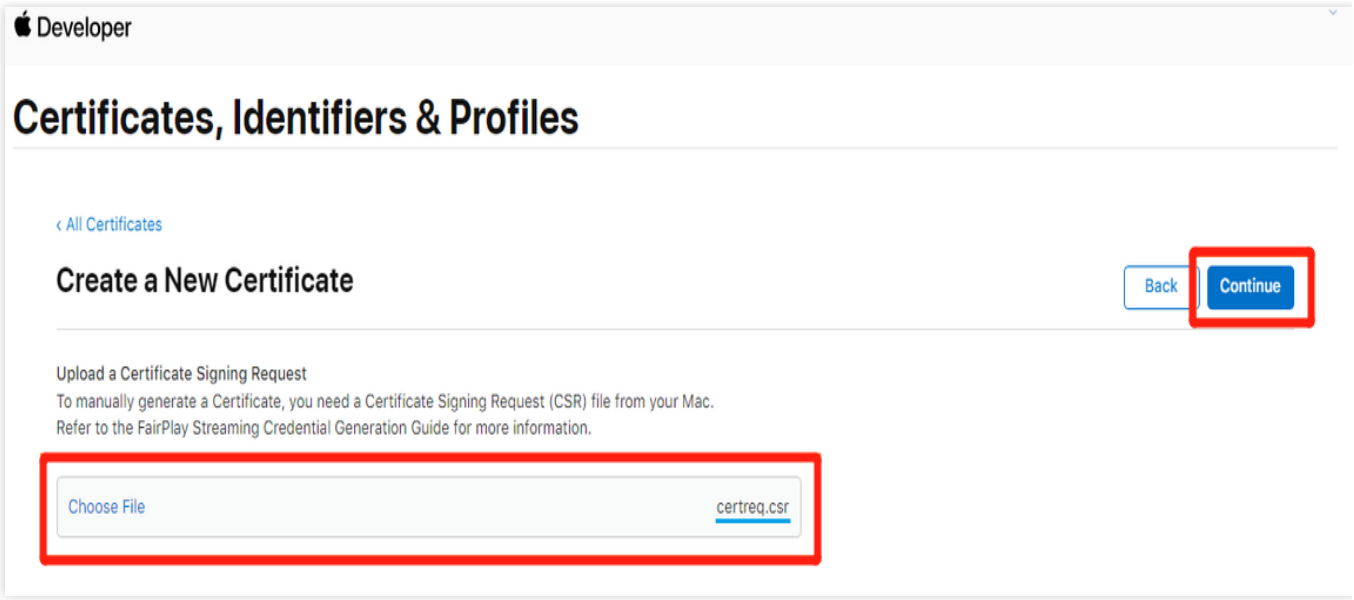
2. Click **+**.



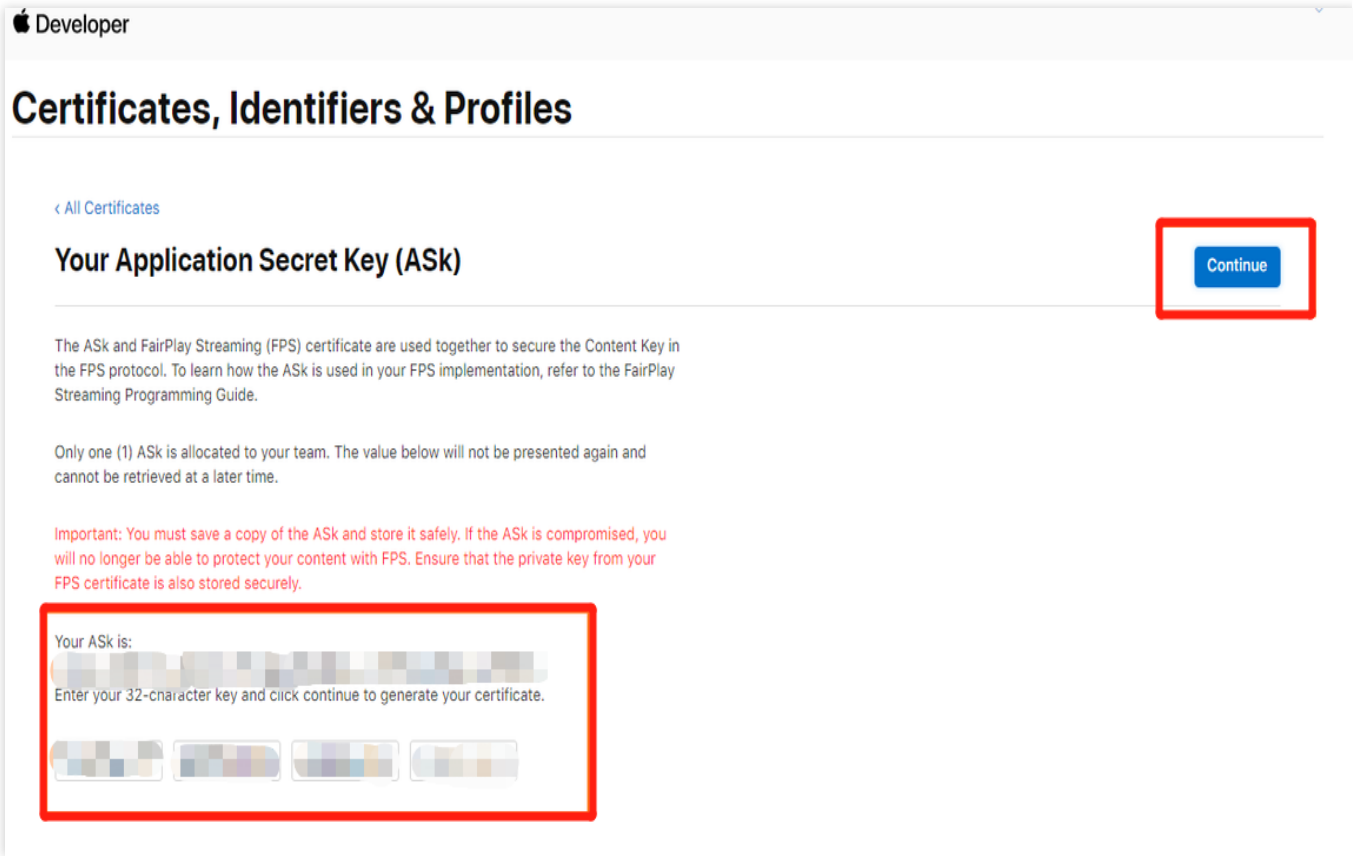
3. Select **FairPlay Streaming Certificate** and click **Continue**.



4. Click **Choose File**, select the `certreq` file created in Step 2, and click **Continue**.



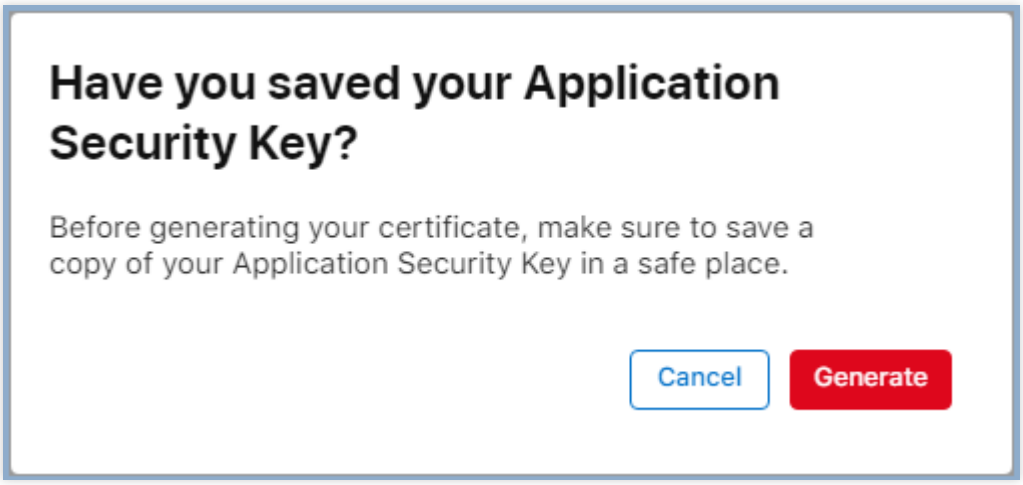
5. Copy the ASK, enter it into the input field below, and click **Continue**.



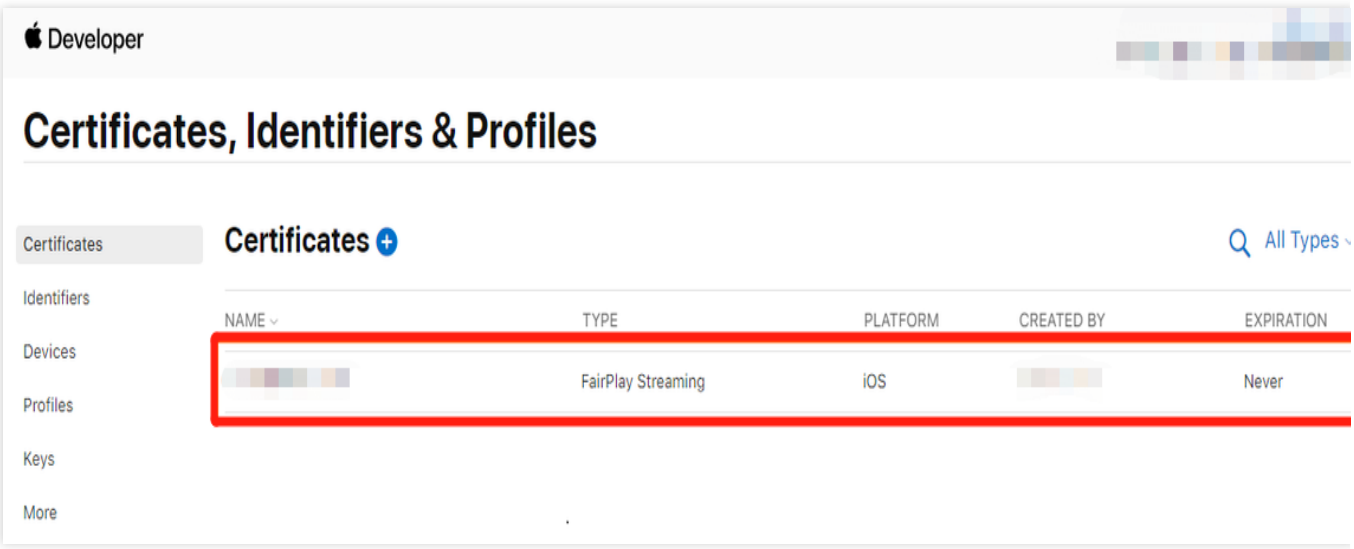
6. A window will pop up to confirm that you have saved the ASK. Click **Generate**.

Note:

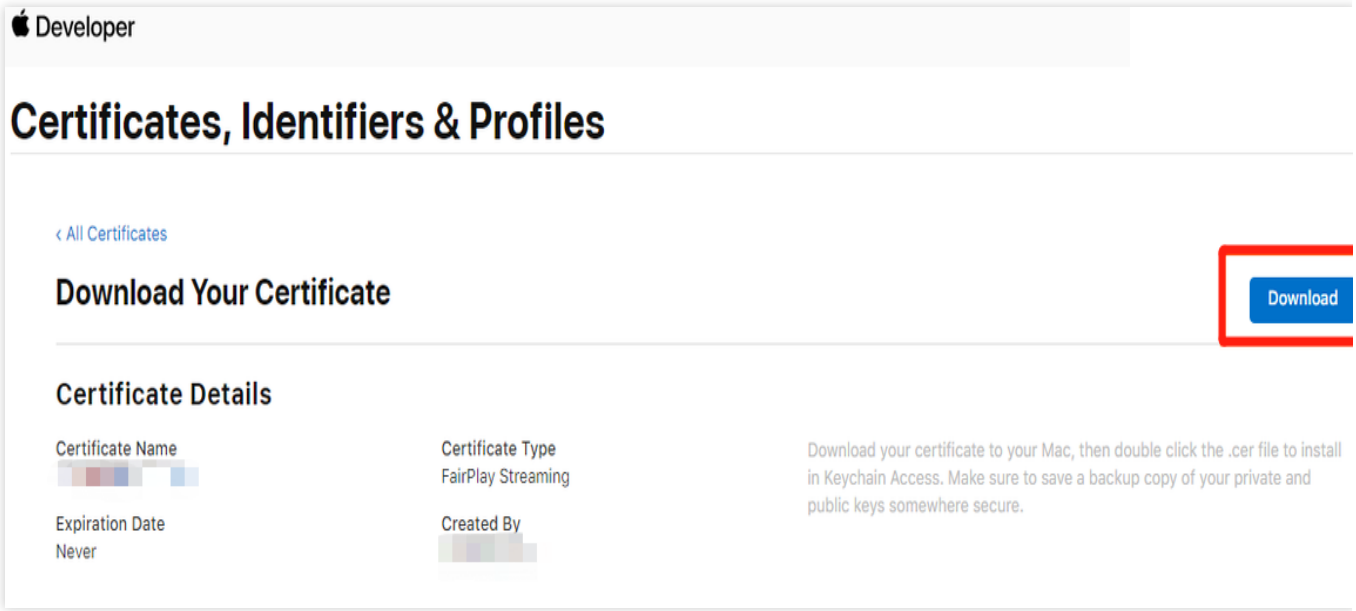
Note: Make sure you save a copy of the ASK. You will be unable to view it afterwards.



7. After the above steps are completed, the FPS certificate generated (type: FairPlay Streaming) will appear in the certificate list.



8. Click **Download** to download the FPS certificate (`fairplay.cer`).



Summary

You have now obtained the necessary FairPlay certificate information.

VOD DRM Scheme
Submitting FairPlay Certificate Information to VOD

Last updated：2022-08-31 17:54:01

This document shows you how to submit the following FairPlay certificate information to the VOD console:

FairPlay Streaming (FPS) certificate (format: CER)

Private key file (format: PEM)

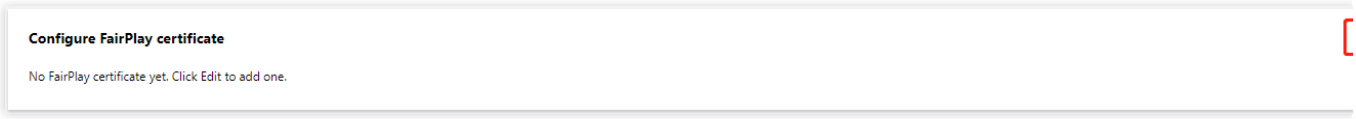
Private key password

Application secret key (ASK)

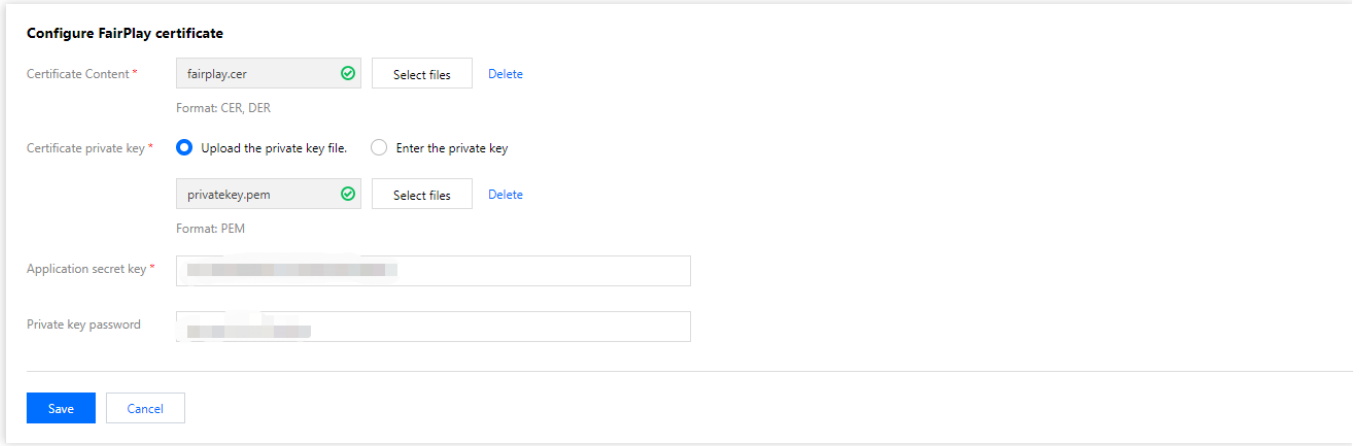
If you don't have a FairPlay certificate yet, refer to Obtaining FairPlay Certificate Information to obtain the information.

Directions

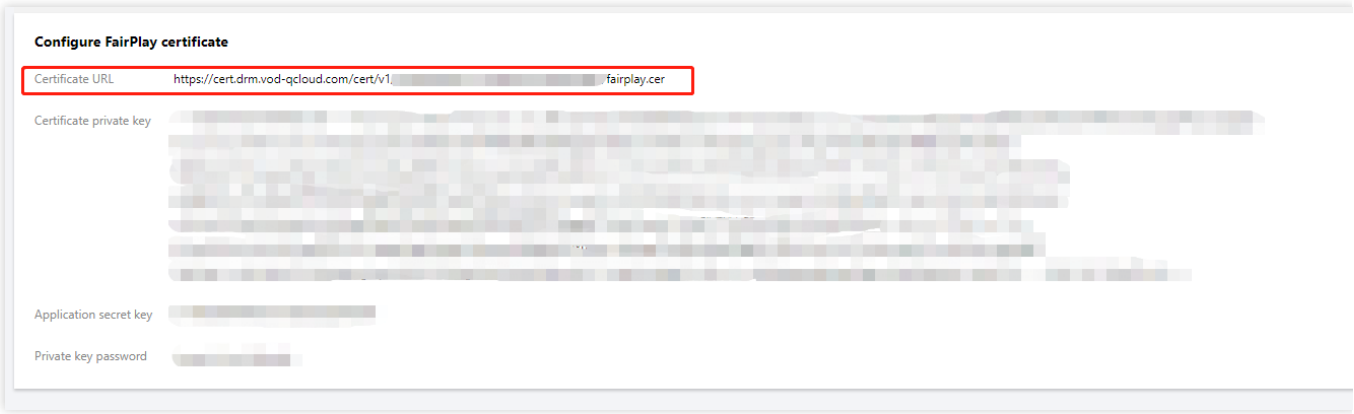
1. Log in to the VOD console.
2. Select **Media Processing > DRM Configuration** on the left sidebar and click **Edit**.



3. Upload the certificate file (`fairplay.cer`) and private key file (`privatekey.pem`) and enter the private key password and ASK.



4. Click **Save**. You will see your certificate information.



Summary

You have now submitted your FairPlay certificate information to the VOD console.

Playing DRM-Encrypted Videos

Last updated：2022-09-13 10:09:55

Overview

This document shows you how to encrypt videos using DRM solutions and play the encrypted videos with a 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 and complete identity verification.
2. Purchase VOD services. For details, see Billing Overview.
3. Go to the VOD console.

At this point, you have activated VOD.

Obtaining FairPlay certificate information

See Obtaining FairPlay Certificate Information.

Submitting the certificate information

See Submitting FairPlay Certificate Information to VOD.

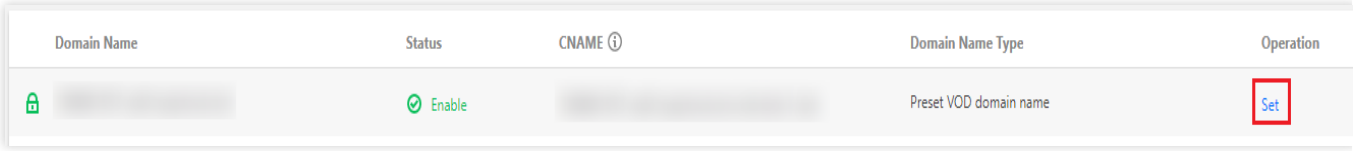
Step 1. Enable Hotlink Protection

The example below shows how to enable key hotlink protection for the default distribution domain under your account:

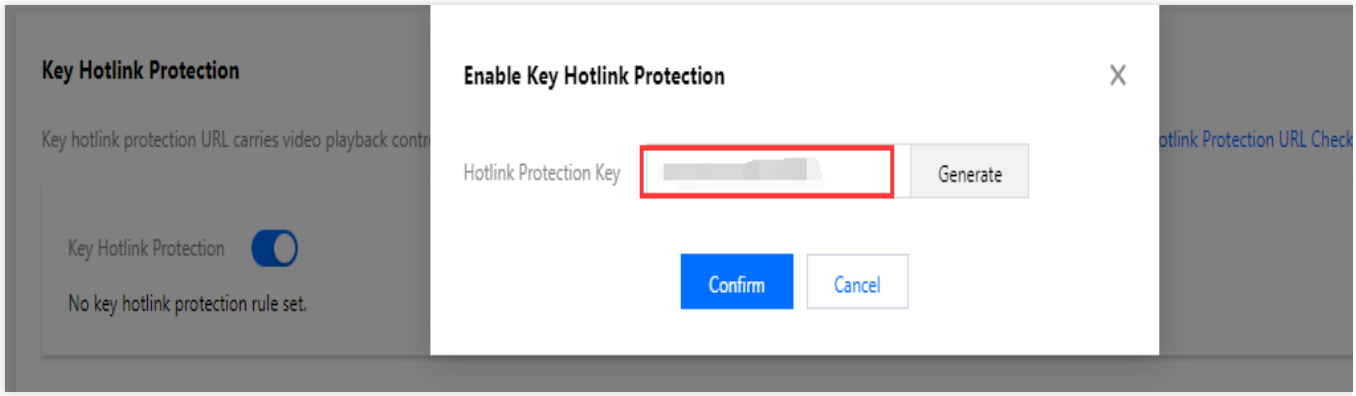
Note:

We do not recommend enabling hotlink protection for a domain name already in use. Doing so may cause failure to play existing videos.

1. Log in to the VOD console, select **Distribution and Playback** > Domain Name on the left sidebar. Find the default distribution domain, click **Set** on the right, and select the **Access Control** tab.

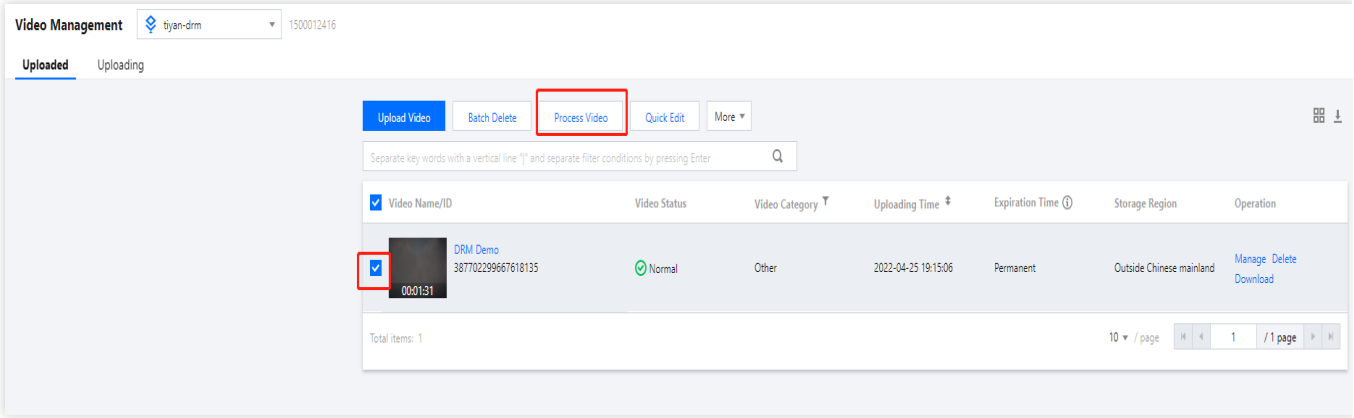


2. Toggle on **Key Hotlink Protection**. In the pop-up window, click **Generate** to generate a random key (suppose it is `vodtestkey`). Copy the key and click **Confirm**. You will use the key later to generate playback signatures.



Step 2. Encrypt a Video

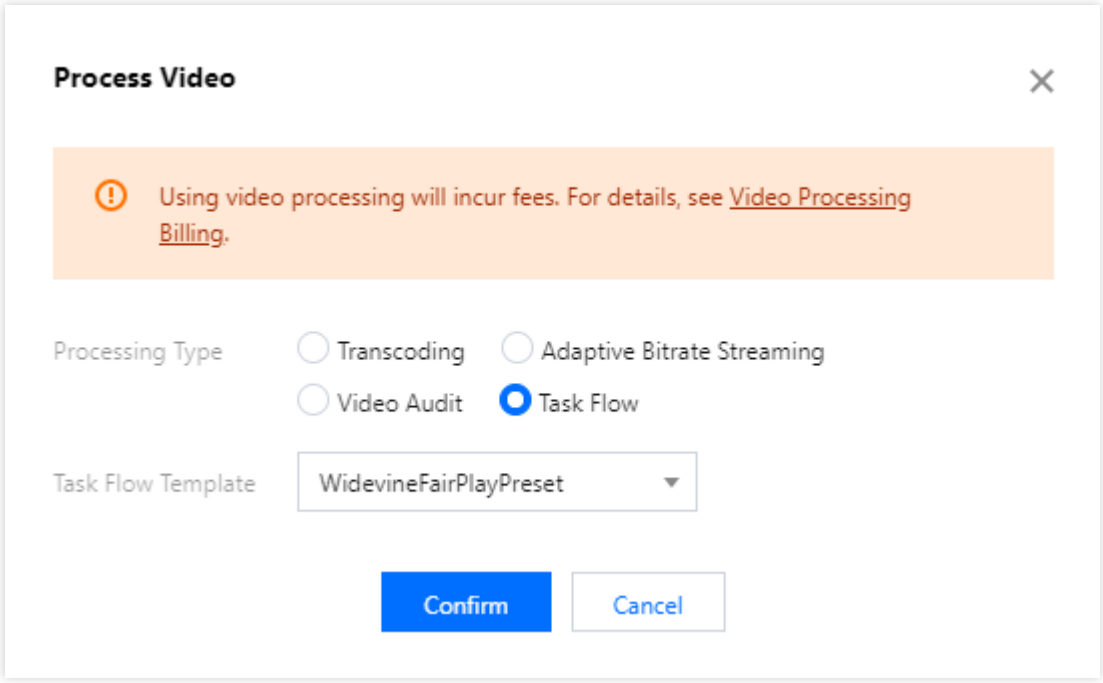
1. In the VOD console, select **Media Assets** > Video/Audio Management on the left sidebar, select the target video (in this example, the file ID of the video encrypted is `387702304941991610`), and click **Process**.



2. On the video processing page:

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

Select **WidevineFairPlayPreset** as the **Task Flow Template**.

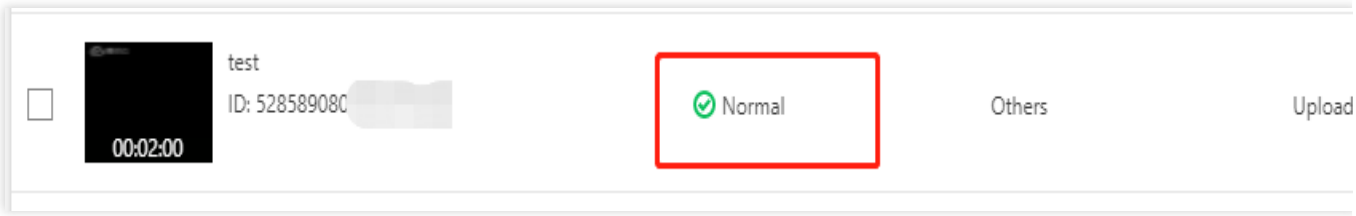


Note:

`WidevineFairPlayPreset` is a preset task flow. It uses the adaptive bitrate streaming template 11 or 13, the time point screenshot template 10 (for thumbnail generation), and the image sprite template 10.

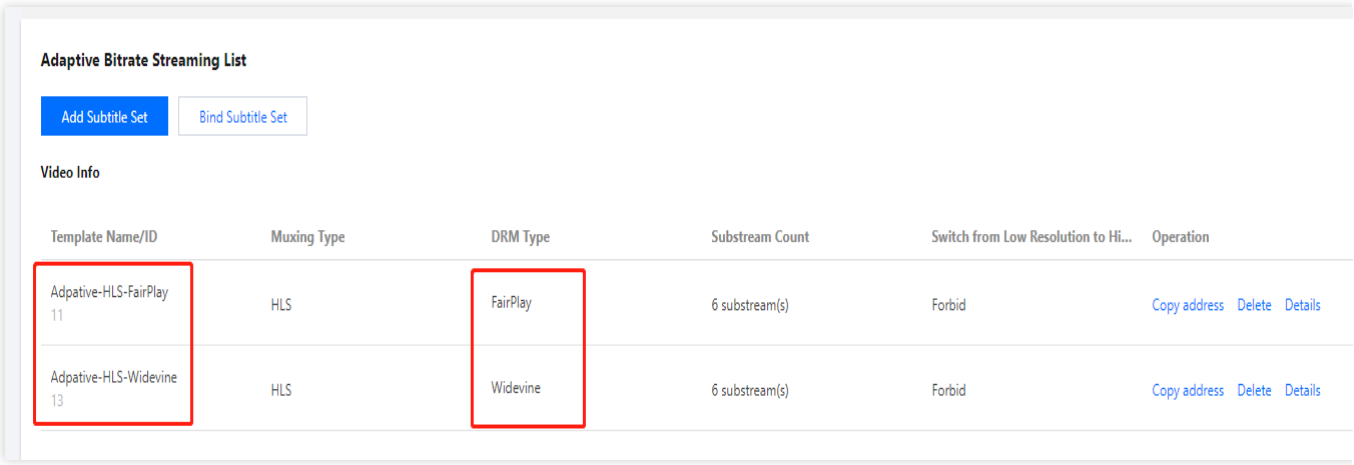
The adaptive bitrate streaming template 11 generates multi-bitrate streams encrypted by FairPlay, and the adaptive bitrate streaming template 13 generates multi-bitrate streams encrypted by Widevine.

3. Click **Confirm** and wait until the **Video Status** changes from "Processing" to "Normal", which indicates that video processing is completed.

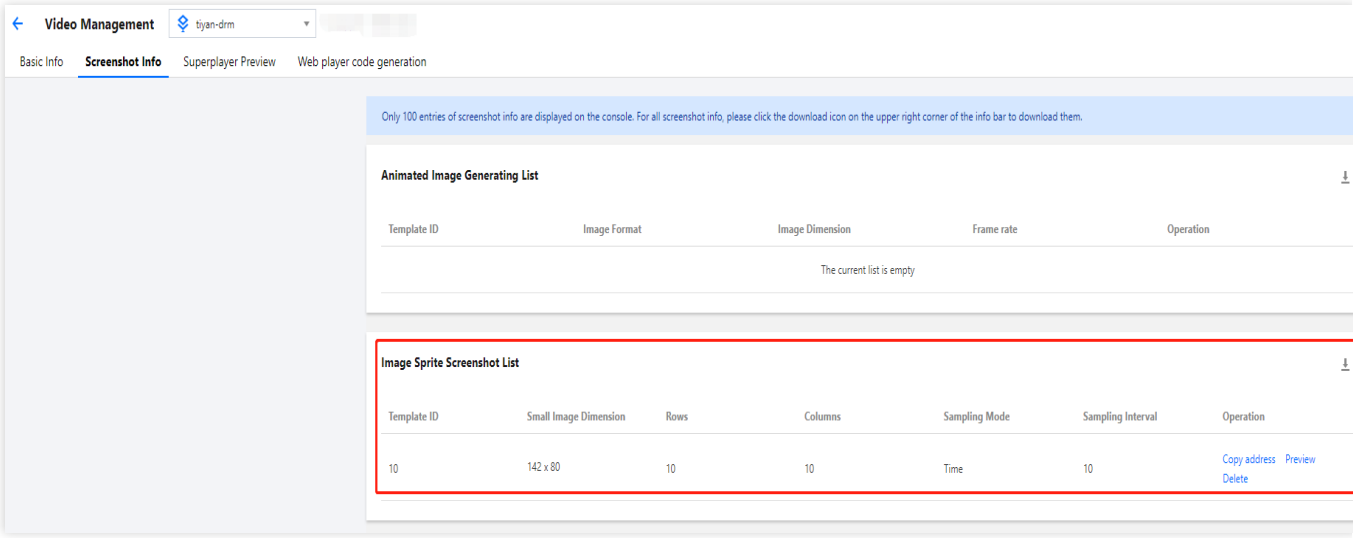


4. Click **Manage** in the **Operation** column of the video.

Under the **Basic Info** tab, you can view the thumbnail generated and outputs of adaptive bitrate streaming (template ID: 11 & 13).



Under the **Screenshot Info** tab, you can view the image sprite generated (template ID: 10).



Step 3. Generate a Player Signature

You will need the player signature to query past playback information. For directions on how to generate a player signature, see Superplayer Signature. For the example in this document, the payload for signature generation is as follows:

```
{
  "appId": 1500014561,
  "fileId": "387702304941991610",
  "currentTimeStamp": 1661163373,
  "expireTimeStamp": 2648557919,
  "pcfg":"advanceDrmPreset"
}
```

The key generated for the example in this document is `vodtestkey`, and the player signature (`psign`) generated is as follows:

```
eyJhbGciOiJIUzI1NiJ9.eyJhcHBJZCI6MTUwMDAxNDU2MSwiZmlsZUlkIjoiMzg3NzAyMzA0OTQxOTkxNjEwIiwiY3VycmVudFRpbWVTdGFtcCI6MTY2MTE2MzM3M
ywiZXhwaXJlVGltZVN0YW1wIjoyNjQ4NTU3OTE5LCJwY2ZnIjoiYWR2YW5jZURybVByZXNldCJ9.rEZLhjgsoLc2htIUI_HckxvhVmdBhQyf5d-2Kku1JeA
```

Step 4. Play the DRM-encrypted video

Web

Using the VOD player

To play the DRM-encrypted video using the VOD player, just pass in the file ID of the video and your VOD account's AppID when initializing the player.

Step 1. Import files

Import the player's style file and script files into the webpage.

```
 <link
href="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.4/tcplayer.min.cs
s" rel="stylesheet"/>
 <script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.4/libs/TXLivePlaye
r-1.2.3.min.js"></script>

 <script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.4/libs/hls.min.1.1
.5.js"></script>

 <script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.4/libs/flv.min.1.6
.3.js"></script>

 <script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.4/libs/dash.all.mi
n.4.4.1.js"></script>

 <script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.4/tcplayer.v4.5.4.
min.js"></script>
```

Step 2. Add a player container

Add a player container to wherever you want to display the player:

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

Note:

You can customize the container ID as well as the height and width of the container.

Step 3. Add the initialization code

Add the following script to your page initialization code and pass in the required initialization parameters (including the Player Signature `psign` generated in Step 3):

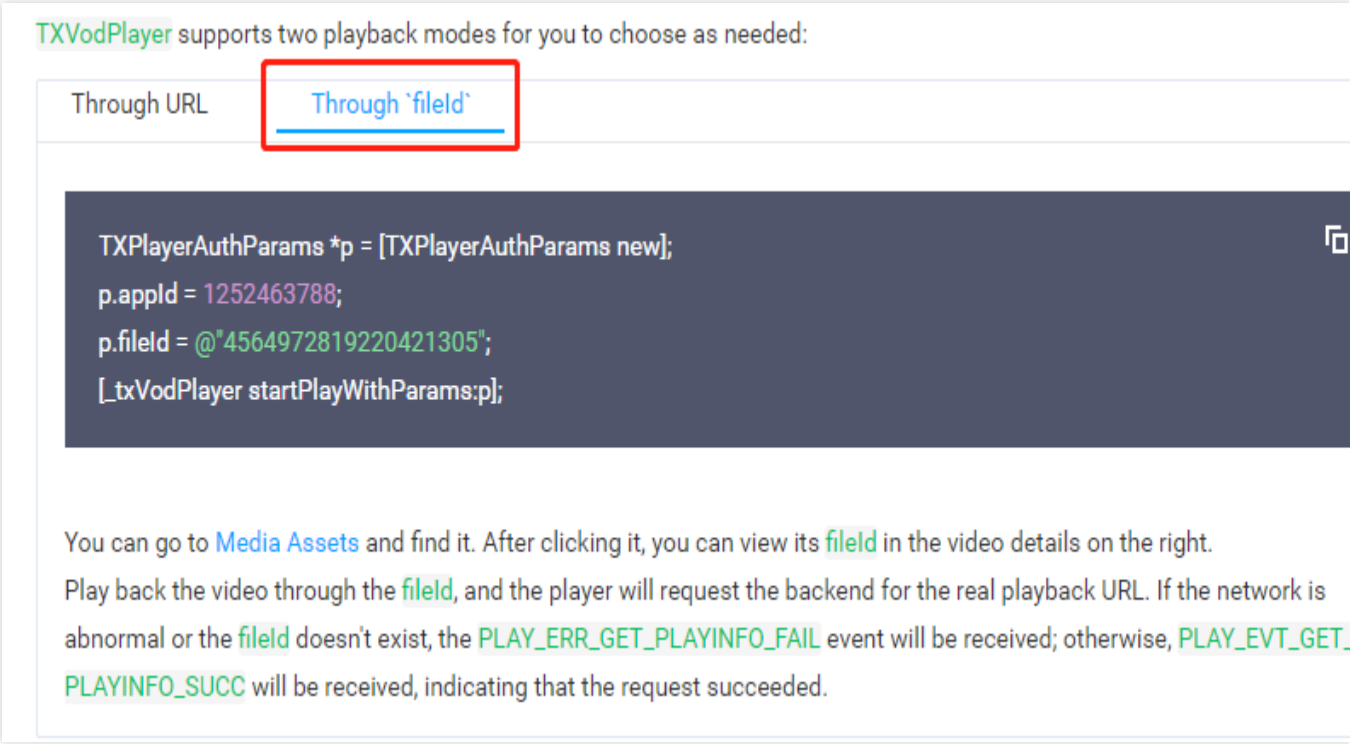
```
var player = TCPlayer('player-container-id', {
    appID: '1500014561' // The appID of your VOD account (required).
    fileID: '387702304941991610', // The file ID of the video to play
(required).
    psign:
'eyJhbGciOiJIUzI1NiJ9.eyJhcHBJZCI6MTUwMDAxNDU2MSwiZmlsZUlkIjoiMzg3NzAyMzA0OTQxO
TkxNjEwIiwiY3VycmVudFRpbWVTdGFtcCI6MTY2MTE2MzM3MywiZXhwaXJlVGltZVN0YW1wIjoyNjQ4
NTU3OTE5LCJwY2ZnIjoiYWR2YW5jZURybVByZXNldCJ9.rEZLhjgsoLc2htIUI_HckxvhVmdBhQyf5d
-2Kku1JeA',
    // For other parameters, see
https://intl.cloud.tencent.com/document/product/266/39105
});
```

iOS

To play the DRM-encrypted video on iOS, refer to Integration Guide (Through `FileId`). You need to use the player signature (`psign`) generated in Step 3 (Generate a Player Signature).

Note:

Please submit a ticket for the player SDK that supports DRM.

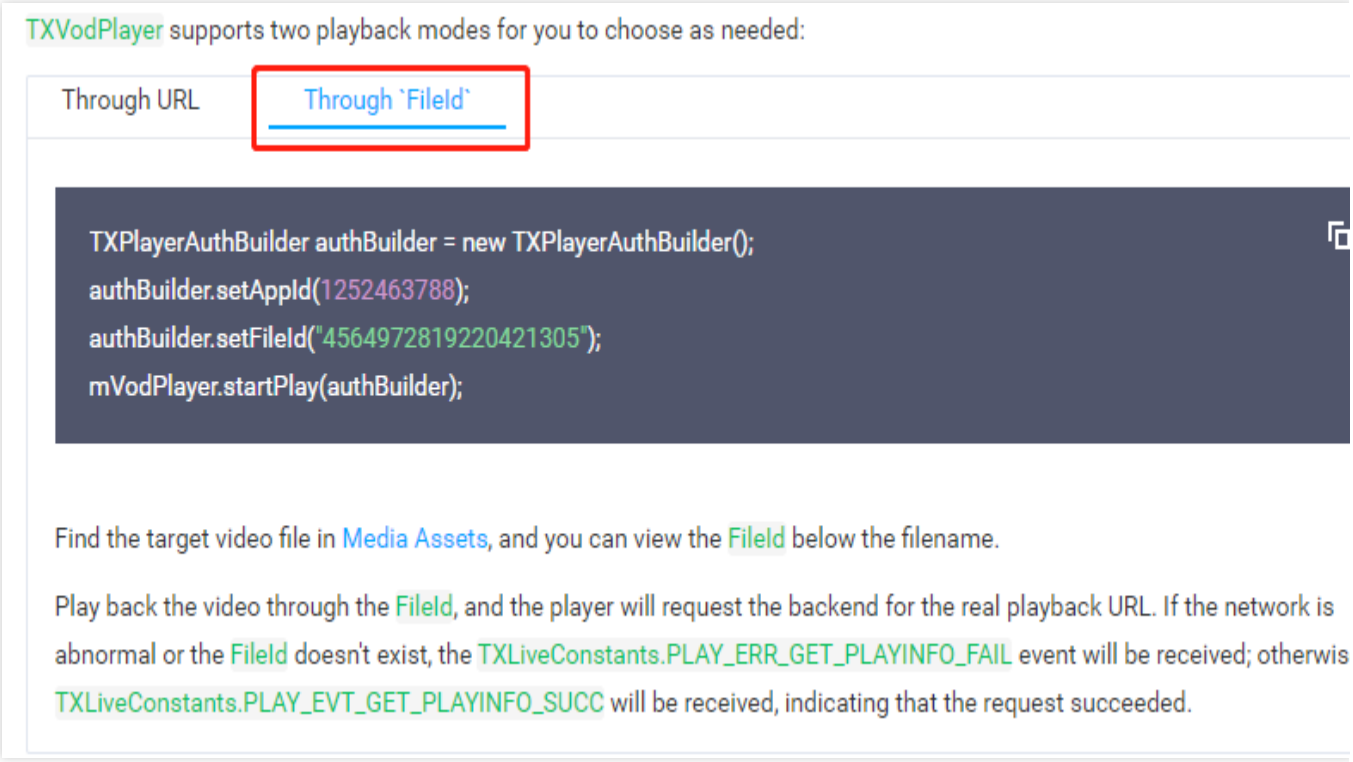


Android

To play the DRM-encrypted video on Android, refer to Integration Guide (Through `FileId`). You need to use the player signature (`psign`) generated in Step 3 (Generate a Player Signature).

Note:

Please submit a ticket for the player SDK that supports DRM.



Summary

Now, you have learned how to encrypt videos using DRM solutions and play the encrypted videos in a player.

Note:

If you have any questions, please submit a ticket.

Third-Party (SDMC) DRM Scheme Submitting FairPlay Certificate Information to SDMC

Last updated：2022-08-31 17:54:01

This document shows you how to submit the following FairPlay certificate information to the SDMC console:

FairPlay Streaming (FPS) certificate (format: CER)

Private key file (format: PEM)

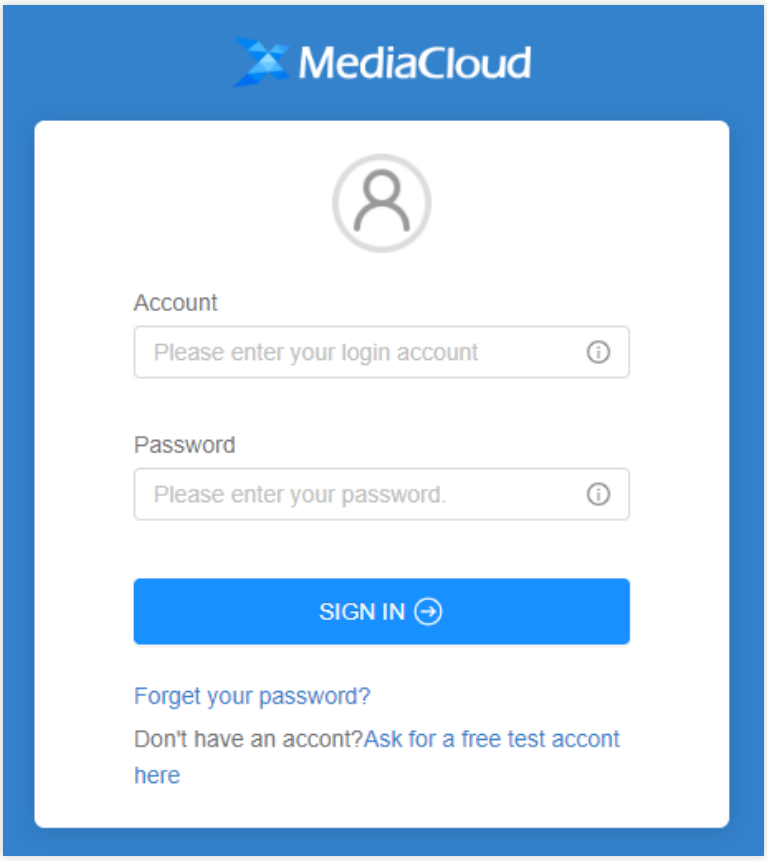
Private key password

Application secret key (ASK)

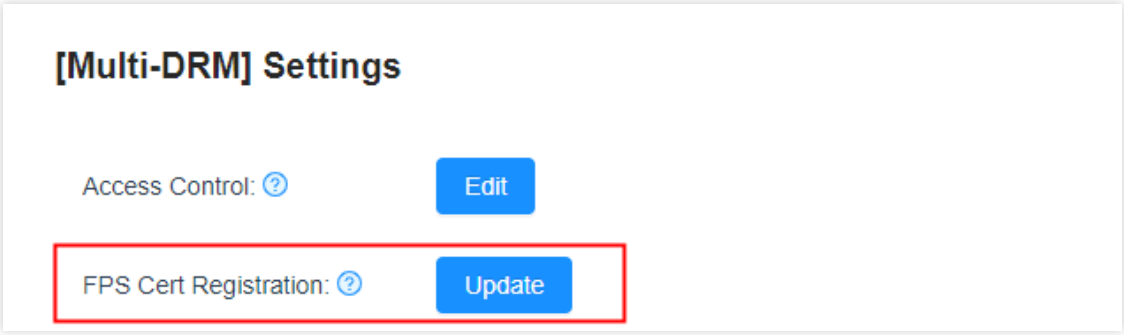
If you don’t have a FairPlay certificate yet, refer to Obtaining FairPlay Certificate Information to obtain the information.

Directions

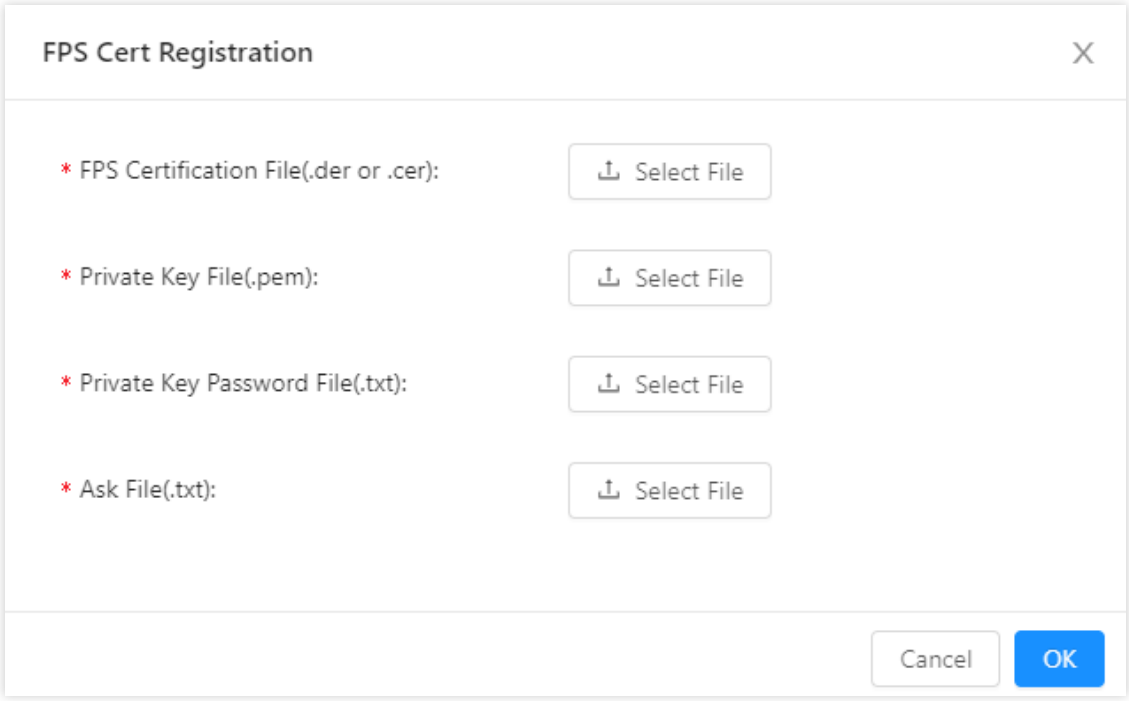
1. Log in to the SDMC DRM console.



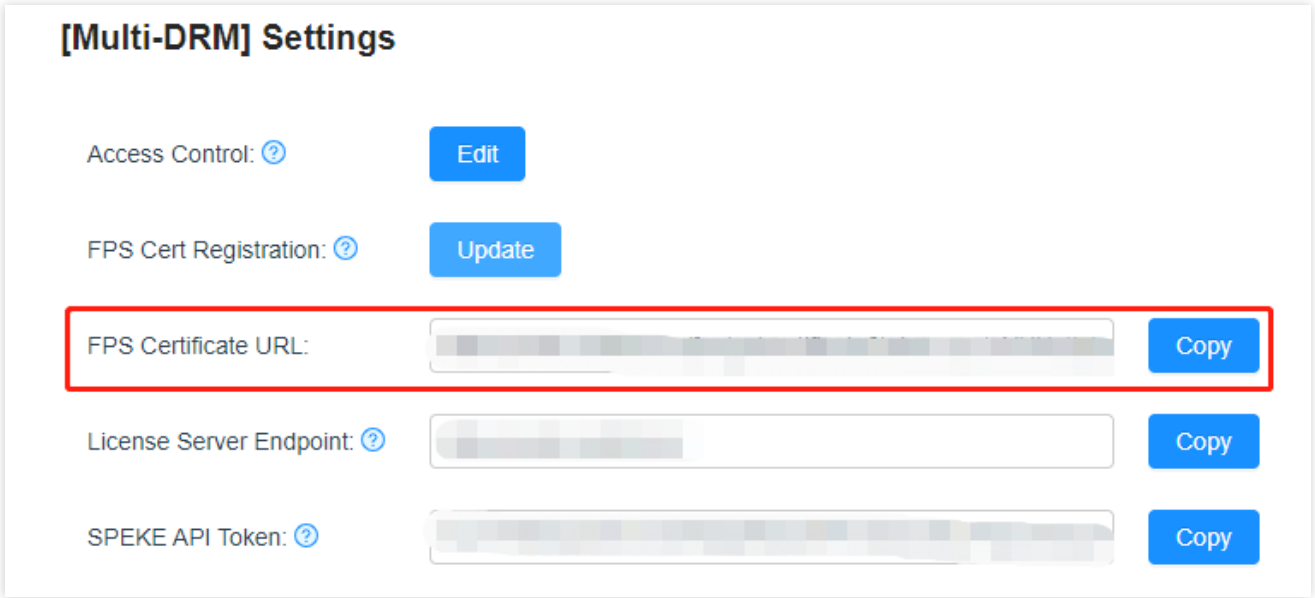
2. Click **DRM SETTINGS**, find **FPS Cert Registration**, and click **Update**.



3. Upload the FPS certificate, private key file, private key password file, and ASK file, and click **OK**.



4. After the files are uploaded, you will see the URL of your FPS certificate.



Summary

You have now submitted your FairPlay certificate information to the SDMC console.

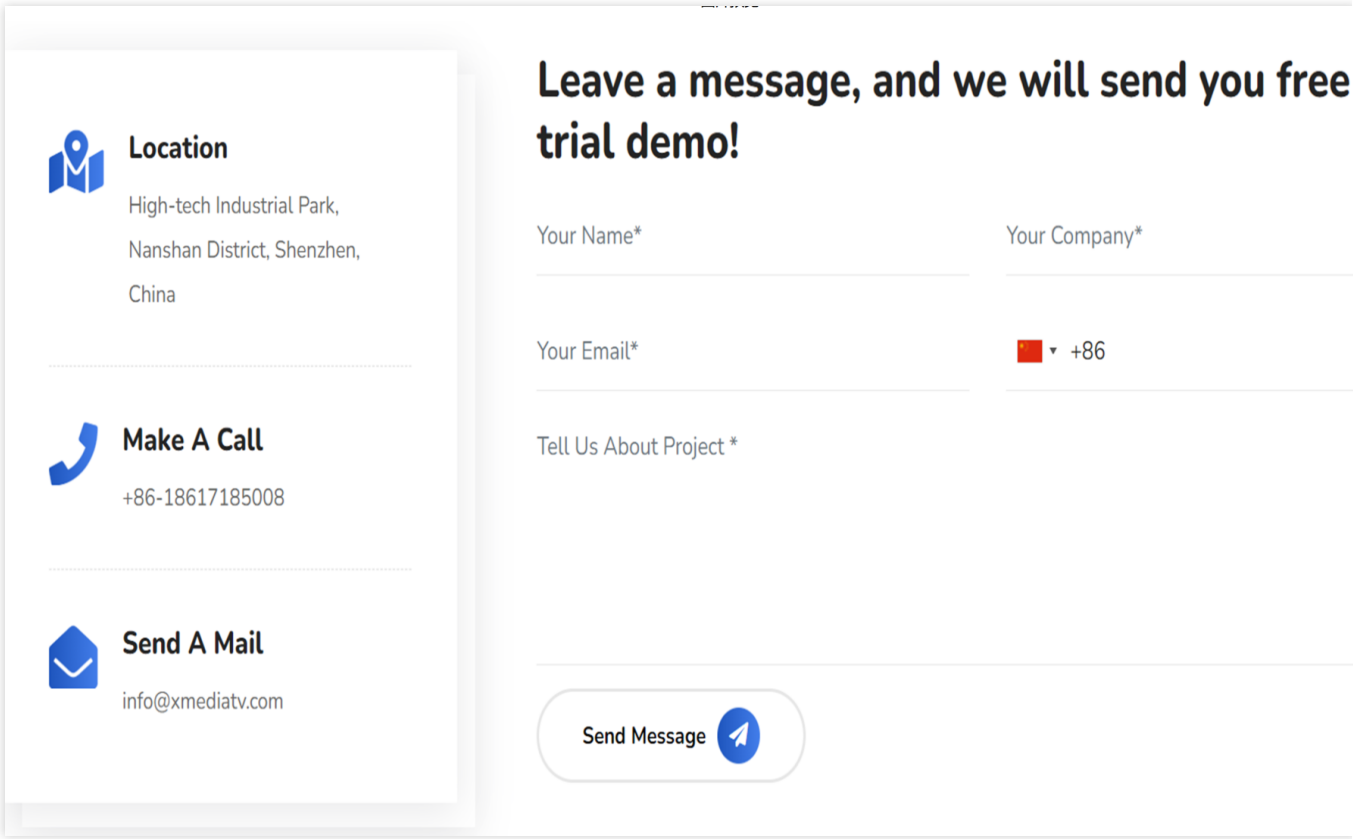
Configuring SDMC UID and Key Information

Last updated：2022-08-31 17:54:01

If you use the third-party (SDMC) DRM scheme, you need to obtain the user ID, secret ID, secret key, and FairPlay certificate URL from SDMC. This document shows you how to obtain such information.

Directions

1. If you don't have an SDMC account yet, sign up first.



2. Enter your information and click **Send Message**. You will receive an acknowledgement email from SDMC in a few hours, and the company's salespeople will contact you to confirm your information.

Thanks for your interesting in our solution. This is XMediaTV Business Team from SDMC, taking charge of OTT platform & Cloud business. Glad to work togehter with you.

Regarding to your project, we have the corresponding solution to meet your requirements and we also have delivered many global similiar projects successfully. May i know your whatsapp or Skype to discuss directly?

who we are:

SDMC is invested by Tencent, which is one of the biggest internet company in China. SDMC and Tencent joint together launched Cloud Online Video SaaS platform TXMedia to provide high quality video streaming service to customers.

Grow your video business in:

We glad to invite you to video and audio streaming service community, here we freely sharing and answering any streaming related questions, helping you empower your video business by content share, white-label platform, video revenue model, cutting-ed infos....come and talk to our experts in:
Whatsapp group → https://chat.whatsapp.com/
Facebook group → https://www.facebook.com/groups/
Linkedin group → https://www.linkedin.com/groups/

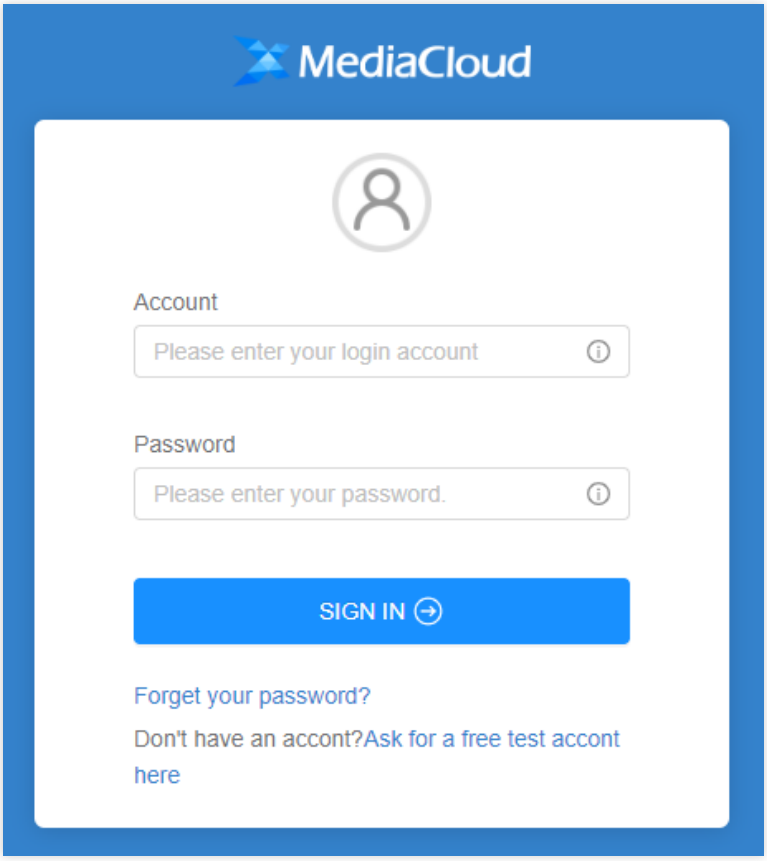
Know our streaming solution first!
SDMC XMediaTV Introduction V4.7 20210626.pdf

Please grab some time with me whenever you're ready - Schedule a Meeting

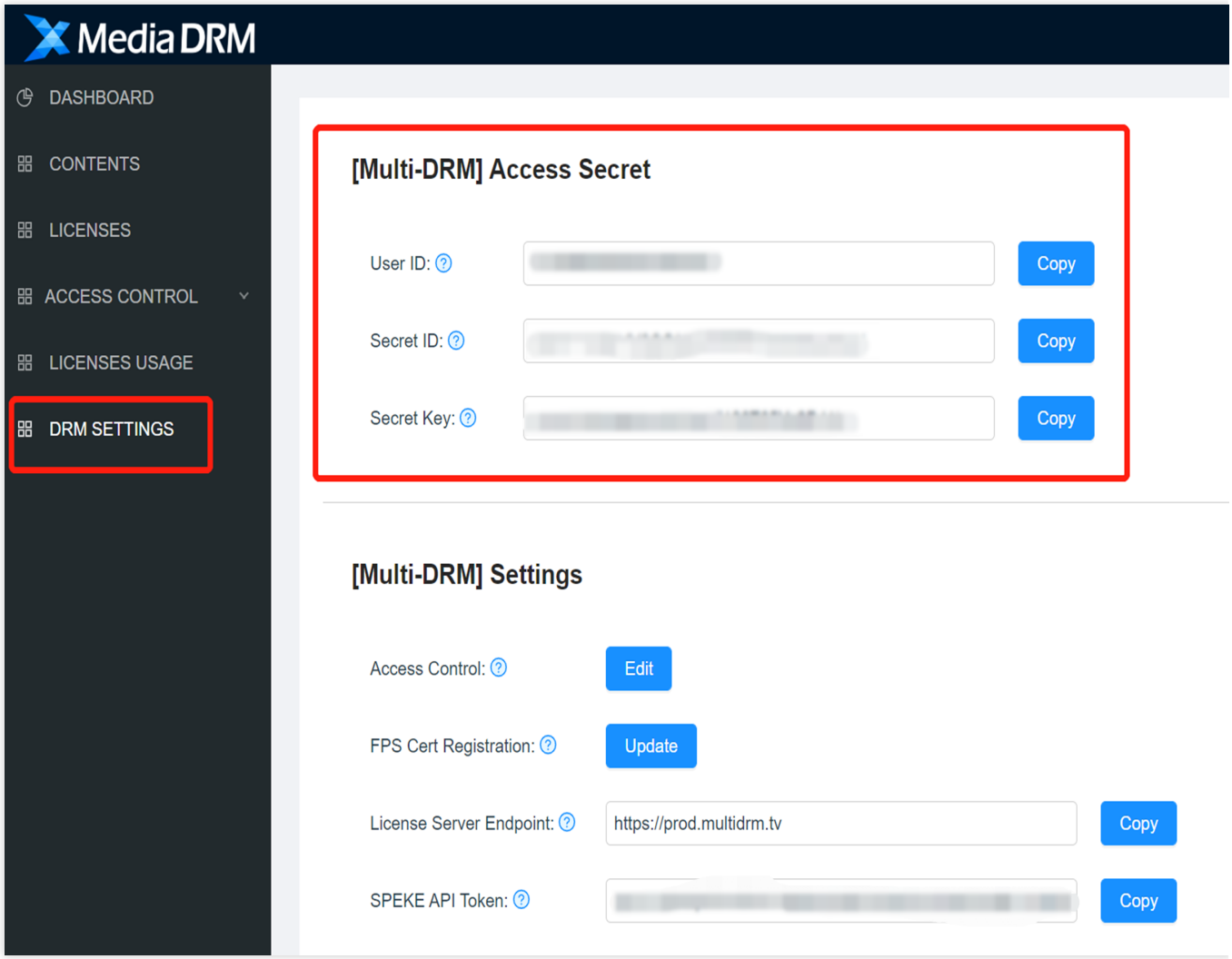
Best regards.

Xmedia Business Team

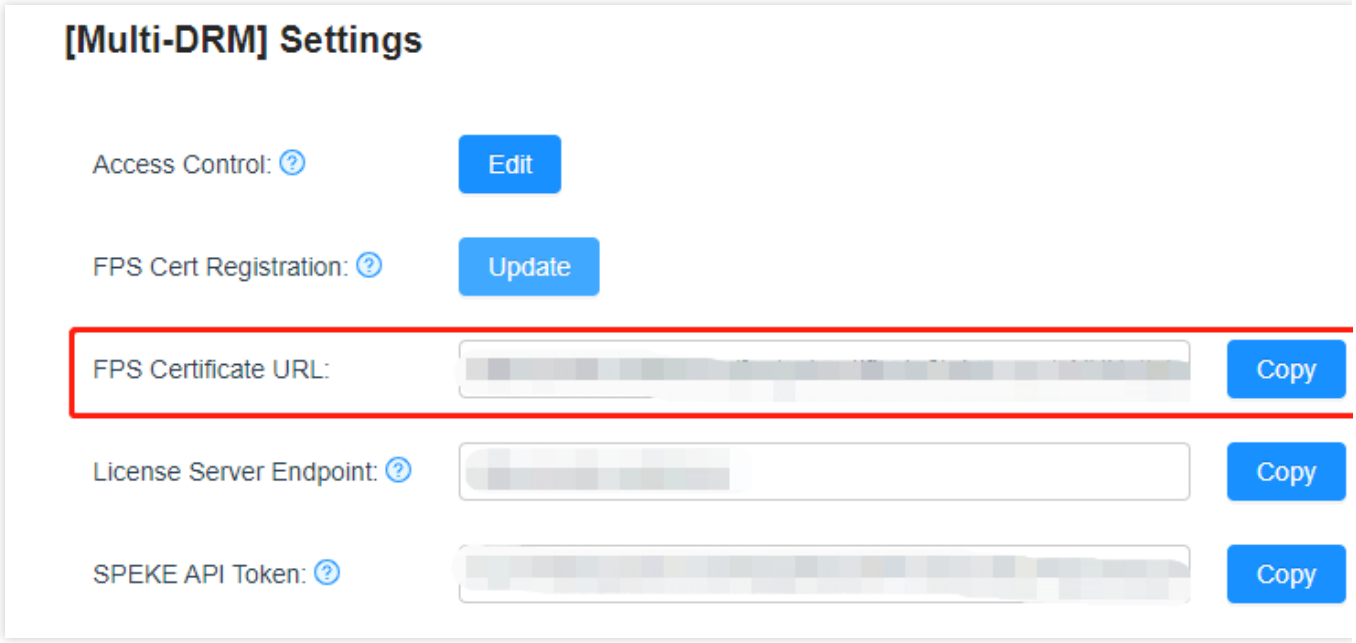
3. After reviewing your application, SDMC will email you the address of its DRM console and your initial password.
4. Log in to the SDMC DRM console with the account and password you received.



5. Click **DRM SETTINGS** to view the user ID, secret ID, and secret key.



6. Get the URL of your FairPlay certificate.

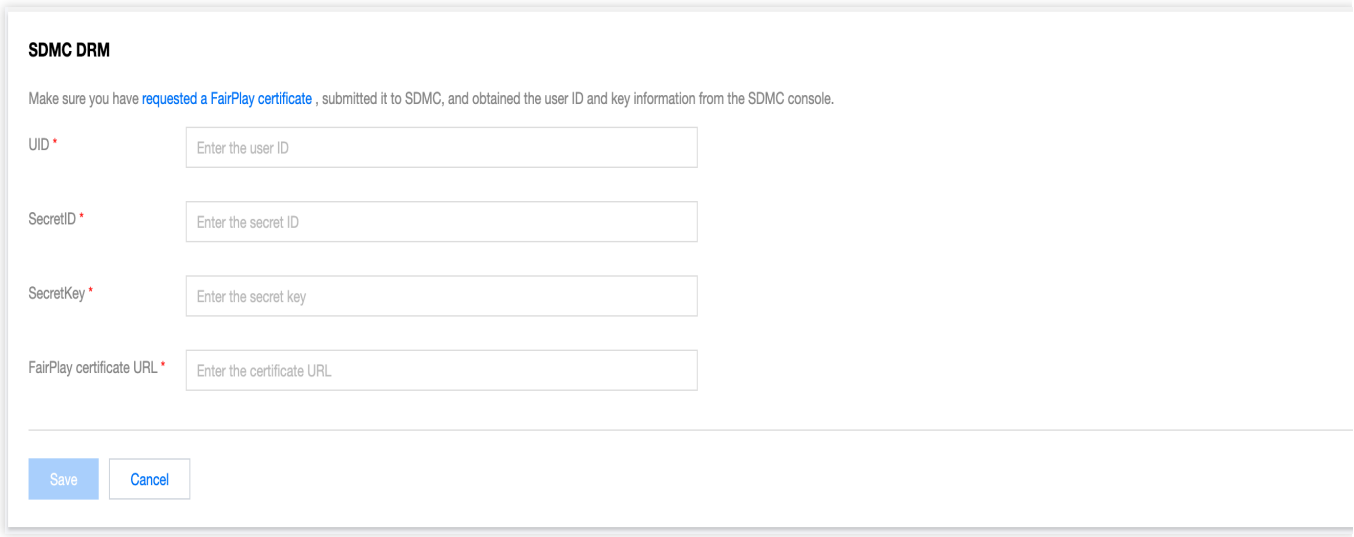


7. Log in to the VOD console.

8. Enter the information you obtained, including the user ID, secret ID, secret key, and FairPlay certificate URL.

Select **Media Processing > DRM Configuration** on the left sidebar and click **Edit** on the `SDMC DRM` .

Enter the information and click **Save**.



Summary

You have now configured the SDMC user ID and key information in the VOD console.

Note:

If you have any questions, please submit a ticket.

Playing DRM-Encrypted Videos

Last updated：2022-09-13 10:48:26

Overview

This document shows you how to encrypt videos using DRM solutions and play the encrypted videos with a 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 and complete identity verification.
2. Purchase VOD services. For details, see Billing Overview.
3. Go to the VOD console.

At this point, you have activated VOD.

Obtaining FairPlay certificate information

See Obtaining FairPlay Certificate Information.

Submitting the certificate information

See Submitting FairPlay Certificate Information to SDMC.

Configuring the SDMC UID and key information

See Configuring the SDMC UID and Key Information.

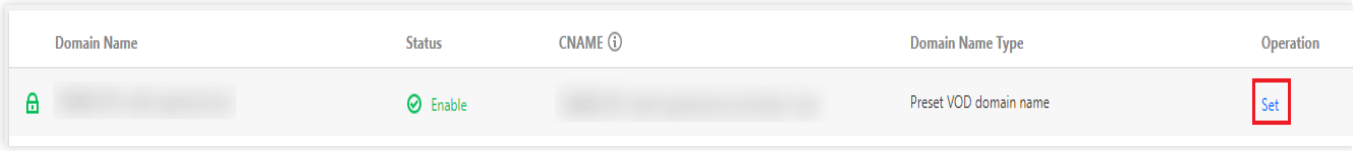
Step 1. Enable Hotlink Protection

The example below shows how to enable key hotlink protection for the default distribution domain under your account:

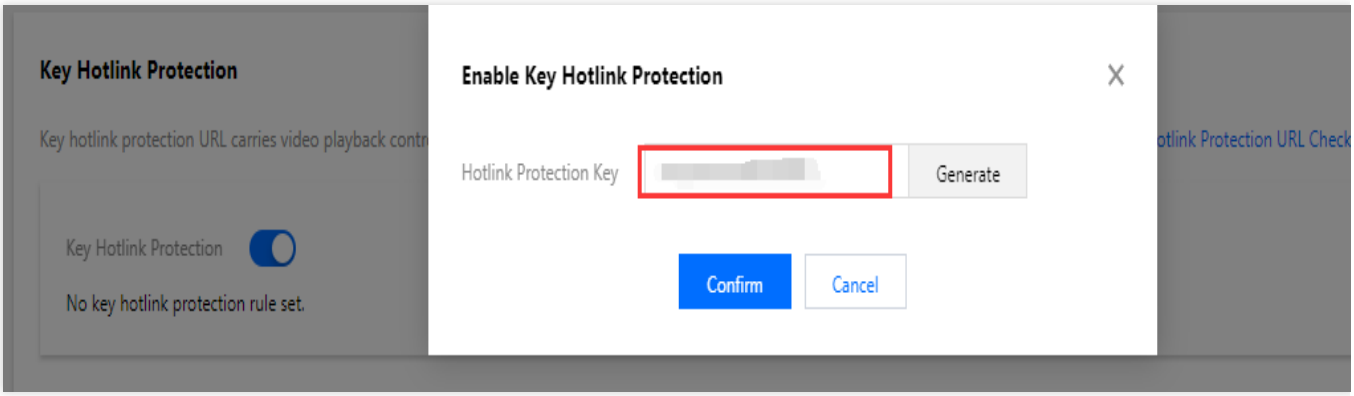
Note:

We do not recommend enabling hotlink protection for a domain name already in use. Doing so may cause failure to play existing videos.

1. Log in to the VOD console, select **Distribution and Playback** > Domain Name on the left sidebar. Find the default distribution domain, click **Set** on the right, and select the **Access Control** tab.

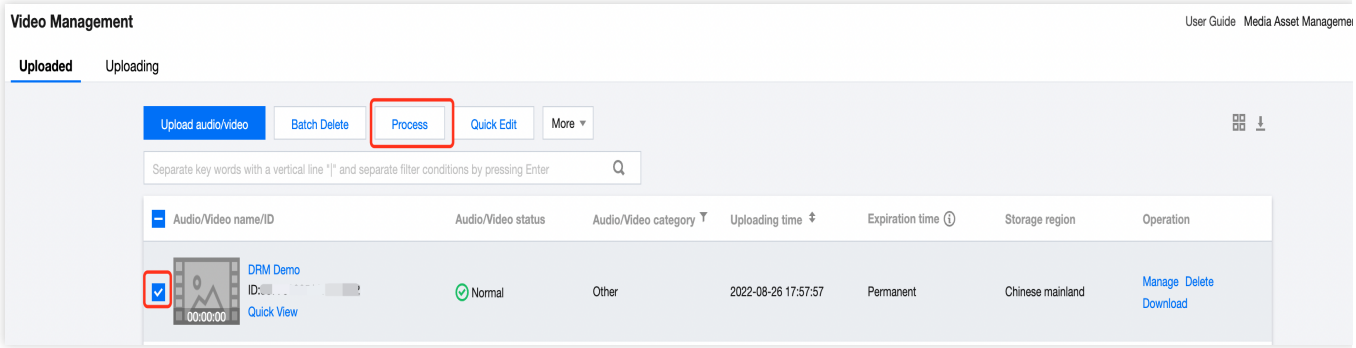


2. Toggle on **Key Hotlink Protection**. In the pop-up window, click **Generate** to generate a random key (suppose it is `vodtestkey`). Copy the key and click **Confirm**. You will use the key later to generate player signatures.



Step 2. Encrypt a Video

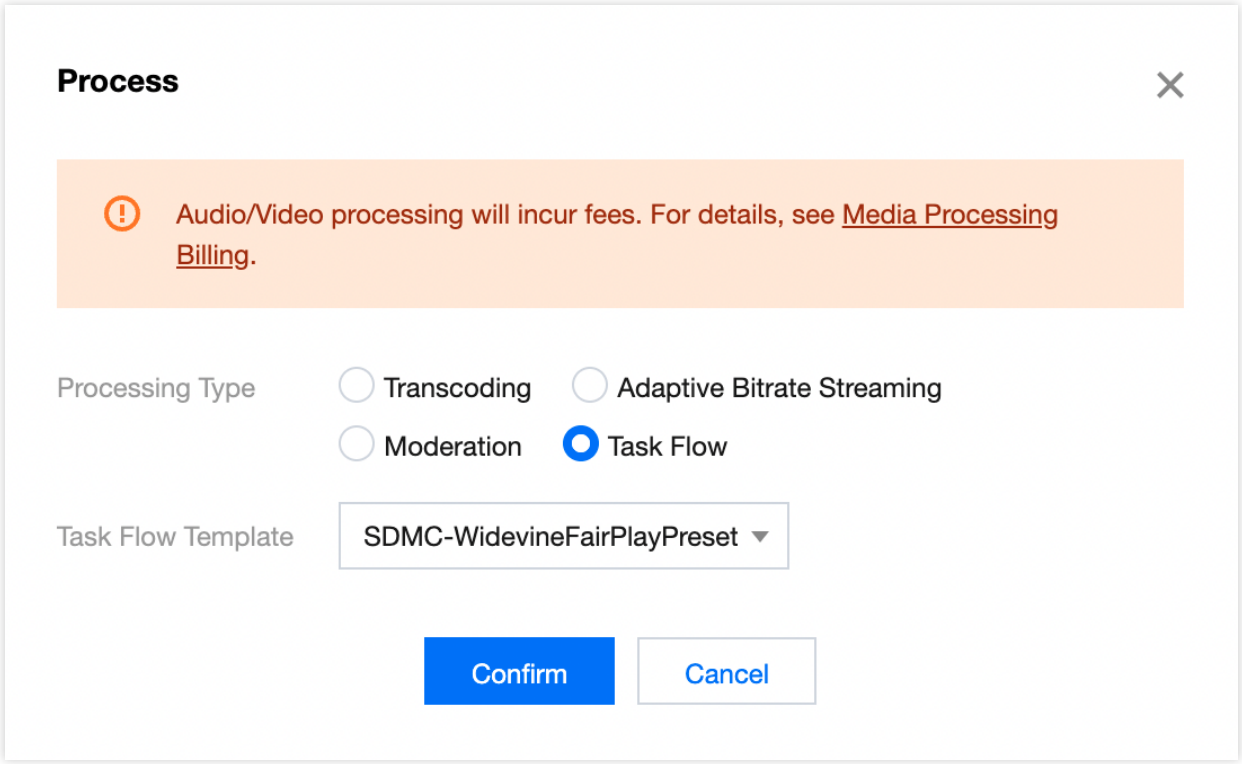
1. In the VOD console, select **Media Assets** > Video/Audio Management on the left sidebar, select the target video (in this example, the file ID of the video encrypted is `387702304941991610`), and click **Process**.



2. On the video processing page:

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

Select **SDMC-WidevineFairPlayPreset** as the **Task Flow Template**.

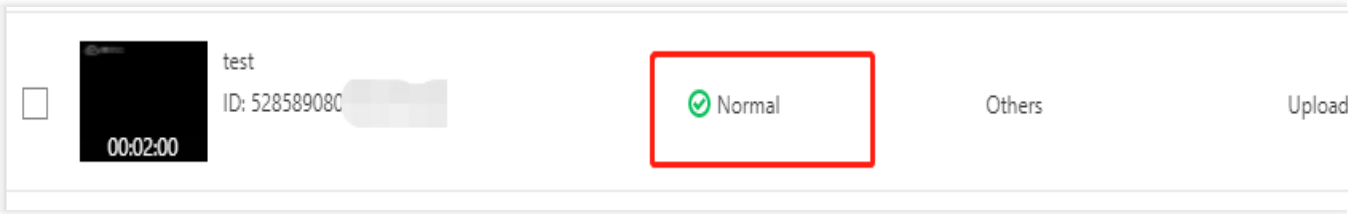


Note:

`SDMC-WidevineFairPlayPreset` is a preset task flow. It uses the adaptive bitrate streaming template 31 or 41, the time point screenshot template 10 (for thumbnail generation), and the image sprite template 10.

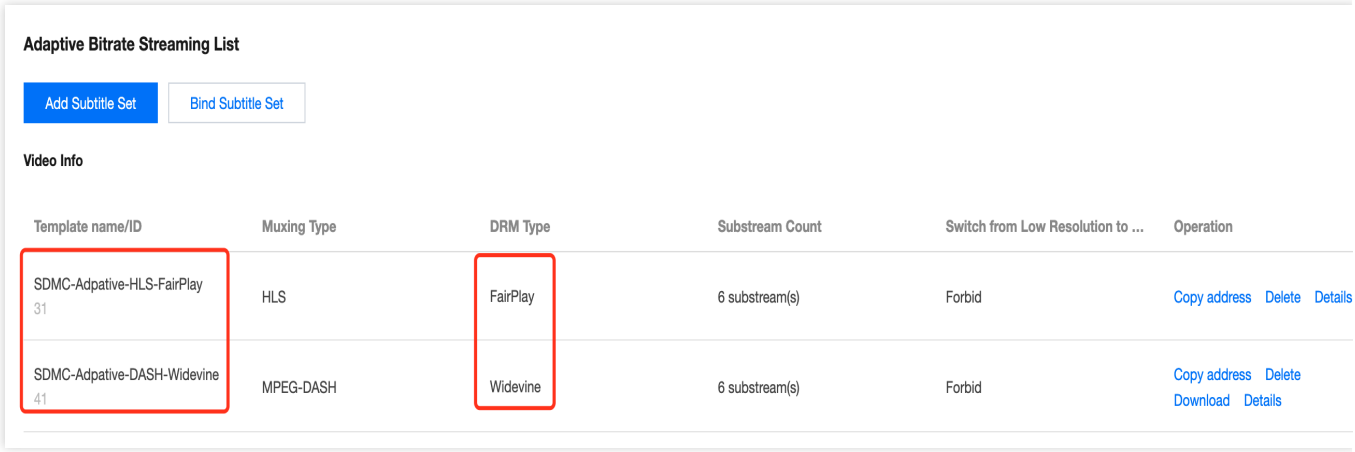
The adaptive bitrate streaming template 31 generates multi-bitrate streams encrypted by FairPlay, and the adaptive bitrate streaming template 41 generates multi-bitrate streams encrypted by Widevine.

3. Click **Confirm** and wait until the **Video Status** changes from "Processing" to "Normal", which indicates that video processing is completed.

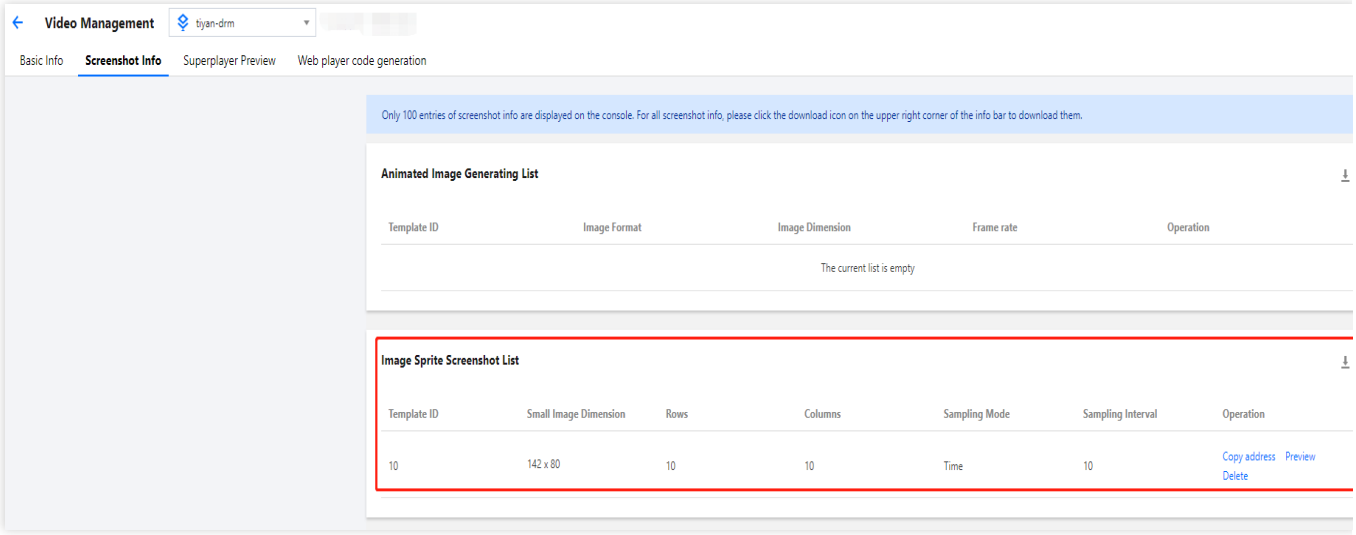


4. Click **Manage** in the **Operation** column of the video.

Under the **Basic Info** tab, you can view the thumbnail generated and outputs of adaptive bitrate streaming (template ID: 31 & 41).



Under the **Screenshot Info** tab, you can view the image sprite generated (template ID: 10).



Step 3. Generate a Player Signature

You will need the player signature to query past playback information. For directions on how to generate a player signature, see Superplayer Signature. For the example in this document, the payload for signature generation is as follows:

```
{
  "appId": 1500014561,
  "fileId": "387702304941991610",
  "currentTimeStamp": 1661163373,
  "expireTimeStamp": 2648557919,
  "pcfg": "SDMC-advanceDrmPreset"
}
```

The key generated for the example in this document is `vodtestkey`, and the player signature (`psign`) generated is as follows:

```
eyJhbGciOiJIUzI1NiJ9.eyJhcHBJZCI6MTUwMDAxNDU2MSwiZmlsZUlkIjoiMzg3NzAyMzA0OTQxOTkxNjEwIiwiY3VycmVudFRpbWVTdGFtcCI6MTY2MTE2MzM3M
ywiZXhwaXJlVGltZVN0YW1wIjoyNjQ4NTU3OTE5LCJwY2ZnIjoiU0RNQy1hZHZhbmNlRHJtUHJlc2V0In0.
BYdxHHEMH0isrta4ERmksGbfu4cLiwl7f1cu04XV890
```

Step 4. Play the DRM-Encrypted Video

Web

Using the VOD player

To play the DRM-encrypted video using the VOD player, just pass in the file ID of the video and your VOD account's AppID when initializing the player.

Step 1. Import files

Import the player's style file and script files into the webpage.

```
<link
href="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.4/tcplayer.min.cs
s" rel="stylesheet"/>

 <script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.4/libs/TXLivePlaye
r-1.2.3.min.js"></script>

 <script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.4/libs/hls.min.1.1
.5.js"></script>

 <script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.4/libs/flv.min.1.6
.3.js"></script>

 <script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.4/libs/dash.all.mi
n.4.4.1.js"></script>

 <script
src="https://web.sdk.qcloud.com/player/tcplayer/release/v4.5.4/tcplayer.v4.5.4.
min.js"></script>
```

Step 2. Add a player container

Add a player container to wherever you want to display the player:

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

Note:

You can customize the container ID as well as the height and width of the container.

Step 3. Add the initialization code

Add the following script to your page initialization code and pass in the required initialization parameters(including the Player Signature `psign` generated in Step 3):

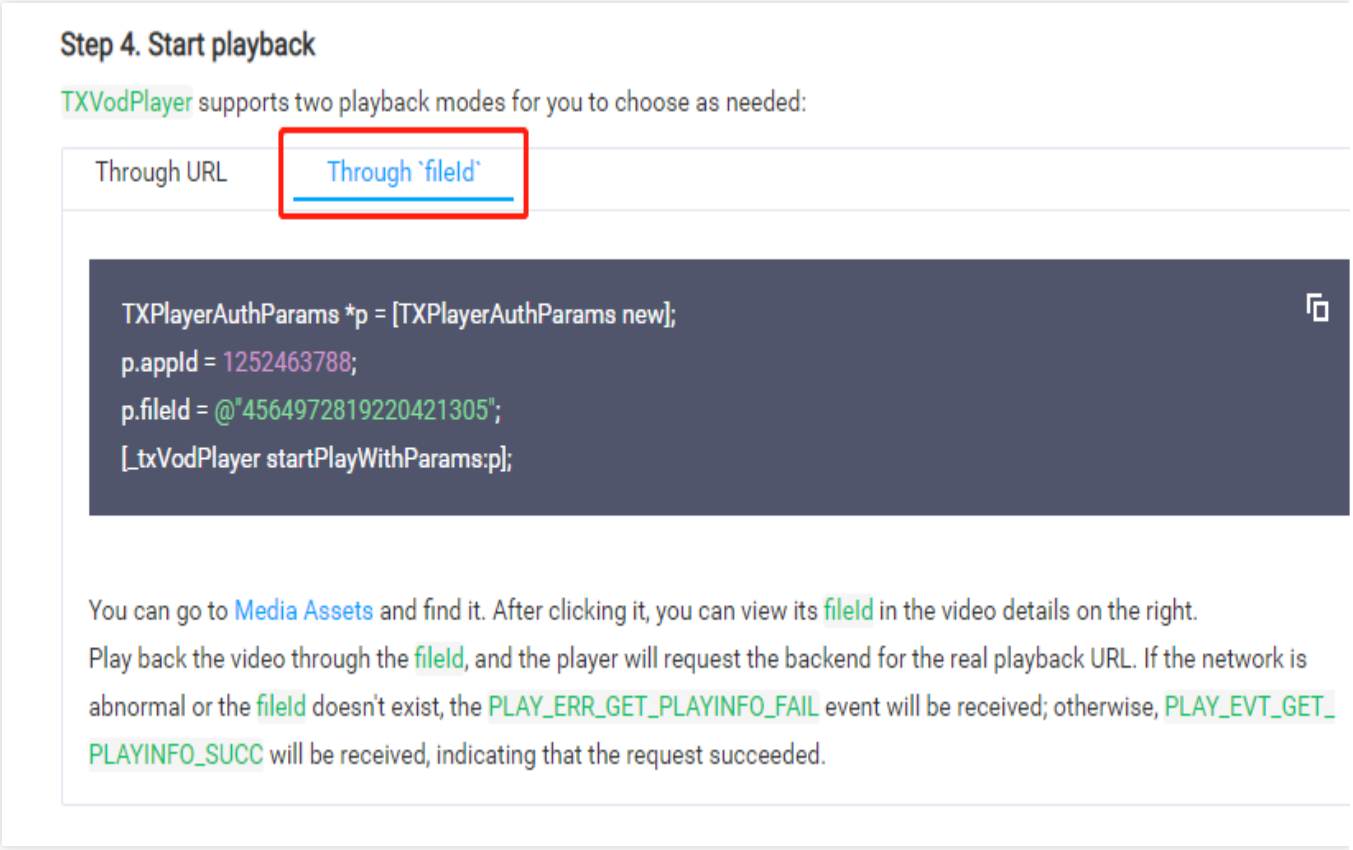
```
var player = TCPlayer('player-container-id', {
    appID: '1500014561' // The appID of your VOD account (required).
    fileID: '387702304941991610', // The file ID of the video to play
(required).
    psign:
'eyJhbGciOiJIUzI1NiJ9.eyJhcHBJZCI6MTUwMDAxNDU2MSwiZmlsZUlkIjoiMzg3NzAyMzA0OTQxO
TkxNjEwIiwiY3VycmVudFRpbWVTdGFtcCI6MTY2MTE2MzM3MywiZXhwaXJlVGltZVN0YW1wIjoyNjQ4
NTU3OTE5LCJwY2ZnIjoiU0RNQy1hZHZhbmNlRHJtUHJlc2V0In0.BYdxHHEMH0isrta4ERmksGbfu4c
Liwl7f1cu04XV890',
    // For other parameters, see
https://intl.cloud.tencent.com/document/product/266/39105
});
```

iOS

To play the DRM-encrypted video on iOS, refer to Integration Guide (Through `FileId`). You need to use the player signature (`psign`) generated in Step 3 (Generate a Player Signature).

Note:

Please submit a ticket for the player SDK that supports DRM.

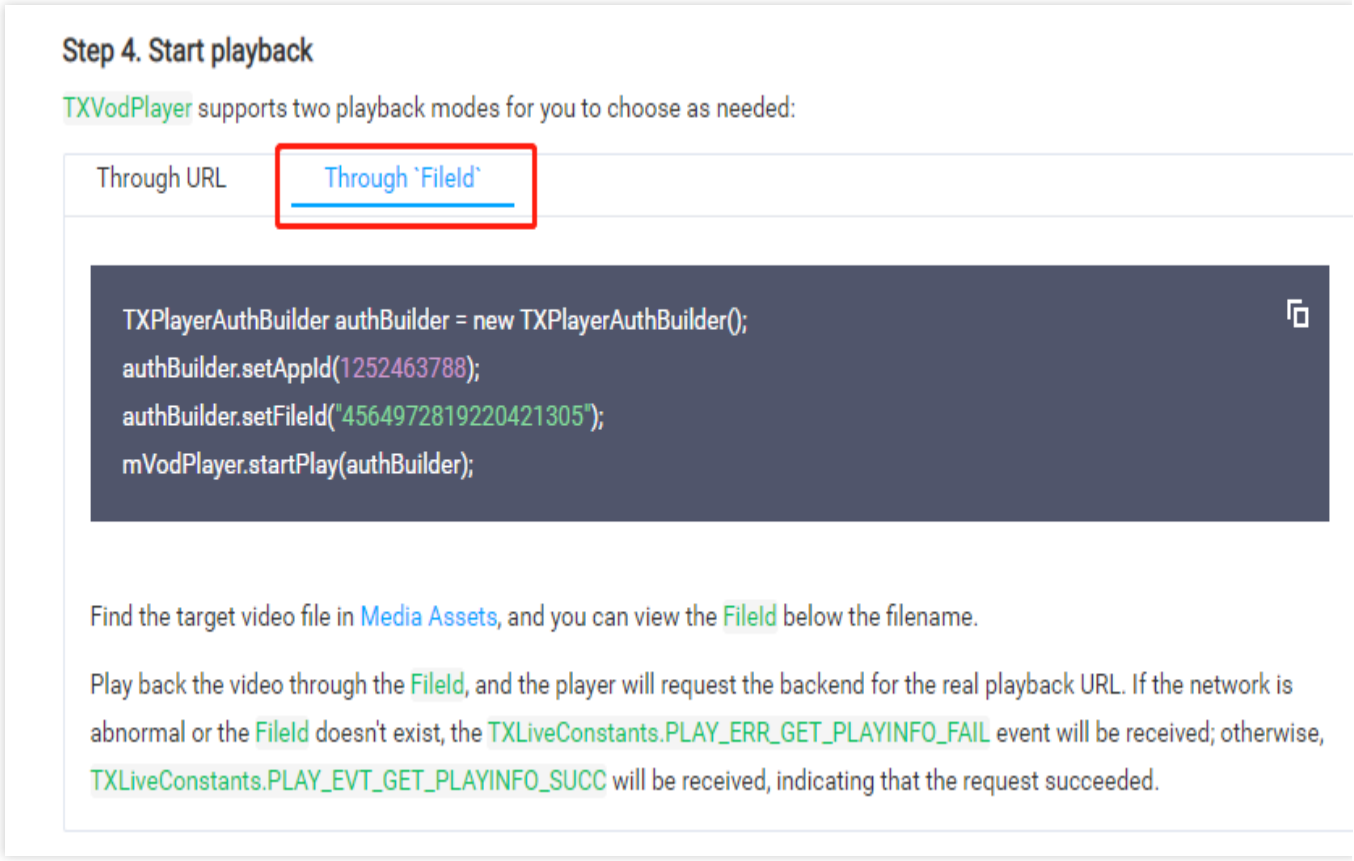


Android

To play the DRM-encrypted video on Android, refer to Integration Guide (Through `FileId`). You need to use the player signature (`psign`) generated in Step 3 (Generate a Player Signature).

Note:

Please submit a ticket for the player SDK that supports DRM.



Summary

Now, you have learned how to encrypt videos using DRM solutions and play the encrypted videos in a player.

Note:

If you have any questions, please submit a ticket.

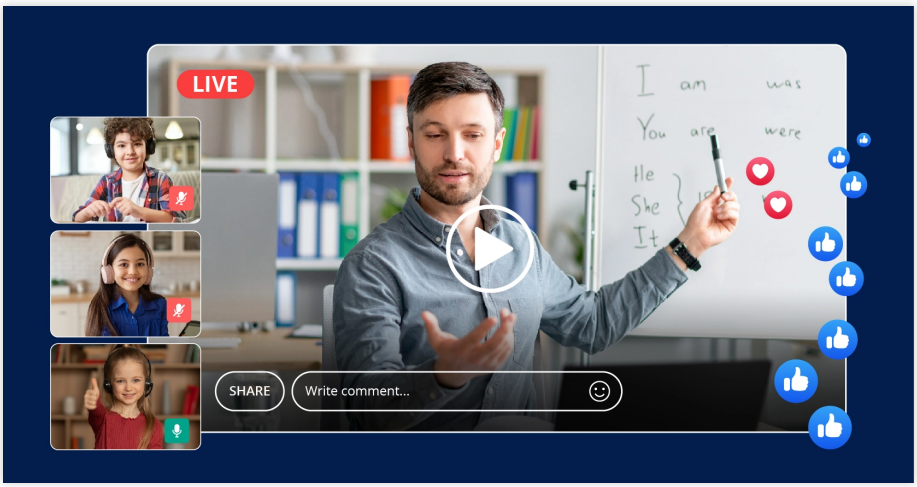
Broadcast Channel

Last updated：2025-02-24 14:57:36

VOD's Virtual Live Streaming and Broadcast Channels allow users to assemble one or multiple on-demand videos for playback in a live streaming format. This capability can be applied to the following three scenarios:

Scenario	Description
Virtual Live Streaming	Virtual live classes: Teachers pre-record course content and play it for students at scheduled times. Live commerce product showcases: Pre-record product promotion content creating a virtual live shopping experience.
Video Program Channels	Broadcast OTT channels: Preset TV program schedules in advance, enabling users to watch content in a traditional TV channel format. Video platform screening room: Allow users to join a virtual "screening room" to watch the same video simultaneously while interacting in real time.

Virtual Live Streaming



Assume that the course originally streamed on CSS has been recorded to VOD (in HLS format) and is scheduled to be played back as a simulated live lesson at 20:00 Beijing Time on December 1, 2024.

The operation steps are as follow:

1. Call the CreateRoundPlay API.

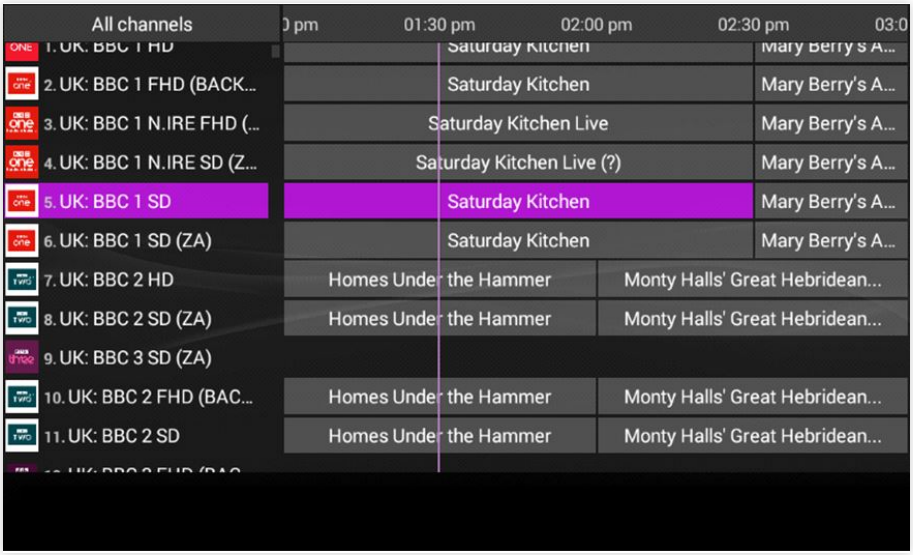
In the RoundPlaylist.N parameter, enter the course video ID in FileId, and set AudioVideoType to Original.

Set StartTime to 2024-12-01T20:00+08:00 (Beijing time).
Specify PlaybackMode as Linear.
After the API call, retrieve the playlist URL from the response's Url parameter and obtain the playlist ID from RoundPlayId.
2. Starting at 20:00 on December 1, 2024 (Beijing time), provide the playlist URL to players. The players will begin streaming the pre-recorded course in a live broadcast format.
3. After playback concludes, call the ModifyRoundPlay API by setting RoundPlayId to the channel ID and updating Status to Disabled to stop the playlist. Call the DeleteRoundPlay API to delete the playlist.

Note：

China's online travel service provider, Ctrip, has utilized the broadcast channel feature to implement interactive virtual live rooms, enabling low-cost promotion of travel products.

Video Program Channels



Assume that a program channel needs to be set up, which will start broadcasting at 10:00 Beijing Time on December 1, 2024, and automatically stop after playing 5 programs.

The operation steps are as follow:

1. Upload the 5 videos to VOD and transcode them into HLS format using the transcoding template with ID 100240 (refer to the Transcoding Task Initiation section for transcoding guidelines).
2. Call the CreateRoundPlay API.

In the RoundPlaylist.N parameter, specify the programs to be played in the channel, enter the uploaded video IDs in FileId, set AudioVideoType to Transcode, and set Definition to 100240 (transcoding template ID).

Set StartTime to 2024-12-01T10:00+08:00 (Beijing time).
Specify PlaybackMode as Linear.
After the API call, retrieve the playlist URL from the response's Url parameter and obtain the playlist ID from RoundPlayId.
3. Starting at 10:00 on December 1, 2024 (Beijing time), provide the playlist URL to players. The players will sequentially play the 5 programs and automatically stop after the last program finishes.
4. After playback concludes, call the ModifyRoundPlay API by setting RoundPlayId to the channel ID and updating Status to Disabled to stop the playlist. Call the DeleteRoundPlay API to delete the playlist.

CAM
Overview

Last updated：2022-04-01 10:10:50

Note:

This document describes access management for VOD. For information about the access management of other Tencent Cloud products, see CAM-Enabled Products.

VOD has been connected to Tencent Cloud Cloud Access Management (CAM). You can grant specified VOD permissions to sub-accounts as needed. The VOD access control feature can be used directly once the VOD service is activated.

This document assumes that you already have some knowledge of Tencent Cloud CAM and VOD's subapplication system. The main concepts involved in this document include:

CAM: user type, API key, policy, and policy syntax

VOD: subapplication

Use Cases

The typical use cases of VOD access control are as follows:

Permission isolation at Tencent Cloud product level

Among the various departments using Tencent Cloud in an organization, department A takes charge of the VOD service. Staff of department A need permission to access VOD but not other Tencent Cloud products. To this end, you can create a sub-user and only grant it VOD-related permissions, and then provide it to department A.

Permission isolation at VOD subapplication level

When multiple businesses in an organization are using VOD, isolation is generally needed. Isolation involves resource isolation and permission isolation, of which the former is enabled by VOD's subapplication system and the latter implemented by VOD access control. In this case, sub-users can be created for each business and granted permission to the corresponding subapplications, so that each business can only access the specified subapplication.

Permission isolation at VOD operation level

Product operations staff of a business using VOD in an organization need to access the VOD Console to get statistics (e.g., geographical distribution of traffic and number of playbacks), but they should be forbidden to perform sensitive operations (e.g., deleting files or disabling domain names) so as to protect the business against any faulty operations. To meet such needs, you can create a custom policy that has permissions to log in to the VOD Console and call statistics APIs, create a sub-user and bind it to that policy, and then deliver the sub-user information to the product operations staff.

Resource Granularity and Operation Granularity

The core feature of CAM is to **allow or forbid an account to perform some operations or manipulate some resources**. For VOD, the resource granularity is subapplication, and the operation granularity is server API.

Limits

VOD access control supports authorization at subapplication level but not at finer-grained resource level (e.g., media files and domain names).

APIs Supporting Authorization at Resource Level

VOD access control supports authorization at resource level. All its APIs, except those with special limits, support authorization at resource level. Please see below for details.

List of APIs not supporting authorization at resource level

API Name	Feature	Description
DescribeSubAppIds	Queries the list of subapplications	All subusers have permission to call this API with no authorization required, and subapplications do not need to be specified.
ModifySubAppIdStatus	Modifies the status of a subapplication	This API can disable specified subapplications, which is highly risky. Therefore, it is available to only subusers with full VOD permissions (i.e., `QcloudVODFullAccess` as described in Preset Policies). Subusers that are granted write permissions to certain subapplications but not `QcloudVODFullAccess` cannot call this API.

List of APIs supporting authorization at resource level

Except those in the above list, all APIs outlined in API Overview support authorization at resource level. In policy syntax, resource descriptions for these APIs are all in the format of `qcs::vod::uin/$uin:subAppId/$subAppId`.

Preset Policy

Last updated：2022-05-31 10:48:56

Note:

This document describes the access management feature of **VOD**. For more information on access management for other Tencent Cloud services, please see CAM-Enabled Products.

Access management is essentially to bind sub-accounts to policies or grant policies to sub-accounts. You can use preset policies directly in the console to implement some simple authorization operations. For more complicated authorization operations, please see Custom Policy.

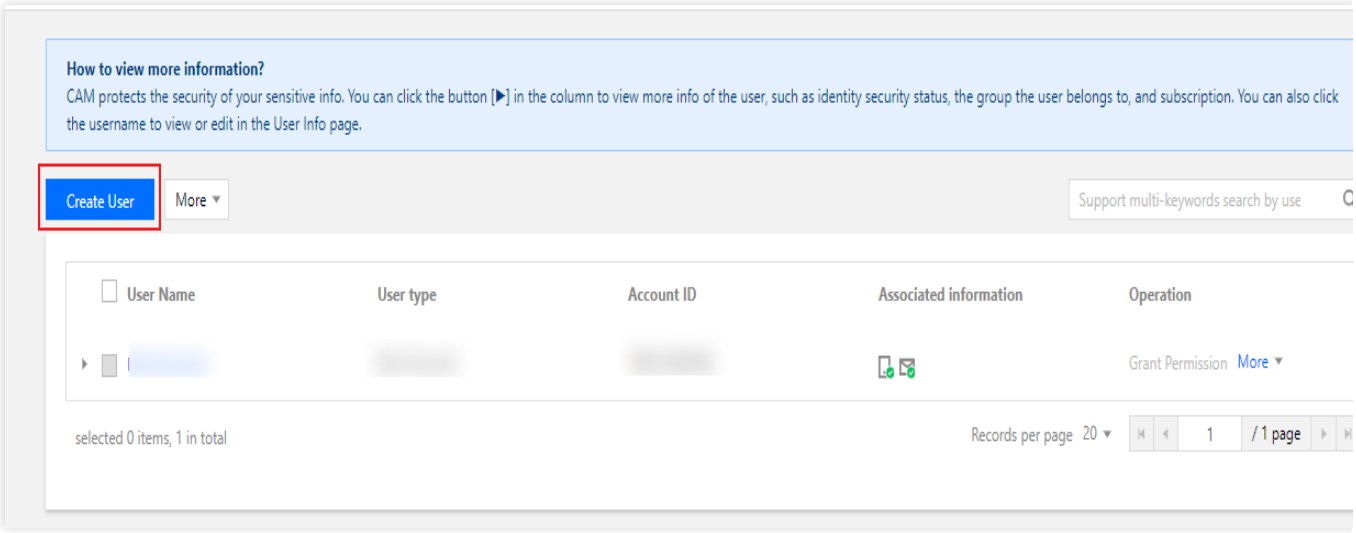
Currently, VOD provides the following preset policies:

Policy Name	Description
QcloudVODFullAccess	Full access to VOD
QcloudVODReadonlyAccess	Read-only access to VOD

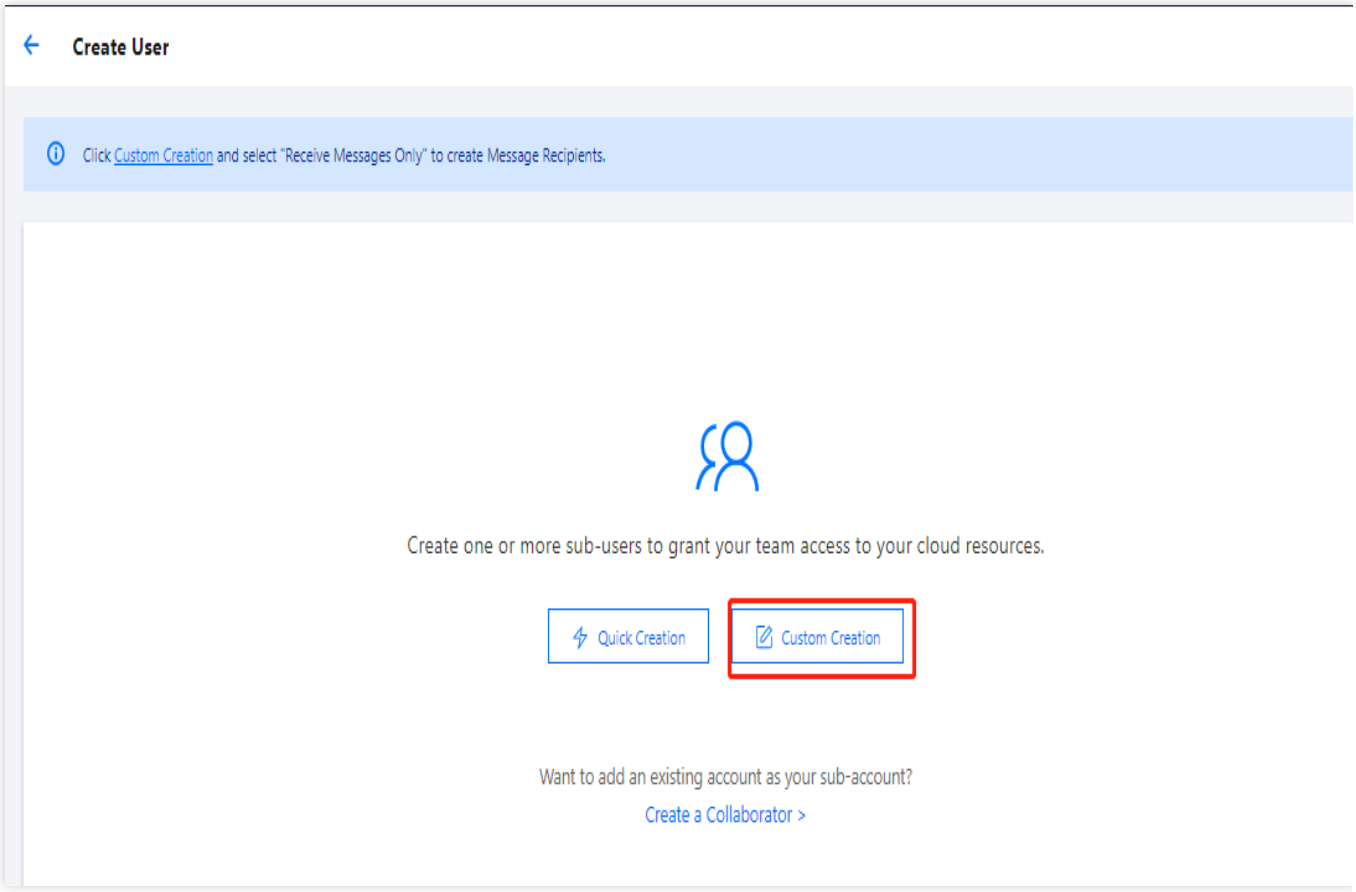
Preset Policy Use Cases

Creating subuser with full access to VOD

1. Access the User List page in the CAM console as a root account and click **Create User**.



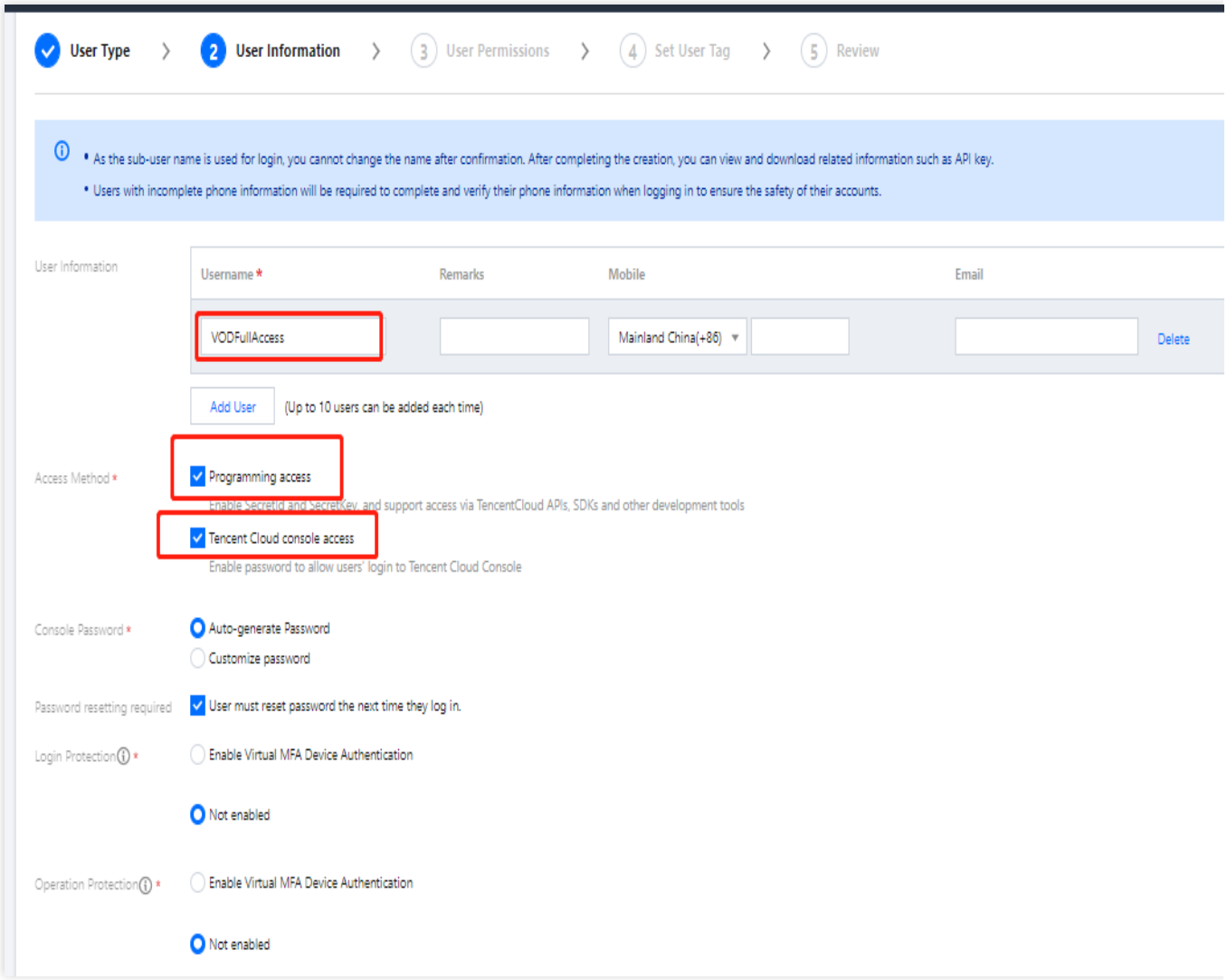
2. On the **Create User** page, click **Custom Creation**.



3. Click **Next** and enter the user information.

Enter the username, select **Programming Access** and **Tencent Cloud Console Access**, and configure other options as needed.

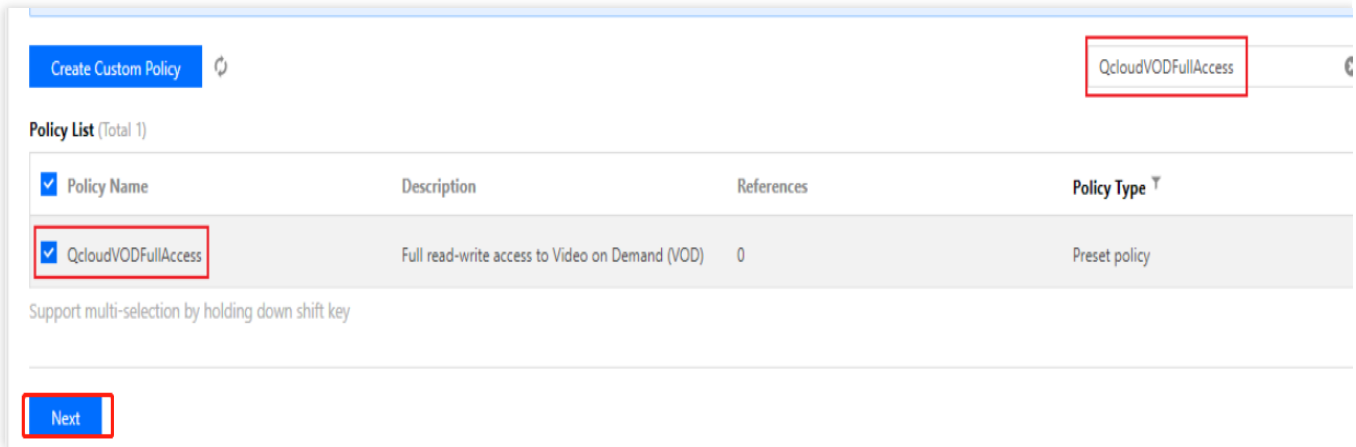
Click **Next** and complete authentication as prompted.



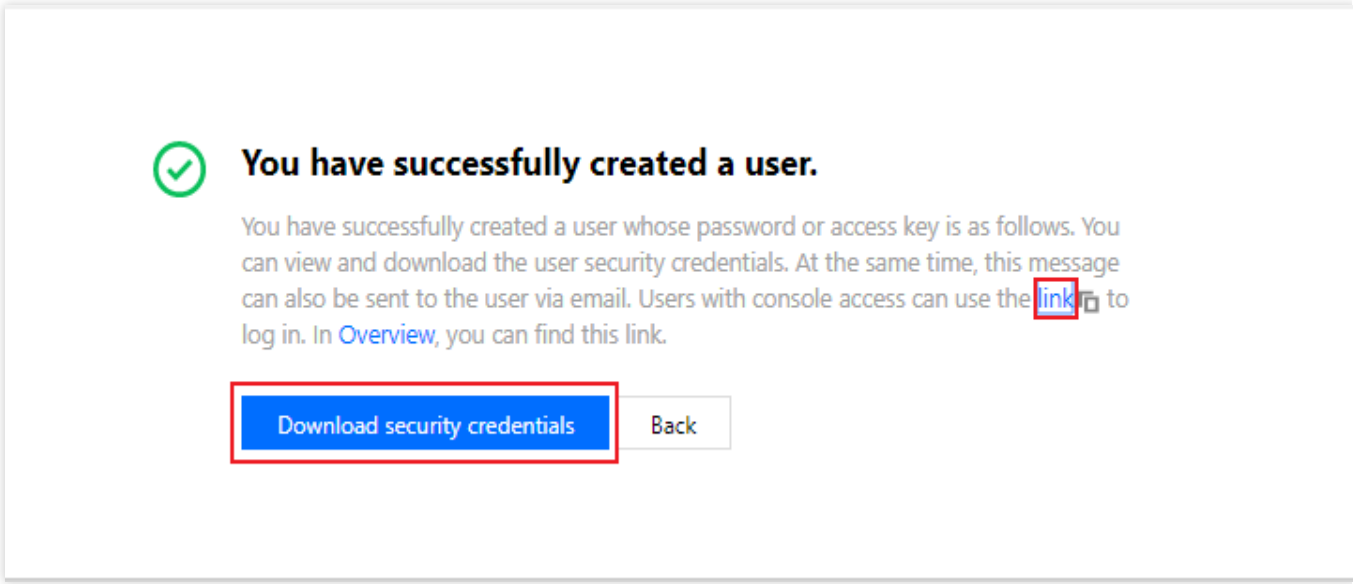
4. Set user permissions.

Search for and select the preset policy `QcloudVODFullAccess` .

Click **Next**.



5. Click **Complete** in the "Review Info and Permission" column. After the user is created successfully, download the login link and security credentials as shown below and keep them safe.



Information	Source	Function	Storage Required
Login link	Copy on the page	Makes it easier to log in to the console without having to enter the root account	No
Username	Security credential file in CSV format	Required for console login	Yes
Password	Security credential file in CSV format	Required for console login	Yes
SecretId	Security credential file in CSV format	Required for server API call. For more information, please see Access Key	Yes
SecretKey	Security credential file in CSV format	Required for server API call. For more information, please see Access Key	Yes

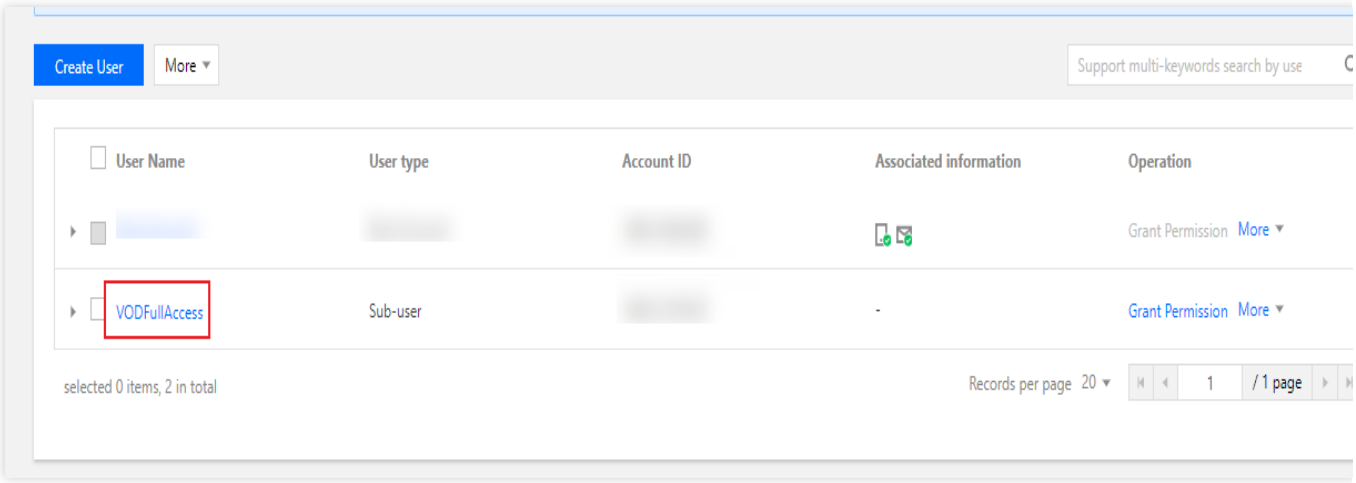
With the above login link and security credentials, you can use this subuser to perform all operations in VOD (e.g., accessing the VOD console and calling VOD server APIs).

Note:

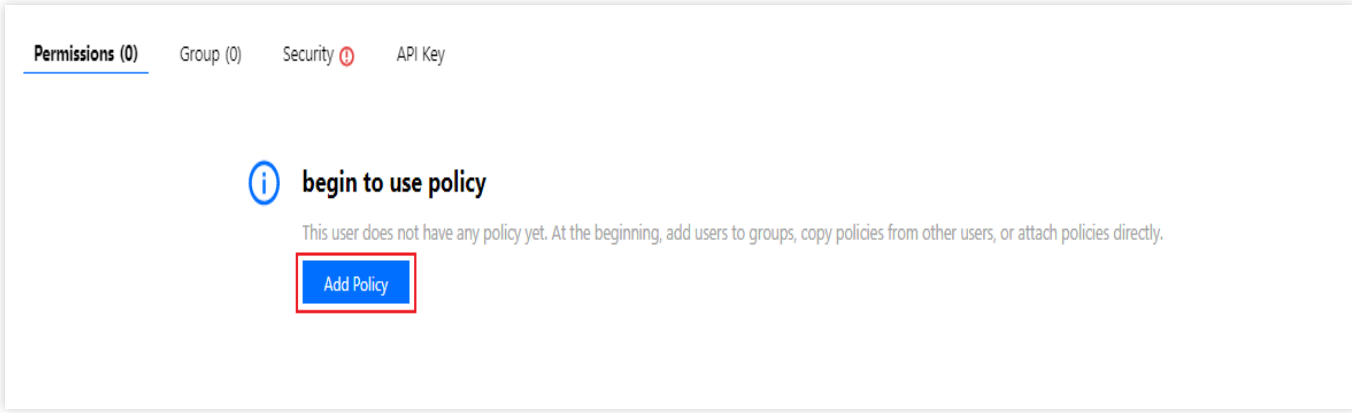
For more information on how to create a subuser, please see the Creating Subusers document of CAM.

Granting full permissions of VOD to existing subusers

1. Access the **User List** in the CAM console as a root account and click the target sub-account.



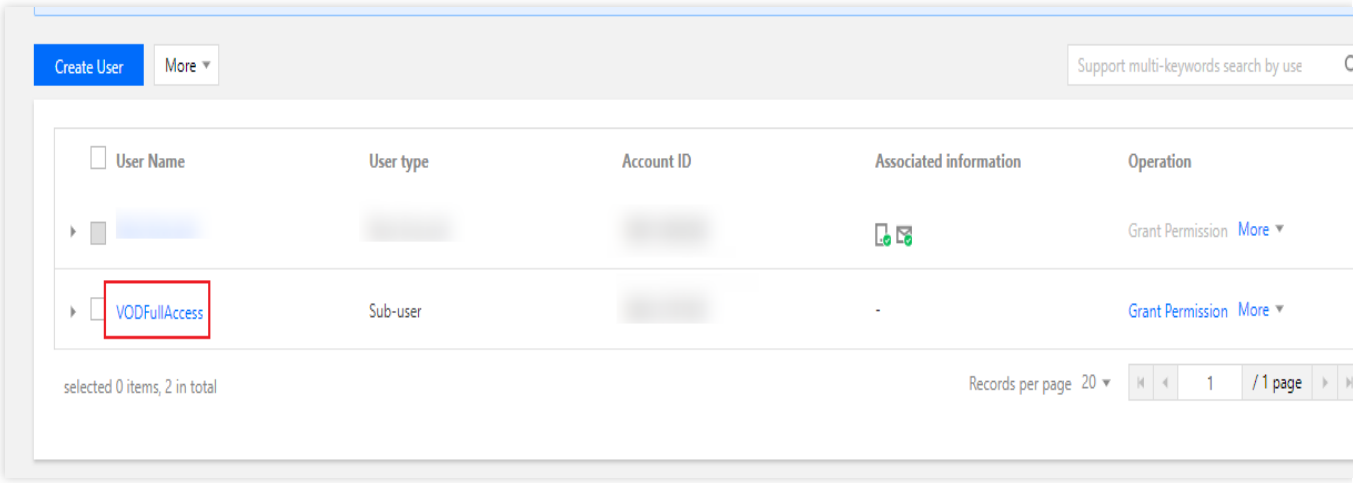
2. Click **Add Policy** in the **Permission** column on the **User Details** page, as shown below (in practice, the information displayed on this page may vary by existing permissions of the sub-account. If the permission of a sub-account is not empty, please click **Associate Policy**).



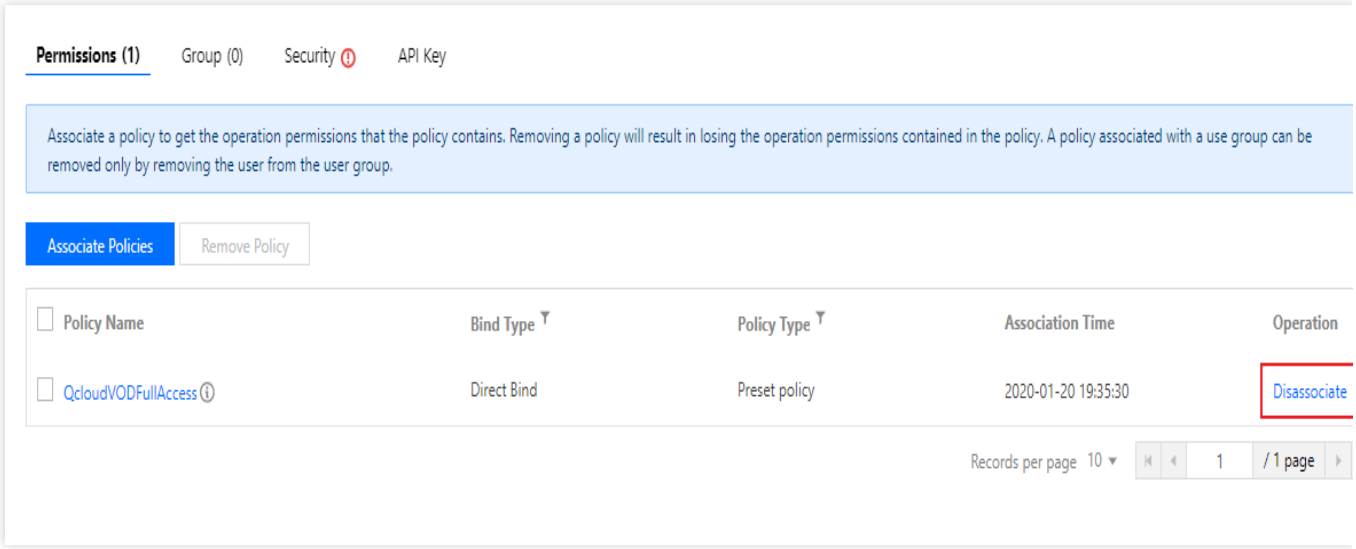
3. Click **Select Policy from Policy List to Associate**, search for and check the preset policy `QcloudVODFullAccess` , and complete authorization as prompted.

Revoking subuser's full access to VOD

1. Access the **User List** in the CAM console as a root account and click the target sub-account.



2. Find the preset policy `QcloudVODFullAccess` in the **Permission** column on the **User Details** page, click **Unassociate** on the right, and complete deauthorization as prompted.



Custom Policy

Last updated：2022-05-31 11:04:43

Note:

This document describes the access management feature of **VOD**. For more information on access management for other Tencent Cloud services, please see CAM-Enabled Products.

It is convenient to use a preset policy in CAM to implement authorization, but its granularity of permission control is coarse and cannot be refined to the subapplication and API levels. If you require fine-grained permissions control, you need to create custom policies.

Custom Policy Creation Method

There are multiple ways to create a custom policy. The table below shows a comparison of various methods. For detailed directions, please see further below.

Creation Entry	Creation Method	Effect	Resource	Action	Flexibility	Difficulty
Console	Policy builder	Manual selection	Syntax description	Manual selection	Medium	Medium
Console	Policy syntax	Syntax description	Syntax description	Syntax description	High	High
Server API	CreatePolicy	Syntax description	Syntax description	Syntax description	High	High

Note:

VOD does not support creating custom policies by product feature.

Manual selection means that you can select an object from the candidate list displayed in the console, while **syntax description** means that you can describe objects through policy syntax.

Policy Syntax Description for Resource

As mentioned above, the resource granularity of permission control in VOD is subapplication. The subapplication description in policy syntax follows the CAM rules. In the example below, the developer's root account ID is 12345678, `APPID` is 1250000001 (which is equivalent to the primary application ID), and the developer has created two VOD subapplications with IDs of 1400000001 and 1400000002 respectively.

Policy syntax description for all VOD resources

```
"resource": [
  "qcs::vod::uin/12345678:subAppId/*"
]
```

Policy syntax description for the primary application

```
"resource": [
  "qcs::vod::uin/12345678:subAppId/1250000001"
]
```

Policy syntax description for a single subapplication

```
"resource": [
  "qcs::vod::uin/12345678:subAppId/1400000001"
]
```

Policy syntax description for the primary application and a single subapplication

```
"resource": [
  "qcs::vod::uin/12345678:subAppId/1250000001",
  "qcs::vod::uin/12345678:subAppId/1400000001"
]
```

Policy Syntax Description for Action

As mentioned above, the action granularity of permission control in VOD is server API. Server APIs such as `DescribeMediaInfos` and `DescribeAllClass` are used as examples below.

Policy syntax description for all VOD server APIs

```
"action": [
  "name/vod:*"
]
```

Policy syntax description for a single server API

```
"action": [
  "name/vod:DescribeMediaInfos"
]
```

Policy syntax description for multiple server APIs

```
"action": [
  "name/vod:DescribeMediaInfos",
```

```
    "name/vod:DescribeAllClass"
]
```

Custom Policy Use Cases

Using policy builder

In the example below, we will create a custom policy, which allows all actions except the server API `ProcessMedia` to be performed on VOD subapplication 1400000001.

1. Access the **Policy** page in the CAM Console as a root account and click **Create Custom Policy**.
2. Select **Create by Policy Generator** to enter the policy creation page.
3. Select services and actions.

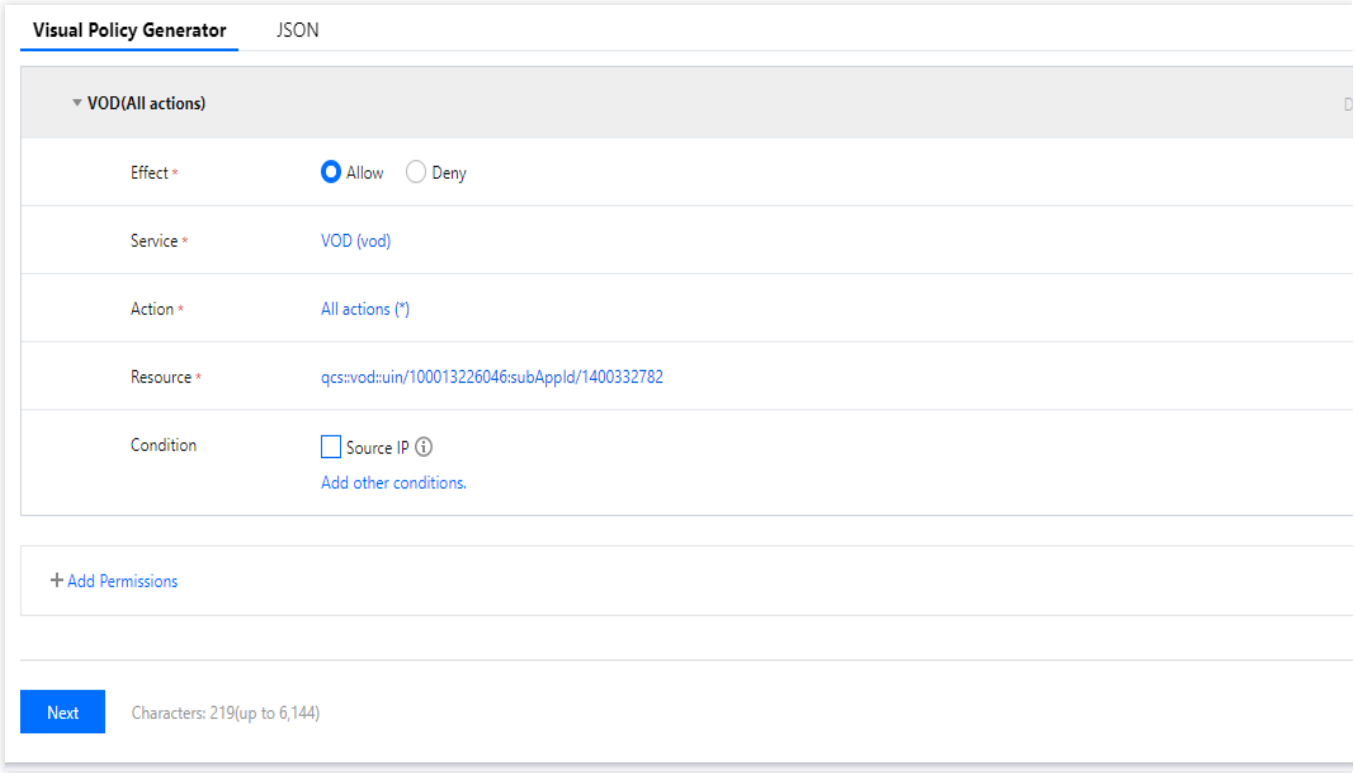
Select **Allow** for **Effect**.

Select **VOD** for **Service**.

Check all items for **Action**.

Enter `qcs::vod::uin/12345678:subAppId/1400000001` for **Resource** according to the syntax description for resource.

The **Condition** configuration item does not need to be configured.



4. Click **Next** and rename the policy name as needed (or leave it unchanged).

5. Click **Complete** to create the custom policy. Subsequently, this policy can be granted to subusers in the same way as granting full permissions of VOD to existing subusers.

Using policy syntax

In the example below, we will create a custom policy, which allows all actions to be performed on VOD subapplications 1400000001 and 1400000002 but denies `ProcessMedia` for subapplication 1400000001.

1. Access the **Policy** page in the CAM Console as a root account and click **Create Custom Policy**.
2. Select **Create by Policy Syntax** to enter the policy creation page.
3. In the **Select Template Type** box, select **Blank Template**.

Note:

A policy template is used to create a policy by copying an existing policy (preset or custom) and then making adjustment to the copy. In actual use, you can choose an appropriate policy template based on the actual conditions to reduce the difficulty and workload of writing policy content.

4. Click **Next** and rename the policy name as needed (or leave it unchanged).
5. Enter the following policy content in the **Edit Policy Content** box:

```
{
 "version": "2.0",
 "statement": [
     {
         "effect": "allow",
         "action": [
             "name/vod:*"
         ],
         "resource": [
             "qcs::vod::uin/12345678:subAppId/1400000001",
             "qcs::vod::uin/12345678:subAppId/1400000002"
         ]
     },
     {
         "effect": "deny",
         "action": [
             "name/vod:ProcessMedia"
         ],
         "resource": [
             "qcs::vod::uin/12345678:subAppId/1400000001"
         ]
     }
 ]
}
```

Note:

The policy content should follow the CAM policy syntax rules, where the syntax of "resource" and "action" is as shown above in Policy Syntax Description for Resource and Policy Syntax Description for Action.
6. Click **Create Policy** to create the custom policy. Subsequently, this policy can be granted to subusers in the same way as the example of granting full permissions of VOD to existing subusers.

Using server API

For most developers, performing permission management operations in the console can meet their business needs. However, if you need to automate and systematize your permission management capabilities, you can use server APIs.
The server APIs related to policies belongs to CAM. For more information, please see the CAM documentation. Only a few main APIs are listed below:
CreatePolicy
DeletePolicy
AttachUserPolicy
DetachUserPolicy

Media File Download

Last updated：2023-03-07 11:20:50

You can download media files stored in VOD and save them to a local disk or elsewhere.

Downloadable media files

VOD offers different types of media files for download, including the original files you uploaded and the files generated as you use VOD's services, such as transcoding files, screenshots, and thumbnails.

Audio/Video

Original audio/video files: The audio/video files you uploaded to VOD.

Processed audio/video files: The audio/video files generated by media processing tasks, such as transcoding files and adaptive bitrate files.

Image

Original image files: The image files you uploaded to VOD.

Processed image files: The image files generated by media processing tasks, such as screenshots, image sprites, and animated images.

Getting download URLs from the console

Log in to the VOD console. Select **Media Assets** on the left sidebar and click **Video/Audio Management** or **Image Management**. Find the target file and click **Manage** on the right to get the download URL of the file. For details, see Audio/Video Management or Managing Image.

Log in to the VOD console and select **Media Assets > Video/Audio Management** on the left sidebar. Click the download icon in the top right corner to export the download URLs of all the files. For details, see Exporting the Audio/Video List.

Getting download URLs using APIs

You can also use the following APIs to get the download URLs of media files.

DescribeMediaInfos

SearchMedia

Naming a downloaded file

Normally, if you open the URL of a media file with a browser, instead of downloading the file, the browser will open the file. For example, if you open the URL of a video with a browser, the browser will start playing the video. To download the file, you can add the parameter `download_name` to the query string. This also allows you to name the downloaded file. Here is an example:

```
http://example.vod2.myqcloud.com/dir1/dir2/myVideo.mp4?download_name=
[download_name]
```

Note:

If you enable hotlink protection, download may be subject to restrictions such as a referer allowlist/blocklist or URL expiration time. For details, see Hotlink Protection Settings.

If a video is encrypted, the transcoding file you download will also be encrypted. You need to decrypt the file first in order to play the video. For details, see Play back an encrypted video.

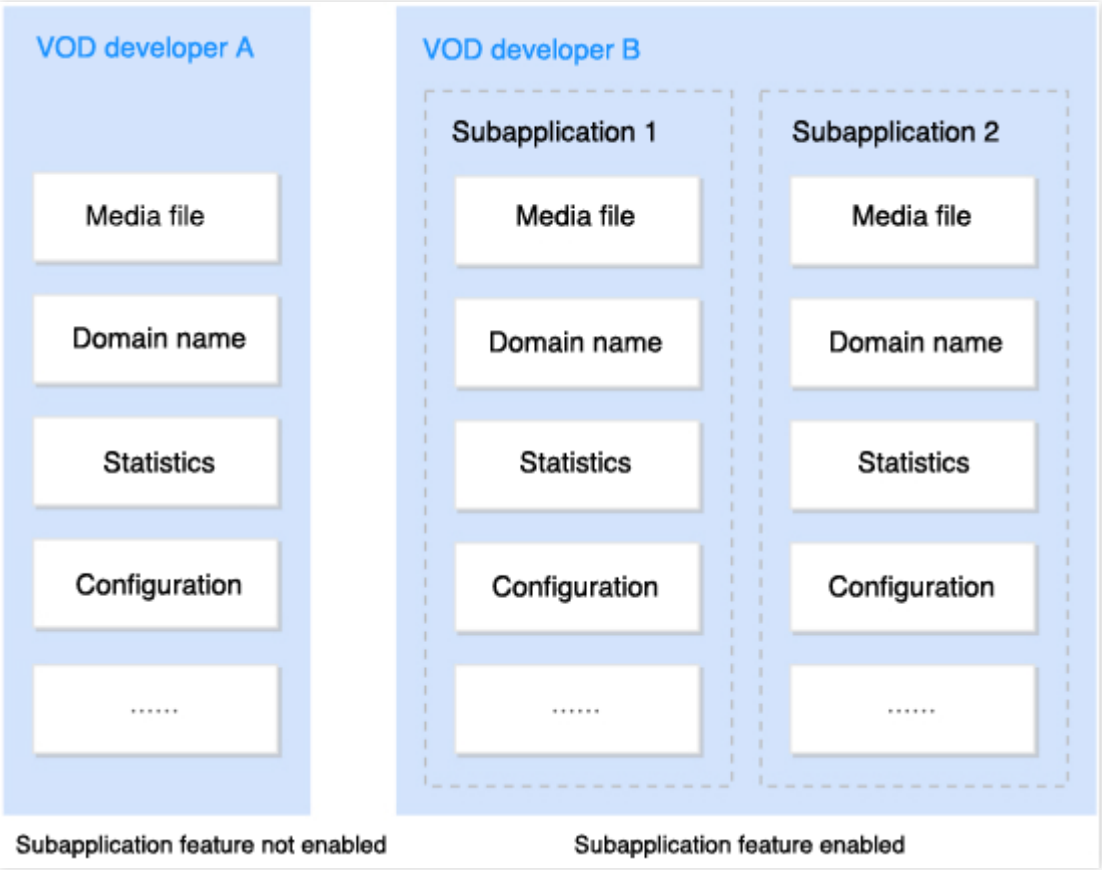
For HLS files, you need to download both the index file and segment files. To avoid this, you can transcode an HLS file into MP4.

Subapplication System

Last updated：2023-09-07 18:10:05

Overview

VOD's **subapplication** feature helps you achieve resource isolation. It is VOD's way of grouping resources. A subapplication functions as an independent account within VOD. The diagram below details how a subapplication works:



Note:

Resources mentioned in this document include media files in VOD and their attributes, derivative files, configurations, CDN domain names, and usage statistics.

Use cases

Below are some typical use cases for VOD subapplications:

Multi-department/multi-business isolation: A company is using Tencent Cloud VOD to develop its products. Department A is developing a UGSV application, and department B is building a video website. These two businesses need to be isolated from each other. However, out of financial considerations, the company does not want to create

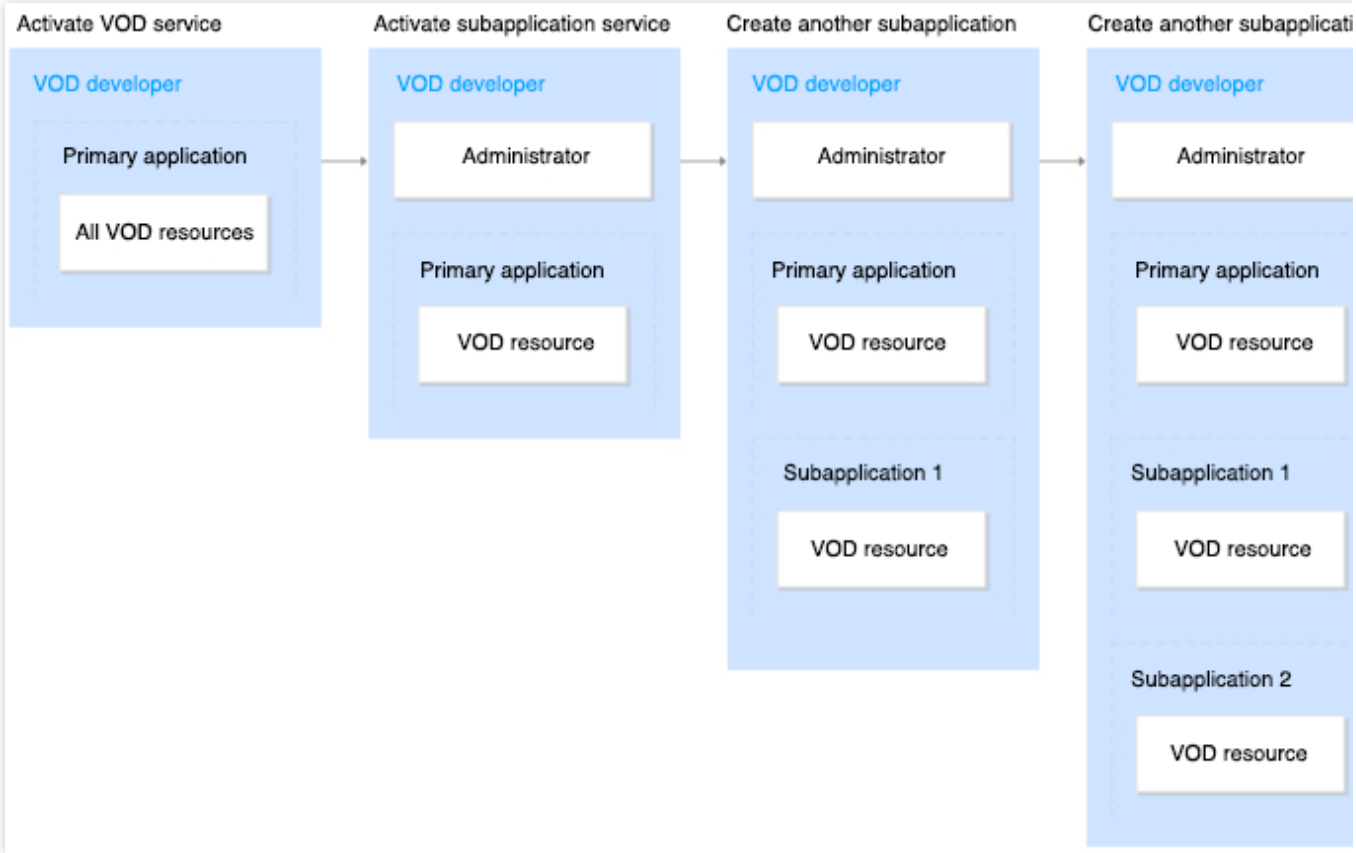
two Tencent Cloud accounts. With the subapplication feature, it can achieve resource isolation simply by creating one subapplication for either department.

Permission control: In the above scenario, the company may also want to control permissions to the resources of the two departments. For example, it may want to allow the two departments to only access their own VOD subapplications. In this case, it can create a sub-user for either department and grant them permissions to the corresponding subapplication. For detailed directions, see Access Management.

Production/Test environment isolation: If you want to test a VOD feature (for example, you may want to modify event notification settings or enable hotlink protection), to prevent it from affecting your active business, you can create one subapplication for your test environment and one for your production environment. Run tests using the test subapplication and, after you are sure about using the feature in the production environment, switch to the production subapplication.

Role definitions and IDs

There are two types of identities in the application system: administrator and application. We illustrate their definitions with the following figure.



1. After the developer activates the cloud on-demand service, a default application is directly generated. At this time, all on-demand resources belong to the **default application**. The default application ID is your account `APPID`, which can be viewed in Account Info in the console.

2. After you enable the VOD subapplication feature, an **admin** role will be generated, which does not own any VOD resources. All resources still belong to the default application.

3. You can then use the admin role to create a **subapplication**. This subapplication will exist in parallel with the default application (a special subapplication) and its resources are separated from the default application. VOD will assign the supplication an ID that is unique across the system. For how to view this ID, see Console Guide - Application Management.
4. If you create another **subapplication**, it will also exist in parallel with the default application and the other subapplication, and its resources will be separated from them.

Note:

Unless otherwise specified, where **subapplication** is mentioned below, the default application is also included.

Capabilities

The VOD subapplication system provides the following capabilities:

Creating and setting subapplications: After you enable the VOD subapplication feature, you can create subapplications in the console as the admin and set the name and description for each subapplication.

Disabling subapplications: All subapplications except the default application can be disabled. When a subapplication is disabled, its domain will also be disabled. However, the resources of the subapplication will not be cleared, and features such as media upload and transcoding will not be affected.

Isolating resources: On-demand resources between applications are isolated from each other.

You can manage VOD resources of any subapplication either via the console or using server APIs.

Statistics such as storage usage, bandwidth/traffic usage, transcoding durations, intelligent recognition durations, and playback data are generated separately for each subapplication.

Overall statistics for all subapplications are also generated.

Limits

The VOD subapplication system has the following limits:

The name and description of the default application cannot be modified.

Subapplications cannot be deleted.

At most 50 subapplications can be created under one VOD account.

Subapplications cannot be billed separately. You cannot use different billing modes or generate separate bills for subapplications. Nor can you buy dedicated packages for a subapplication. The usage (including storage, traffic, transcoding duration, intelligent recognition duration) of all subapplications under a VOD account is aggregated and billed together.

Console Guide

Enabling the subapplication feature

1. Log in to the VOD console.
2. Click **Activate Subapplication** on the left sidebar.

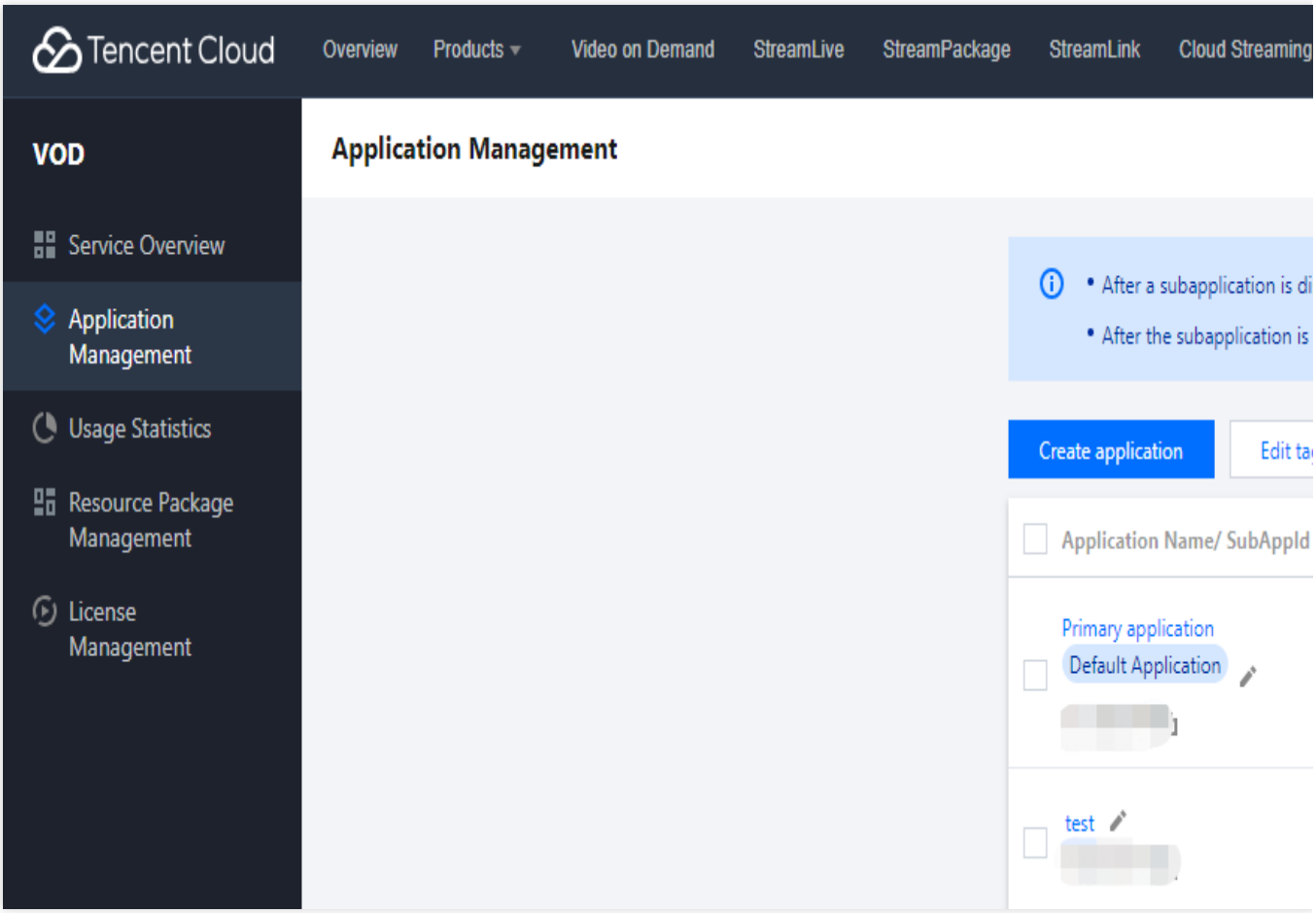
3. Click **Use now**.

Note:

If you cannot find **Enable Subapplication** on the left sidebar, the feature is already enabled for your account.

Selecting a role

After activating the application function, enter the VOD console application management list, where developers can select identities. If the developer has just activated the application function, there will be only one option in the list: "Primary application"; if the developer creates a new application, the corresponding identity option will be added to the list.



Admin

Under the admin role, you will find **Service Overview**, **Application Management**, **Usage Statistics**, **Resource Package Management**, and **License Management** on the left sidebar.

Service Overview: This page displays your VOD billing mode, aggregated key data for all subapplications, as well as separate key data for each subapplication.

Application Management: On this page, you can view, create, edit, or disable subapplications. Subapplication IDs are also displayed on this page.
Usage Statistics: On this page, you can view your account's usage of different VOD features.
Resource Package Management: This page shows your package usage.
License Management: On this page, you can view the RT-Cube licenses you have bound.

Subapplication

The console view under a subapplication role is basically the same as the view before the subapplication feature is enabled, except that a subapplication view does not have a billing section. Under a subapplication role, you can view and manage resources under the current supplication.

Server APIs

After enabling the subapplication feature, you must specify the subapplication whose resources you want to access when using VOD server APIs.

Specifying a subapplication for server APIs

VOD server APIs have been upgraded to TencentCloud API 3.0. You can use the `SubAppId` parameter to specify the subapplication you want to access. If you want to access the default application, pass in the default application ID or leave the parameter empty.

Specifying a subapplication for server APIs 2017

The 2017 version also supports subapplications. Just add a `SubAppId` parameter (case-sensitive) in your request. This parameter has the same level as other common request parameters of API 2017. If you want to access the default application, pass in the default application ID or leave the parameter empty.
Note:
Server API 2017 documentation does not disclose the `SubAppId` parameter. This does not affect its use though. The `SubAppId` parameter is also involved in signature calculation for server APIs. The calculation rules are the same.

File Upload

After enabling the VOD subapplication feature, you must specify the subapplication to which you want to upload your media files.

Live recording

You can save live recordings to a specific subapplication by adding the publishing parameter `vod_sub_app_id=xxx` (`xxx` is the subapplication ID). You don't need to carry this parameter if you record to the primary application.

Uploading files from the server

You can upload files from the server to a specific subapplication. For detailed directions, see the documents below. If you want to upload files to the default application, pass in the default application ID or leave it empty.

Server upload SDKs

Java SDK
PHP SDK
Python SDK
Node.js SDK
Golang SDK

Server APIs

ApplyUpload and CommitUpload are the two APIs used to upload files. For detailed directions on how to use them, see Specifying a subapplication for server APIs.
You are strongly recommended to use the SDK for upload.

Uploading files from a client

When uploading files from a client, you can specify the subapplication to upload to by adding a `vodSubAppId=xxx` parameter (`xxx` is the subapplication ID) in the client upload signature. If you want to upload to the default application, pass in the default application ID or leave the parameter empty.
Note:
The `vodSubAppId` parameter is also involved in the calculation of client upload signatures. The calculation rules are the same.

Pulling files from a URL

If you upload files by having VOD pull from a URL, you can also specify the subapplication to upload to.
Console: For detailed directions, see Console Guide.
Server API: Use the PullUpload API. For more information, see Specifying a subapplication for server APIs.

Permission Management

VOD has been connected to CAM and supports authorization at the subapplication level. For details, see CAM.

FAQs

After the subapplication feature is enabled, will it affect existing business logic in the production environment?

No. The subapplication system is designed with compatibility in mind. If a subapplication ID is not specified, all server APIs will take effect only for the default application.

Will fees be charged for enabling the subapplication feature?

Each user can create 20 applications (including default applications) for free, and some applications exceeding 20 will be charged separately. At the same time, the consumption generated by each application will be included in the cloud-on-demand account, and will be charged according to the VOD billing logic for billing.

My company uses the subapplication feature to implement business isolation. How can I allocate the costs to different businesses?

As described in Limits, VOD only generates one aggregated bill for a VOD account. However, we provide usage statistic for each subapplication to facilitate your cost allocation.

What will happen to a subapplication if my VOD service is suspended?

If your VOD service is suspended, all subapplications under your account will be disabled.

Can I migrate videos from one subapplication to another?

The resources of different subapplications are isolated from one another. You cannot move resources between subapplications.

Error Codes

Last updated：2021-11-10 10:59:42

Video processing

Error Code	Description
InvalidInput	Invalid input parameter. Please check.
InvalidInput.InvalidTimeOffset	Invalid input parameter: the specified time point is invalid.
InvalidInput.DefinitionNotExist	Invalid input parameter: the specified template ID doesn't exist.
InvalidInput.ConfigurationUnsupported	Invalid input parameters. Reasons include but are not limited to: the user has not registered. The input parameter value is invalid (due to errors in the format, value range or others). The parameter template configuration is invalid. No video processing task is specified.
InvalidInput.TaskDuplicated	Invalid input parameter: duplicate task
InvalidInput.PermissionDenied	Invalid input parameter: you do not have permission to use this feature. Please apply for the permission first.
InvalidInput.ResultFileSizeTooLarge	Invalid input parameter: the spliced file is too large after inputting multiple files.
SourceFileError	Invalid source file: for example, video data is corrupted. Please check whether the source file is normal.
SourceFileError.NoVideoMedia	Invalid source file: there is no video image.
SourceFileError.NoVideoResolution	Invalid source file: the resolution of the source file cannot be obtained.
SourceFileError.ContentMalformed	Invalid source file: errors occur in the input content. For example, the file does not exist, the file is corrupted, or the media file cannot be decoded.
SourceFileError.ContentUnsupported	Invalid source file: the input file is invalid due to unsupported file format, size, duration, or other reasons.
SourceFileError.DownloadNotAccessible	Invalid source files: the files are not accessible during download.

	Check the availability of the source files.
InternalError	Internal service error. Please try again.