ALIBABA CLOUD

Alibaba Cloud

视频直播

Stream ingest SDK

Document Version: 20210826

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Product introduction

This topic provides an overview of Push SDK and describes the features, benefits, application scenarios, and operation flow of Push SDK.

Overview

Push SDK is a stream pushing development tool for ApsaraVideo Live clients. It is developed on top of the powerful Content Delivery Network (CDN) and real-time audio-video communication technologies of Alibaba Cloud. Push SDK provides easy-to-use APIs, Network Auto-Adaptability (NAA), multi-node-based low-latency optimization, and real-time beauty filters for a superior user experience. Push SDK is free of charge. Push SDK offloads the burden of designing and maintaining complex system architectures, allowing you to focus on implementing business logic and improving user experience.

Push SDK integrates smart retouching to provide advanced beauty filters, such as face shrinking, face slimming, eye enlarging, and skin whitening.

> **Note**
>
> To use the smart retouching feature, you need to apply for a license. For more information about smart retouching and how to enable this feature, see Overview.

Features

Feature	Description
Real-Time Messaging Protocol (RTMP) push	Supports stream pushing over RTMP, and stream pulling over RTMP, Flash Video (FLV), HTTP Live Streaming (HLS), and Advanced Real-Time Communication (ARTC). Supports the resolution of 180p to 720p. We recommend that you use 540p.
Web Real-Time Communication (WebRTC) push	Supports stream pushing over ARTC based on User Datagram Protocol (UDP).
Live stream recording	Supports live recording on iOS devices using ReplayKit and live multi-camera recording on Android devices.
Insertion of SEI messages	Allows you to insert SEI messages into live streams. SEI messages are parsed by the player, allowing you to implement advanced functions.
Dynamic watermarking	Allows you to add or remove animated watermarks from live streams in real time.
Ingest of external audio and video streams	Supports external audio and video streams for live streaming.

Feature	Description
Background image stream pushing	Allows you to push images after you move the app to the background. This function also allows you to push images under poor network connections.
Audio and video encoding	Supports H.264 software and hardware encoding and Advanced Audio Coding (AAC) software and hardware encoding.
Real-time beauty filters	Supports advanced beauty filters based on facial recognition, including skin polishing, skin whitening, face thinning, face slimming, and eye enlarging.
Dynamic bitrate	Automatically adjusts the bitrate during stream pushing based on network conditions and sets modes to improve your live streaming quality.
Dynamic resolution	Automatically adjusts the resolution during stream pushing based on network conditions. This function is supported only in high resolution-priority and fluency-priority modes.
Stream pushing from background	Supports uninterrupted live streams when switching between the foreground and background. When you switch to the foreground, live streaming continues.
Stereo stream pushing	Supports stereo stream pushing in mono or binaural mode.
Watermarks	Allows you to add up to three watermarks. You can change the position and size of the watermarks.
Stream pushing in landscape mode	Supports stream pushing in portrait, landscape left, and landscape right modes.
Capture parameters	Allows you to set capture parameters, such as the resolution, frame rate, audio sampling rate, group of pictures (GOP), and bitrate, to meet diverse requirements.
Stream pushing in mirroring mode	Allows you to mirror the video images collected from cameras and push the streams. By default, the mirroring feature must be enabled for front cameras.
Audio-only stream pushing	Allows you to collect only audio streams and push the audio streams. This reduces your bandwidth consumption and traffic usage in audio-only scenarios.

Feature	Description
Muted stream pushing	Allows you to mute the microphone and push only video images.
Autofocus	Allows you to enable or disable the autofocus feature. You can also manually set the focus point.
Zoom of lens	Zooms captured images according to the maximum zoom ratio supported by the camera.
Camera switching and flash	Allows you to switch between the front and rear cameras, and turn on or turn off the flash when you use the rear camera.
Background music	Plays music in the background, including start, stop, pause, resume, and loop.
Audio mixing	Allows you to mix music with vocals and adjust the volume of music and vocals.
In-ear monitoring	Supports the in-ear monitoring feature. For example, when streamers are singing with headsets, they can hear their voice from headsets in real time. This meets the live streaming requirements in Karaoke scenarios.
Noise reduction	Reduces noise that comes from the environment and mobile phones.

Benefits

- **Ease of integration**

  Push SDK provides unified synchronous and asynchronous APIs and error codes for Android and iOS apps to meet the integration requirements in different development architectures. A full set of API references and demos are available for you to use.

- **All-in-one solution**

  Push SDK delivers an all-in-one live streaming solution that integrates video capture, rendering, stream pushing, transcoding, distribution, and playback. Push SDK streamlines adaptive-bitrate stream pushing on clients, Narrowband HD™ transcoding in the cloud, and instant loading on playback devices to provide one-stop quality services.

- **High performance and low latency**

  Push SDK delivers the industry-leading freezing rate, CPU utilization, memory usage, power consumption, and heat generation. More than 2,500 CDN nodes around the world ensure low latency in each region.

- **WebRTC push**

WebRTC push based on UDP is provided. You can add an ingest domain for WebRTC in the console. WebRTC push can better prevent stuttering in the cases where the upstream network quality is poor.

Scenarios

Scenario 1: Live streaming for education

- **Scenario description**: Live streaming for education usually focuses on the interaction between teachers and students. When accessing a live online class, you can use Alibaba Cloud Message Service to implement real-time text and voice interactions between teachers and students. Push SDK enables teachers to answer questions raised by students anytime and anywhere, in a timely and effective manner. By using cloud-based recording and transcoding, students can replay lessons on demand to review knowledge learned and reinforce their learning.
- **Usage instructions**: Activate ApsaraVideo Live and enable recording and transcoding. Use Push SDK and Message Service SDK to have live online classes or Q&A. Use ApsaraVideo Player SDK on a playback device for users to watch or replay live video lessons in a low-latency and interactive manner.

Scenario 2: Entertainment live streaming

- **Scenario description**: Due to the convenience of smartphones, entertainment live streaming has become a nationwide trend. Streamers use beauty and effect filters a lot. They interact with viewers through live chat, likes, and rewards to improve their popularity and boost audience engagement. In addition, the threshold for entertainment live streaming on a smartphone is relatively low, and the security of the corresponding content (such as pornography and violence) needs to be strictly controlled. You can use live pornographic-content identification to reduce costs of censorship.
- **Usage instructions**: Activate ApsaraVideo Live and enable recording and pornographic-content identification. Use Push SDK and enable beauty filters to push live streams and integrate Alibaba Cloud Message Service to serve your interactive chat needs. Viewers can send text messages and images on the chat panel when they watch a live stream. You can also build your own gift system (IM to support payment) and use Alibaba Cloud Player SDK on a playback device to watch or replay a live stream.

Scenario 3: Video game live streaming

- **Scenario description**: Mobile game broadcasting uses screen recording to merge the current game image with the image captured from the camera, and then uses Push SDK to initiate stream pushing. Therefore, Push SDK needs to support screen recording. The interaction between the streamer and audience is basically similar to that of entertainment live streaming. You can use Alibaba Cloud Message Service SDK to have interactions like chats, likes, and rewards, and record exciting content in the game for on-demand playback.
- **Usage instructions**: Activate ApsaraVideo Live and enable live recording. Access Push SDK, use live recording, and integrate Alibaba Cloud Message Service to serve your interactive chat needs. Viewers can send text messages and images on the chat panel when they watch a live stream. You can also build your own gift system (IM to support payment) and integrate Alibaba Cloud Player SDK to achieve instant loading and dynamic frame synchronization when watching or replaying a live stream.

SDK operation flow

1. Your app initiates a request to the app server for obtaining a stream ingest URL.
2. The app server assembles the stream ingest URL based on the specified rules and returns the URL to the app.
3. The app passes the stream ingest URL to Push SDK and uses Push SDK to initiate stream pushing.

4. Push SDK pushes the live stream to CDN.

Support for developers

If you have any questions or suggestions about Push SDK, join the developer ecosystem group of Push SDK. Search for group number 32825314 on DingTalk or scan the following QR code.

Push SDK support group

2.Terms

This topic describes the terms that are related to Push SDK.

- Bitrate control: Bitrate control uses an optimized coding algorithm to control the bitrate of video streams. In the same video coding format, video streams at a higher bitrate contain more information and provide clearer images.
- Frame skipping: In cases of low network bandwidth, frames may be congested when they are sent. The SDK can skip specific frames to shorten the latency of stream ingest.
- In-ear monitoring: In-ear monitoring allows streamers to hear their voice in real time from the headset that they are wearing. For example, streamers can enable in-ear monitoring to help tune their voice when they are singing with a headset. This is because the effects of audio transmitted into ear canals over the network are quite different from the effects of audio transmitted through the air. Streamer need to know the effects of audio that viewers can hear on their clients.
- Audio mixing: Audio mixing combines multiple audio sources into a stereo or mono audio track. Push SDK supports mixing of music and vocals.
- Stream merging: Stream merging overlays video frames with video image data from multiple sources based on the timeline. This feature is supported only by Push SDK for Android.
- Dynamic library: A dynamic library is also known as a dynamic link library (DLL). Dynamic libraries are different from commonly used static libraries in that dynamic libraries are not copied to an application at the time of compilation. Only references to the dynamic libraries are stored in the application. Dynamic libraries are loaded only when the application is running.

> **Note** When you load dynamic libraries in Xcode, you must add the dynamic libraries to the Embedded Binaries section instead of the Linked Frameworks and Libraries section.

3.SDK download and release notes

This topic provides the download QR code of the demo and the download links of Push SDK. This topic also describes the release notes of Push SDK.

Installation package of the demo

Scan the QR code to download the demo.

Release notes

Release date	Version	Description	Download link
2021-06-25	V4.1.0	Push SDK is optimized in the following aspects: • Stream ingest over Real-Time Streaming (RTS) is supported. • The Queen SDK for smart retouching is integrated. • New UI design for the demo is used. • The ApsaraVideo Player SDK is updated.	• Push SDK for Andriod • Push SDK for iOS • Source code of the demo for Android • Source code of the demo for iOS

Release date	Version	Description	Download link
2021-01-13	v4.0.2	**Notice** Push SDK is updated to V4.1.0. For more information about how to update your SDK to the latest version, see the related topic for Android or iOS. The following features are supported: • Stream ingest from cameras is supported. In this mode, stream ingest over Real-Time Messaging Protocol (RTMP) and co-stream ingest over Web Real-Time Communication (WebRTC) are supported. • The playback of live streams is supported. RTMP, Flash Video (FLV), HTTP Live Streaming (HLS), and RTS live streams can be pulled to play. • Parameter settings for collection, coding, and stream ingest are supported. • Background music and sound effects can be added. • Retouching is supported. • Co-streaming and streamer challenges are supported.However, these two features are not supported in V4.1.0 or later.	To enable these features, contact the business manager or submit a ticket.
2020-12-01	v3.6.1	The presentation timestamps (PTS) difference between audio and video streams that occurs in poor network conditions is removed.	• Push SDK for Android and demo • Push SDK for iOS and demo

The size description of the package is shown below:

Version	Platform	Package size	Install the package increments
v4.1.0	Android	9.5 MB	8.3 MB
	iOS	21.4 MB	7.8 MB

4.Android Push SDK

4.1. Update Push SDK from V4.0.2 to V4.1.0

This topic describes how to update Push SDK for Android from V4.0.2 to V4.1.0.

Procedure

Remove the class libraries and resource files of Push SDK V4.0.2 from your project. Then, add the class libraries and resource files of Push SDK V4.1.0 to your project.

1. In the libs directory, use *AlivcLivePusher.aar* of Push SDK V4.1.0 to replace *AliLiveSdk.aar* of Push SDK V4.0.2.
2. Update the ApsaraVideo Player SDK to V5.4.1 by using AliyunPlayer:5.4.1-full and AlivcArtc:5.4.1.

```
implementation 'com.aliyun.sdk.android:AliyunPlayer:5.4.1-full'
implementation 'com.aliyun.sdk.android:AlivcArtc:5.4.1'
```

3. If you want to implement the Queen smart retouching feature in your app, add the beauty, beautyui, imageutil, and queenbeauty model files in the demo to facilitate the integration of the Queen SDK and UI.
4. Modify specific interfaces, as described in the Comparison of basic operations section of this topic.
5. Modify specific interfaces for the major process, as described in the Changes in the interfaces for major process section of this topic.

Changes in the interfaces for major process

1. **Create an engine**

AliLiveEngine in V4.0.2	AlivcLivePusher in V4.1.0

// Create an AliLiveRTMPConfig object.
AliLiveRTMPConfig rtmpConfig = new AliLiveRTMPConfig();
// Initialize the bitrate settings.
rtmpConfig.videoInitBitrate = 1000;
rtmpConfig.videoTargetBitrate = 1500;
rtmpConfig.videoMinBitrate = 600;
// Create an AliLiveConfig object.
AliLiveConfig mAliLiveConfig = new AliLiveConfig(rtmpConfig);
// Initialize the resolution and frame rate, specify whether to enable high-definition preview, and specify the default image to be displayed after a pause.
mAliLiveConfig.videoFPS = 20;
mAliLiveConfig.videoPushProfile = AliLiveConstants.AliLiveVideoPushProfile.AliLiveVideoProfile_540P;
mAliLiveConfig.enableHighDefPreview = false;
mAliLiveConfig.pauseImage = bitmap;
mAliLiveConfig.accountId = "";
AliLiveEngine mAliLiveEngine = AliLiveEngine.create(PushActivity.this, mAliLiveConfig);
	// Create an AlivcLivePushConfig object.
	AlivcLivePushConfig mAlivcLivePushConfig = new AlivcLivePushConfig();
	// Initialize the resolution. The default resolution is 540p. The supported maximum resolution is 720p.
	mAlivcLivePushConfig.setResolution(AlivcResolutionEnum.RESOLUTION_540P);
	// Initialize the frame rate. We recommend that you set the frame rate to 20 fps.
	mAlivcLivePushConfig.setFps(AlivcFpsEnum.FPS_20);
	// Enable adaptive bitrate streaming. The default value is true.
	mAlivcLivePushConfig.setEnableBitrateControl(true);
	// By default, the preview is in portrait mode. You can change the mode to landscape left or landscape right.
	mAlivcLivePushConfig.setPreviewOrientation(AlivcPreviewOrientationEnum.ORIENTATION_PORTRAIT);
	// Set the audio coding format.
	mAlivcLivePushConfig.setAudioProfile(AlivcAudioAACProfileEnum.AlivcAudioAACProfileEnum.AAC_LC);
	AlivcLivePusher mAlivcLivePusher = new AlivcLivePusher();mAlivcLivePusher.init(mContext, mAlivcLivePushConfig);

2. **Create a preview**

V4.0.2	V4.1.0

// Create a preview view.
AliLiveRenderView mAliLiveRenderView = mAliLiveEngine.createRenderView(false);
// Add the preview view to the layout.
addSubView(mAliLiveRenderView);
// Set the preview mode.
mAliLiveEngine.setPreviewMode(AliLiveRenderModeAuto, AliLiveRenderMirrorModeOnlyFront);
// Start the preview.
mAliLiveEngine.startPreview(mAliLiveRenderView);
	You can start the preview after you initialize the livePusher object. Use the SurfaceView parameter for camera previews. The following code provides an example:
	// Start the preview. You can also call the asynchronous startPreviewAysnc interface based on your needs to start the preview.
	mAlivcLivePusher.startPreview(mSurfaceView)

3. Start stream ingest

V4.0.2	V4.1.0
mAliLiveEngine.startPush(mPushUrl);	mAlivcLivePusher.startPush(mPushUrl);

4. Stop stream ingest

V4.0.2	V4.1.0

// Stop the preview.
mAliLiveEngine.stopPreview();
// Stop stream ingest.
mAliLiveEngine.stopPush();
// Destroy the AliLiveEngine object.
mAliLiveEngine.destroy();
mAliLiveEngine = null;
	// Stop the preview.
	mAliLivePusher.stopPreview();
	// Stop stream ingest.
	mAliLivePusher.stopPush();
	// Destroy the AliLivePusher object.
	mAliLivePusher.destroy();
	mAliLivePusher = null;

Comparison of basic operations

- Basic interfaces

V4.0.2	V4.1.0	Description
getSdkVersion	getSdkVersion	Queries the version of Push SDK.
create	init	Creates a stream ingest object.
destroy	destroy	Destroys the stream ingest object.
setStatusCallback	setLivePushInfoListener	Sets the callback for the stream ingest status.
setNetworkCallback	setLivePushNetworkListener	Sets the callback for the network status during stream ingest.
setLogDirPath	setLogDirPath	Sets the path for storing the log files of Push SDK. To prevent log loss, you must call this interface before other API interfaces. In addition, make sure that the specified path exists and is writable.
setLogLevel	setLogLevel	Sets the log level.

- **Basic stream ingest interfaces**

V4.0.2	V4.1.0	Description
startPreview	startPreview	Starts the preview. This interface is called on the streamer side.
stopPreview	stopPreview	Stops the preview. This interface is called on the streamer side.
pausePush	pause	Stops the camera from collecting live streams and ingests standby streams. This interface is supported only for stream ingest over Real-Time Messaging Protocol (RTMP). You must call the startPush interface before the pausePush interface to prevent an invalid call order.
resumePush	resume()	Enables the camera to collect live streams again and stops ingesting standby streams. This interface is supported only for stream ingest over RTMP. You must call the pausePush interface before the resumePush interface to prevent an invalid call order.
startPush	startPush	Starts stream ingest.
stopPush	stopPush	Stops stream ingest.

V4.0.2	V4.1.0	Description
isPublishing	isPublishing	Queries whether streams are being ingested.
getPublishUrl	getPushUrl	Queries the current ingest URL.

- **Video-related interfaces**

V4.0.2	V4.1.0	Description
setPreviewMode	setPreviewMode	Sets the preview mode.
isAudioOnly	isAudioOnly	Queries whether the ingested streams are audio-only streams.
switchCamera	switchCamera	Switches between the front camera and the rear camera.
setCameraZoom	setCameraZoom	Sets the zoom factor of the camera and specifies whether to enable the flash.
isCameraExposurePointSupported	setExposureCompensation	Queries whether an exposure point can be set for the camera.
setCameraFocusPoint	setLiveCameraFocus	Sets the focus point of the camera.

- **Audio-related interfaces**

V4.0.2	V4.1.0	Description
setMute	setMute	Specifies whether the frames collected from the local audio are mute frames.
isAudioOnly	isAudioOnly	Queries whether the ingested streams are audio-only streams.
enableEarBack	setBGMEarsBack	Enables in-ear monitoring. To prevent echoes, we recommend that you enable in-ear monitoring after you insert the headset.
playBGM	startBGMAsync	Plays background music.
stopBGM	stopBGM	Stops playing background music.
pauseBGM	pauseBGM	Pauses the playback of background music.
resumeBGM	resumeBGM	Resumes the playback of background music.
setBGMVolume	setBGMVolume	Sets the volume of background music.

4.2. Demo compilation

This topic describes the requirements for the runtime environment of the Push SDK demo for Android. This topic also describes how to compile the demo and provides the directory structure of the demo.

Environment requirements

- The following table describes the mandatory requirements for the runtime environment.

Item	Requirement
Android version	Android 5.0 or later
Android API version	Jelly Bean (API level 18) or later
CPU architecture	ARM64 or ARMv7
Integration tool	Android Studio (recommended). To download Android Studio, visit the Android Studio page.

- The following table describes the optional requirements for the runtime environment. The demo is based on the environment information described in the table. The environment information is provided for your reference.

Item	Requirement
Android Studio version	4.1.3
JRE	1.8.0_152-release-1136-b06 amd64
JVM	OpenJDK 64-Bit
compileSdkVersion	30
buildToolsVersion	30.0.3
minSdkVersion	18
targetSdkVersion	29
gradle version	gradle-5.6.4-all
gradle plugin version	com.android.tools.build:gradle:3.6.2

Run the demo

1. Download the demo package.

   Download the demo package of the required version by using the download URL provided in the SDK download and release notes topic.

2. Import the demo project.

   Open **Android Studio**, click **Open an Existing Project**, and then select the *AlivcLivePusherDemo* project in the directory of the demo to import the project to Android Studio.

3. Compile and run the demo.

   After the demo is compiled, click **Run** to install the demo on an Android device.

4. Experience the features provided in the demo.

   - Homepage of the mobile app for stream ingest

   - Stream ingest settings

     Tap **Camera capture** to configure the stream ingest settings, as shown in the following figure.

     
     Stream ingest settings - Feature settings

   - Stream ingest and retouching

     Enter the ingest URL and tap **Enter**. The retouching feature is available during stream ingest, as shown in the following figure.

     Stream ingest and retouching

   - Stream ingest for screen recording

     Screen recording 1

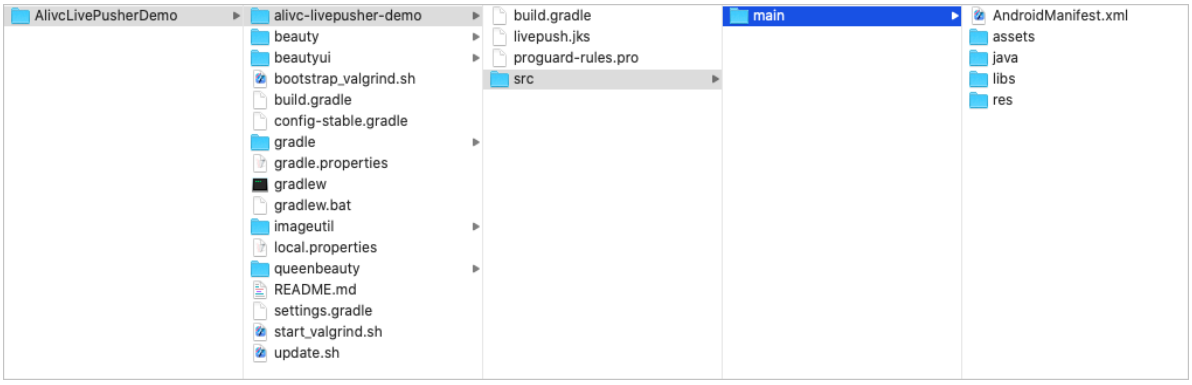
     Screen recording 2

     Screen recording 3

You must specify a valid Real-Time Messaging Protocol (RTMP) URL to start stream ingest. After the stream is ingested, you can use the ApsaraVideo Player SDK, FFPlay, or VLC to watch the stream.

Directory structure of the demo for Android

- **Directory structure of the demo V4.1.0**



The main folder contains the following files or subfolders:

- AndroidManifest.xml: the configuration file of the demo for Android.
- assets: the folder where the resource files reside.
- java: the folder where the demo code resides.

  The following figure shows the directory structure of the java folder.

- libs: the folder where the dependent JAR packages reside.
- res: the folder where the resource files and layout files reside.

- **Directory structure of the demo V3.6.1**

  The main folder contains the following files or subfolders:

  - aarLibs: the folder where the dependent AAR packages reside.
  - AndroidManifest.xml: the configuration file of the demo for Android.
  - assets: the folder where the resource files reside.
  - java: the folder where the demo code resides.
  - jniLibs: the folder where the dependent SO libraries reside.
  - libs: the folder where the dependent JAR packages reside.
  - res: the folder where the resource files and layout files reside.

4.3. SDK integration

This topic describes how to integrate Push SDK for Android.

Environment requirements

Item	Requirement
Android version	Android 5.0 or later
Android API version	Jelly Bean (API level 18) or later
CPU architecture	ARM64 or ARMv7
Integration tool	Android Studio (recommended). To download Android Studio, visit the Android Studio page.

Integrate the SDK

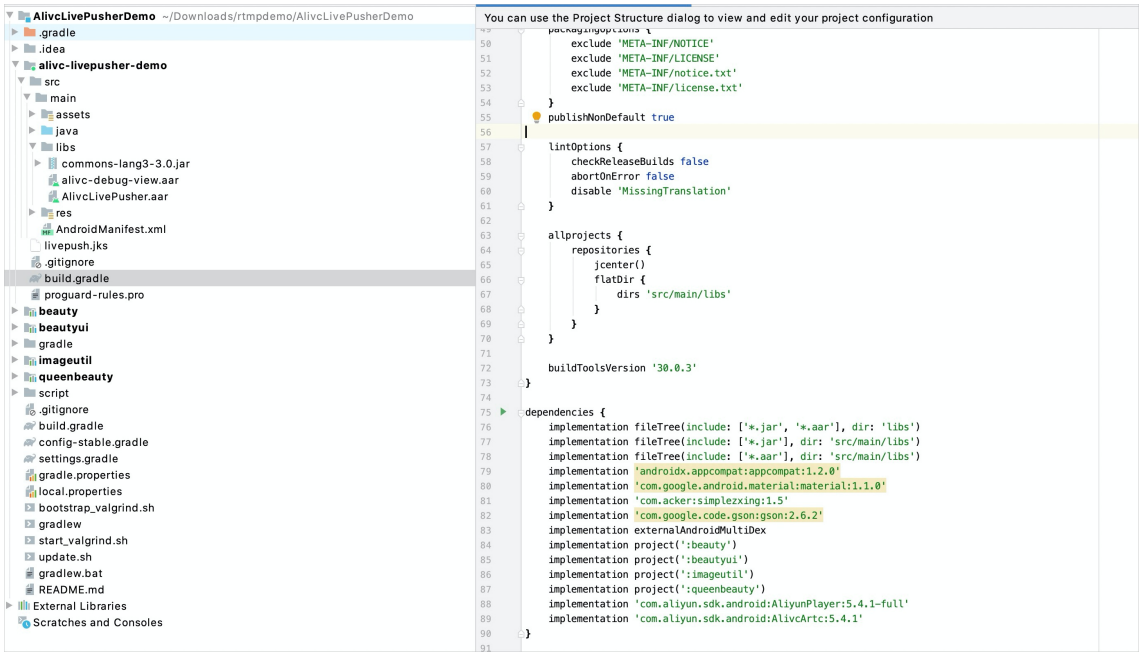
1. Integrate the SDK by using AAR packages or Maven dependencies.

   > **Note** Download the SDK package of the required version by using the download URL provided in the SDK download and release notes topic.

   - **Use AAR packages**

     Create a project and copy all the files in the SDK package to the libs folder of your project. Modify the dependencies in the build.gradle file of the main module in your project. Generally, the main module is named as app. Then, run the following code to recompile your project:

     ```
     implementation fileTree(dir: 'libs', include: ['*.jar', '*.aar'])
     ```



> **Note** If you want to add background music (BGM), you must use the ApsaraVideo Player SDK, which is encapsulated in the AliyunPlayer.aar package.

- **Use Maven dependencies**

  Add the following code to the allprojects block in the build.gradle file:

```
implementation 'com.alivc.pusher:AlivcLivePusher:4.1.0'
```

2. Configure the permissions to access features and resources.

   Add the following permission configuration to the AndroidManifest file:

```
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.REORDER_TASKS" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
```

> **Note** To use the recording feature and the camera, you must manually specify the names of the related permissions.

3. Add the obfuscation configuration.

   Add the following code to the proguard-rules.pro file:

   ```
   -keep class com.alivc.** { *;}
   ```

4. Use the SDK by referring to the related topics.
   - For information about the interfaces provided in the SDK, see Interfaces in the SDK for Android V4.1.0 or the **api** folder in the SDK package.
   - For information about the sample code to use specific interfaces, see Use Push SDK for Android.

4.4. Use Push SDK for Android

This topic describes how to use the interfaces in Push SDK for Android and the procedure for using Push SDK for Android. This topic also provides examples on how to use Push SDK for Android to implement specific features.

Push SDK for Android

- Supports the stream ingest over Real-Time Messaging Protocol (RTMP).
- Adopts H.264 for video coding and Advanced Audio Coding (AAC) for audio coding.
- Supports custom configurations for features such as bitrate control, resolution, and display mode.
- Supports a variety of camera operations.
- Supports real-time retouching and allows you to customize retouching effects.
- Supports using animated stickers as animated watermarks and allows you to add and remove animated watermarks.
- Supports live stream recording.
- Supports external audio and video input in different formats such as YUV and pulse-code modulation (PCM).
- Supports mixing streams.
- Supports the ingest of audio-only and video-only streams and stream ingest in the background.
- Supports background music and allows you to manage background music.
- Supports capturing snapshots from streams.
- Supports automatic reconnection and exception handling.

List of operations by function

Parameter	Description
AlivcLivePushConfig	The configuration class that is used to initialize the stream ingest configurations.
AlivcLivePusher	The core class of Push SDK.
AlivcLivePusherErrorListener	The callbacks that are used when errors occur.

Parameter	Description
AlivcLivePusherNetworkListener	The callbacks that are used to manage network services.
AlivcLivePusherInfoListener	The callbacks that are used for notifications and status detection.
AlivcLivePusherBGMListener	The callbacks that are used for playing background music.
AlivcLivePushCustomFilter	The callbacks that are used for custom filter services.
AlivcLivePushCustomDetect	The callbacks that are used for custom face detection.
AlivcSnapshotListener	The callbacks that are used for capturing snapshots.

Procedure

You can use Push SDK for Android by following these steps:

1. Set the stream ingest parameters.
2. Create a stream ingest preview.
3. Start to ingest streams.
4. Adjust ingest parameters in real time.

Set the stream ingest parameters

You can configure stream ingest parameters in the AlivcLivePushConfig class. Each parameter has a default value. For more information about the default values and valid values, see Interfaces in the SDK for Android V4.1.0 or comments in the codes.

You can modify the values of parameters as needed. To modify these parameters in real time, see the parameters and methods of the AlivcLivePusher class.

This section provides sample code for specific configurations.

- **Basic configurations**

```
AlivcLivePushConfig mAlivcLivePushConfig = new
AlivcLivePushConfig(); // The configuration class used to initialize the stream ingest configurations.
mAlivcLivePushConfig.setResolution(AlivcResolutionEnum.RESOLUTION_540P); // Set the resolution to 5
40p. The maximum is 720p.
mAlivcLivePushConfig.setFps(AlivcFpsEnum.FPS_20); // We recommend that you set the frame rate to 20 f
ps.
mAlivcLivePushConfig.setEnableBitrateControl(true); // Enable adaptive bitrate streaming. The default va
lue is true.
mAlivcLivePushConfig.setPreviewOrientation(AlivcPreviewOrientationEnum.ORIENTATION_PORTRAIT);
// The default screen orientation is portrait. You can change the orientation to landscape left or landscape
right.
mAlivcLivePushConfig.setAudioProfile(AlivcAudioAACProfileEnum.AlivcAudioAACProfileEnum.AAC_LC); //
Set the audio coding format.
mAlivcLivePushConfig.setEnableBitrateControl(true); // Enable adaptive bitrate. The default setting is tru
e.
```

Note:

- All these parameters have default values. We recommend that you use the default values.
- Considering the performance of most mobile phones and network bandwidth requirements, we recommend that you set the resolution to 540p. Most mainstream live streaming apps use 540p.
- After adaptive bitrate streaming is disabled, the bitrate is fixed at the initial value and is not automatically adjusted between the specified target bitrate and the minimum bitrate. If the network is unstable, this setting may cause stuttering. Use the adaptive bitrate streaming feature with caution.

- **Bitrate control**

  Push SDK for Android provides three bitrate control modes. You can specify a mode by setting the AlivcQualityModeEnum parameter.

  - **QM_RESOLUTION_FIRST**: quality first. Push SDK for Android configures bitrate parameters to ensure the quality of video streams first.
  - **QM_FLUENCY_FIRST**: smoothness first. Push SDK for Android configures bitrate parameters to ensure the smoothness of video streams first.
  - **QM_CUSTOM**: custom mode. Push SDK for Android configures bitrate parameters based on your custom settings.

  The following code provides an example:

  - The following code provides an example on how to configure quality-first or smoothness-first mode:

    ```
    mAlivcLivePushConfig.setQualityMode(AlivcQualityModeEnum.QM_RESOLUTION_FIRST); // Quality firs
    t.
    ```

    Note:

    If you use the quality-first or smoothness-first mode, you do not need to set the initialVideoBitrate, minVideoBitrate, and targetVideoBitrate parameters. Push SDK for Android automatically ensures the quality or smoothness of video streams when the network is unstable.

- The following code provides an example on how to configure the custom mode:

```
mAlivcLivePushConfig.setQualityMode(AlivcQualityModeEnum.QM_CUSTOM);
mAlivcLivePushConfig.setTargetVideoBitrate(1200); // The maximum bitrate is 1,200 kbit/s.
mAlivcLivePushConfig.setMinVideoBitrate(400); // The minimum bitrate is 400 kbit/s.
mAlivcLivePushConfig.setInitialVideoBitrate(900); // The initial bitrate is 900 kbit/s.
```

Note:

If you use the custom mode, you must configure the maximum, minimum, and initial bitrates.

- Initial bitrate: The initial bitrate is the bitrate when the live streaming starts.
- Minimum bitrate: When the network bandwidth is low, the bitrate is gradually reduced to the minimum value to avoid stuttering.
- Maximum bitrate: When the network bandwidth is high, the bitrate is gradually increased to the maximum value to improve the quality of the video stream.

We recommend that you use the following parameter values for the custom mode:

Resolution	initialVideoBitrate	minVideoBitrate	targetVideoBitrate
360P	800	200	600
480P	1000	300	700
540P	1200	400	900
720P	1500	600	1200

- **Adaptive resolution streaming**

After adaptive resolution streaming is enabled, the resolution is automatically reduced to ensure the smoothness and quality of video streams in poor network conditions. The adaptive resolution streaming feature is not supported by all players. If you need to use this feature, we recommend that you use ApsaraVideo Player. The following code provides an example:

```
mAlivcLivePushConfig.setEnableAutoResolution(true); // Enable adaptive resolution streaming. The defa
ult value is false.
```

Note:

Adaptive resolution streaming is supported only when the AlivcQualityModeEnum parameter is set to the quality-first or smoothness-first mode.

- **Retouching**

i. To integrate the retouching feature, you must import the retouching resources by using Android Archive (AAR) files.

```
com.aliyun.maliang.android:queen:1.1.2_client-alivc-pro
```

You can integrate the following features that are contained in the demo:

File or folder	Description
beauty	The abstract class of retouching.

File or folder	Description
beautyui	The UI widgets of retouching.
queenbeauty	The Queen retouching components.
Imageutil	The Imageutil public class that is used to manage the images for retouching.

ii. In addition, you must set two callbacks in the code:

```
/**
* The callbacks that are used for facial recognition.
**/
mAlivcLivePusher.setCustomDetect(new AlivcLivePushCustomDetect() {
  @Override
  public void customDetectCreate() {
    Log.d(TAG, "customDetectCreate start");
    initBeautyManager();
    Log.d(TAG, "customDetectCreate end");
  }
  @Override
  public long customDetectProcess(long data, int width, int height, int rotation, int format, long extr
a) {
    Log.d(TAG, "customDetectProcess start: data ptr:" + data + ",width:" + width + ",height:" + height
+ "," + format + "," + rotation);
    if (mBeautyManager != null) {
      mBeautyManager.onDrawFrame(data, BeautyImageFormat.kNV21, width, height, 0, mCameraId
);
      Log.d(TAG, "keria: " + mCameraId);
    }
    Log.d(TAG, "customDetectProcess end");
    return 0;
  }
  @Override
  public void customDetectDestroy() {
    Log.d(TAG, "customDetectDestroy start");
    destroyBeautyManager();
    Log.d(TAG, "customDetectDestroy end");
  }
});
/**
* The callbacks that are used for retouching.
**/
mAlivcLivePusher.setCustomFilter(new AlivcLivePushCustomFilter() {
  @Override
  public void customFilterCreate() {
    Log.d(TAG, "customFilterCreate start");
    initBeautyManager();
    Log.d(TAG, "customFilterCreate end");
  }
  @Override
  public void customFilterUpdateParam(float fSkinSmooth, float fWhiten, float fWholeFacePink, floa
t fThinFaceHorizontal, float fCheekPink, float fShortenFaceVertical, float fBigEye) {
```

```
    }
    @Override
    public void customFilterSwitch(boolean on) {
    }
    @Override
    public int customFilterProcess(int inputTexture, int textureWidth, int textureHeight, long extra) {
        Log.d(TAG, "customFilterProcess start: textureId" + inputTexture + ",width:" + textureWidth + ",h
eight:" + textureHeight);
        int ret = mBeautyManager != null ? mBeautyManager.onTextureInput(inputTexture, textureWidth
, textureHeight) : inputTexture;
        Log.d(TAG, "customFilterProcess end, textureId:" + ret);
        Log.d(TAG, "keria1: " + mCameraId);
        return ret;
    }
    @Override
    public void customFilterDestroy() {
        destroyBeautyManager();
    }
});
```

Note:

If you need to access a third-party retouching library, you can set the setCustomDetect and setCustomFilter callbacks in the code.

- The data values returned by the customDetectProcess callback that includes the long data, int width, int height, int rotation, int format, and long extra parameters serve as the pointer for querying data. The third-party retouching library can identify or process the returned data.
- The values of the inputTexture parameter returned by the customFilterProcess callback that includes the int inputTexture, inttextureWidth, int textureHeight, and long extra parameters are the image textures that can be processed by the third-party retouching library. If you need to return a processed texture, the ID of the processed texture is returned. Otherwise, the ID of the original texture is returned.

- **Ingest of images**

  Push SDK for Android supports the ingest of an image when your app is switched to the background or the bitrate is low. This enhances the user experience. When your app is switched to the background, stream ingest is paused by default. In this case, only an image and audio streams can be ingested. For example, you can ingest an image that displays a message reminding the leave of the streamer, The following code provides an example:

  ```
  mAlivcLivePushConfig.setPausePushImage("The path of the specified image for background streaming in the PNG format"); // Specify the image for background stream ingest.
  ```

  In addition, you can specify a static image for stream ingest in poor network conditions. After that, the specified image is ingested when the bitrate is low. This avoids stuttering. The following code provides an example:

  ```
  mAlivcLivePushConfig.setNetworkPoorPushImage("The path of the image that is ingested for poor network conditions"); // Specify the image for stream ingest in poor network conditions.
  ```

- **Watermarks**

Push SDK for Android allows you to add one or more watermarks in the PNG format. The following code provides an example:

```
mAlivcLivePushConfig.addWaterMark(waterPath,0.1,0.2,0.3); // Add a watermark.
```

Note:

- The values of the coordX, coordY, and width parameters are relative. For example, a value of 0.1 for the CoordX parameter indicates that the left edge of the watermark is displayed at the 10% position on the x-axis of the streaming image. Therefore, if the stream ingest resolution is 540 × 960, the value for the watermarkCoordX parameter is 54.
- The height of the watermark is calculated based on the input width in a proportional aspect ratio.
- If you want to add a text watermark, you can transform the text into an image and then call the addWaterMark interface to add the image as a watermark.

- **Preview mode**

```
mAlivcLivePushConfig.setPreviewDisplayMode(AlivcPreviewDisplayMode.ALIVC_LIVE_PUSHER_PREVIEW_
ASPECT_FIT);
```

Note:

- AlivcPreviewDisplayMode.ALIVC_LIVE_PUSHER_PREVIEW_SCALE_FILL: Specify the video to fill the entire preview view. If the aspect ratio of the video does not match the view, the preview image is deformed.
- AlivcPreviewDisplayMode.ALIVC_LIVE_PUSHER_PREVIEW_ASPECT_FIT: Specify the initial aspect ratio of the video when the preview is displayed. If the aspect ratio of the video does not match the view, black edges appear on the preview view.
- AlivcPreviewDisplayMode.ALIVC_LIVE_PUSHER_PREVIEW_ASPECT_FILL: Change the aspect ratio of the video to fit for the preview view. If the aspect ratio of the video does not match the view, the video is cropped to fit the preview view.

The preceding three preview modes do not affect stream ingest.

Use AlivcLivePusher

AlivcLivePusher is the core class of Push SDK for Android. This class provides parameters for video preview, stream ingest callback, and stream ingest control. You can also use this class to modify parameters during stream ingest.

- Execute the try-catch statement to manage exceptions if you use these interfaces.
- You must follow the specified sequence to call these interfaces. An incorrect sequence may cause an error.

This section provides sample code for specific configurations.

- **Initialization**

Use the init method to initialize the configured stream ingest parameters. The following code provides an example:

```
AlivcLivePusher mAlivcLivePusher = new AlivcLivePusher();
mAlivcLivePusher.init(mContext, mAlivcLivePushConfig);
```

Note:

AlivcLivePusher does not support multiple instances. Therefore, Push SDK for Android provides a destroy method for each init method.

- **Register stream ingest callbacks**

  Stream ingest callbacks are grouped into Info, Error, and Network callbacks.

  - Info: the callbacks that are used for notifications and status detection.
  - Error: The callbacks that are used when errors occur.
  - Network: The callbacks that are used to manage network services.

  When an event occurs, the corresponding callback is triggered to notify you. The following code provides an example:

```
/**
 * Configure stream ingest errors.
 *
 * @param errorListener Error listener.
 */
mAlivcLivePusher.setLivePushErrorListener(new AlivcLivePushErrorListener() {
 @Override
 public void
onSystemError(AlivcLivePusher livePusher, AlivcLivePushError error) {
    if(error != null) {
       // Add UI notifications or custom error solutions.
    }
 }
 @Override
 public void onSDKError(AlivcLivePusher
livePusher, AlivcLivePushError error) {
    if(error != null) {
       // Add UI notifications or custom error solutions.
    }
 }
});
/**
* Configure notification events for stream ingest.
*
* @param infoListener Notification listener.
*/
mAlivcLivePusher.setLivePushInfoListener(new AlivcLivePushInfoListener() {
 @Override
 public void
onPreviewStarted(AlivcLivePusher pusher) {
    // Indicates the start of preview.
 }
 @Override
 public void
onPreviewStoped(AlivcLivePusher pusher) {
    // Indicates the end of preview.
 }
 @Override
 public void
onPushStarted(AlivcLivePusher pusher) {
    // Indicates that stream ingest starts.
```

```
    }
    @Override
    public void onPushPauesed(AlivcLivePusher
pusher) {
        // Indicates that stream ingest is paused.
    }
    @Override
    public void
onPushResumed(AlivcLivePusher pusher) {
        // Indicates that stream ingest is resumed.
    }
    @Override
    public void
onPushStoped(AlivcLivePusher pusher) {
        // Indicates that stream ingest ends.
    }
    @Override
    public void onPushRestarted(AlivcLivePusher
pusher) {
        // Indicates that stream ingest is restarted.
    }
    @Override
    public void
onFirstFramePreviewed(AlivcLivePusher pusher) {
        // Indicates that the first frame appears.
    }
    @Override
    public void onDropFrame(AlivcLivePusher
pusher, int countBef, int countAft) {
        // Indicates frame loss.
    }
    @Override
    public void
onAdjustBitRate(AlivcLivePusher pusher, int curBr, int targetBr) {
        // Indicates that the bitrate is adjusted.
    }
    @Override
    public void onAdjustFps(AlivcLivePusher
pusher, int curFps, int targetFps) {
        // Indicates that the frame rate is adjusted.
    }
});
/**
* Configure network notifications.
*
* @param infoListener Notification listener.
*/
mAlivcLivePusher.setLivePushNetworkListener(new AlivcLivePushNetworkListener()
{
    @Override
    public void
onNetworkPoor(AlivcLivePusher pusher) {
        // Indicates poor network conditions.
    }
```

```
 @Override
 public void
onNetworkRecovery(AlivcLivePusher pusher) {
    // Indicates the network is recovered.
 }
 @Override
 public void
onReconnectStart(AlivcLivePusher pusher) {
    // Indicates that reconnection starts.
 }
 @Override
 public void
onReconnectFail(AlivcLivePusher pusher) {
    // Indicates that reconnection failed.
 }
 @Override
 public void
onReconnectSucceed(AlivcLivePusher pusher) {
    // Indicates that reconnection succeeded.
 }
 @Override
 public void
onSendDataTimeout(AlivcLivePusher pusher) {
    // Indicates that data transmission times out.
 }
 @Override
 public void
onConnectFail(AlivcLivePusher pusher) {
    // Indicates that connection failed.
 }
});
/**
* Configure notifications for background music.
*
* @param pushBGMListener Background music listener.
*/
mAlivcLivePusher.setLivePushBGMListener(new AlivcLivePushBGMListener() {
 @Override
 public void onStarted() {
    // Indicates that the playing is started.
 }
 @Override
 public void onStoped() {
    // Indicates that the playback of background music is stopped.
 }
 @Override
 public void onPaused() {
    // Indicates that the playback of background music is paused.
 }
 @Override
 public void onResumed() {
    // Indicates that the playback of background music is resumed.
 }
 /**
  * Set the parameters of the playback progress
```

```
    Set the parameters of the playback progress.
     *
     * @param progress Playback progress.
     */
    @Override
    public void onProgress(final long
progress, final long duration) {
    @Override
    public void onCompleted() {
        // Indicates that the playback ends.
    }
    @Override
    public void onDownloadTimeout() {
        // Indicates that the player times out. The player is reconnected and seeks the previous playback positi
on.
    }
    @Override
    public void onOpenFailed() {
        // Indicates an invalid stream. The stream is inaccessible.
    }
});
```

- **Start preview**

  You can start preview after you initialize the livePusher object. Use the SurfaceView parameter for camera previews. The following code provides an example:

  ```
  mAlivcLivePusher.startPreview(mSurfaceView)// Start preview. You can also use the asynchronous interf
  ace startPushAsync to start stream ingest.
  ```

- **Start stream ingest**

  You can start stream ingest only after preview succeeds. Therefore, you must set the onPreviewStarted callback by adding the following code to the callback:

  ```
  mAlivcLivePusher.startPush(mPushUrl);
  ```

  Note:

  - Push SDK for Android provides the asynchronous interface startPushAsync to start stream ingest.
  - Push SDK for Android supports the URLs of the streams that are ingested over RTMP. For more information about how to obtain ingest URLs, see Ingest and streaming URLs.
  - Start stream ingest with a valid URL. Then, use a player, such as ApsaraVideo Player, FFplay, or VLC, to test stream pulling. For more information about how to obtain source URLs, see Ingest and streaming URLs.

- **Other stream ingest configurations**

  Push SDK for Android allows you to control stream ingest. For example, you can start, stop, restart, pause, and resume stream ingest, stop preview, and destroy stream ingest objects. You can add buttons as needed to perform these operations. The following code provides an example:

```
/*You can pause the ongoing stream ingest. If you pause the ongoing stream ingest, the system pauses th
e video preview and the video stream ingest at the last frame, and continues the ingest of audio-only stre
ams. */
mAlivcLivePusher.pause();
/*You can resume the paused stream ingest. Then, the system resumes the audio and video preview and t
he stream ingest. */
mAlivcLivePusher.resume();
/*You can stop the ongoing stream ingest. */
mAlivcLivePusher.stopPush();
/*You can stop the ongoing preview. However, this operation does not take effect for the ongoing stream
ingest. When the preview is stopped, the preview view is frozen at the last frame. */
mAlivcLivePusher.stopPreview();
/*You can restart stream ingest that is in the streaming status or when the method receives an error callb
ack. When an error occurs, you can only use this method or reconnectPushAsync to restart stream ingest.
You can also use the destroy method to destroy the stream ingest object. After you complete the precedi
ng operation, restart all ALivcLivePusher resources that are required for operations such as the preview a
nd stream ingest. */
mAlivcLivePusher.restartPush();
/*You can use this method in the streaming status or when the method receives callbacks caused by error
s in AlivcLivePusherNetworkDelegate. When an error occurs, you can only use this method or restartPush
to restart stream ingest. You can also use the destroy method to destroy the stream ingest object. After y
ou complete the preceding operation, you can restart the stream ingest over RTMP. */
mAlivcLivePusher.reconnectPushAsync();
/*After the stream ingest object is destroyed, the stream ingest and the preview are stopped, and preview
views are deleted. All resources related to AlivcLivePusher are destroyed. */
mAlivcLivePusher.destroy();
```

- Background music

  Push SDK for Android allows you to manage background music. For example, you can configure the playback of background music, and the audio mixing, noise reduction, in-ear monitoring, and muting features. You can call background music interfaces only after preview starts. The following code provides an example:

```
/*Start the playback of background music. */
mAlivcLivePusher.startBGMAsync(mPath);
/*Stop the playback of background music. To change the background music, call the interface that is used to start the playback of background music. You do not need to stop the playback of the current background music. */
mAlivcLivePusher.stopBGMAsync();
/*Pause the playback of the background music. You can call this interface only after the background music starts. */
mAlivcLivePusher.pauseBGM();
/*Resume the background music. You can call this interface only after the background music is paused. */
mAlivcLivePusher.resumeBGM();
/*Enable looping.*/
mAlivcLivePusher.setBGMLoop(true);
/*Configure noise reduction. When noise reduction is enabled, the system filters out non-vocal parts from collected audio. This feature may slightly reduce the volume of the human voice. Therefore, we recommend that you allow your users to determine whether to enable this feature. This feature is disabled by default.*/
mAlivcLivePusher.setAudioDenoise(true);
/*Configure in-ear monitoring. In-ear monitoring applies to the KTV scenario. When in-ear monitoring is enabled, you can hear your voice on your earphones during streaming. When in-ears monitoring is disabled, you cannot hear your voice on your earphones during streaming. This feature does not take effect if no earphones are detected. */
mAlivcLivePusher.setBGMEarsBack(true);
/*Configure audio mixing by adjusting the volumes of the background music and the human voice. */
mAlivcLivePusher.setBGMVolume(50); // Set the volume of the background music.
mAlivcLivePusher.setCaptureVolume(50); // Set the volume of the human voice.
/*Configure muting. If you enable this feature, background music and the human voice are muted. To separately mute background music or the human voice, call the interface that is used to configure audio mixing. */
mAlivcLivePusher.setMute(true);
```

- Camera operations

  You can perform camera-related operations only after you start preview in streaming status, paused status, or reconnecting status. For example, you can switch the camera, and configure the flash, the focal length, zooming, and the mirroring mode. If you do not start preview, the following interfaces are invalid. The following code provides an example:

```
/*Switch between the front and the rear cameras.*/
mAlivcLivePusher.switchCamera();
/*Enable or disable the flash. You cannot enable the flash for the front camera.*/
mAlivcLivePusher.setFlash(true);
/*Set the focal length to zoom in and out of images. Zooming ranges from [0 to getMaxZoom()]. */
mAlivcLivePusher.setZoom(5);
/*Configure manual focus. To configure manual focus, you must set the following two parameters: point, which specifies the coordinates of the focus point, and autoFocus, which is used to enable or disable auto focus. The autoFocus parameter is valid only when you call this interface. Other auto focus settings follo w the same settings. */
mAlivcLivePusher.focusCameraAtAdjustedPoint(x, y, true);
/*Configure auto focus.*/
mAlivcLivePusher.setAutoFocus(true);
/*Configure the mirroring mode. Mirroring-related interfaces are PushMirror and PreviewMirror. The Push Mirror interface is used to configure the mirroring mode for stream ingest. The PreviewMirror interface is used to configure the mirroring mode for preview. PushMirror is valid only for playback images. PreviewM irror is valid only for preview views. */
mAlivcLivePusher.setPreviewMirror(false);
mAlivcLivePusher.setPushMirror(false);
```

- External audio and video input

  Push SDK for Android allows you to import external audio and video sources for stream ingest. For example, you can ingest an audio or video file.

  - Configure the input of external audio and video sources in stream ingest settings. The following code provides an example:

    ```
    /**
    * AlivcImageFormat: indicates the format of the input video images.
    * AlivcSoundFormat: indicates the format of the input audio frames.
    * Other parameters: To specify the resolution, the audio sample rate, and the number of channels, set setResolution, setAudioSamepleRate, and setAudioChannels in config.
    * Note: To import custom video and audio streams, use interfaces such as inputStreamVideoData and i nputStreamAudioData.
    */
    mAlivcLivePushConfig.setExternMainStream(true,AlivcImageFormat.IMAGE_FORMAT_YUVNV12, AlivcSoundFormat.SOUND_FORMAT_S16);
    ```

- The following code provides an example on how to import external video data:

```
/**
* Import custom video streams.
*
* @param data The byte array of the video image.
* @param width The width of the video image.
* @param height The height of the video image.
* @param size The size of the video image.
* @param pts The presentation timestamp (PTS) of Video image, in microseconds.
* @param rotation The rotation angle of the video image.
* This interface does not control the time sequence. You must control the time sequence of input video frames.
* Note: Set the etExternMainStream(true,***) parameter in the config parameter when you use call this interface.
*/
mAlivcLivePusher.inputStreamVideoData(buffer, 720, 1280, 1280*720*3/2, System.nanoTime()/1000, 0);
/**
* Import custom video streams.
*
* @param dataptr The Native memory pointer for the video image.
* @param width The width of the video image.
* @param height The height of the video image.
* @param size The size of the video image.
* @param pts The PTS of the video image, in microseconds.
* @param rotation The roration angle of the video image.
* This interface does not control the time sequence. You must control the time sequence of input video frames by yourself.
* Note: Set setExternMainStream(true,***) in the config parameter when you call this interface.
*/
mAlivcLivePusher.inputStreamVideoPtr(long dataptr, int width, int height, int
size, long pts, int rotation);
```

- The following code provides an example on how to import external audio data:

```
/**
* Import custom audio data.
* @param data The byte array of audio data.
* @param size
* @param pts The PTS of audio, in microseconds.
* This interface does not control the time sequence. You must control the time sequence of input audio and audio frames by yourself.
*/
mAlivcLivePusher.inputStreamAudioData(buffer, 2048, System.nanoTime()/1000);
/**
* Import custom audio data.
* @param dataptr The native memory pointer for audio data.
* @param size
* @param pts PTS of audio, in microseconds.
* This interface does not control the time sequence. You must control the time sequence of input audio frames by yourself.
*/
AlivcLivePusher.inputStreamAudioPtr(long dataPtr, int size, long pts);
```

- **Animated stickers**

  Push SDK for Android allows you to add animated stickers as watermarks to live streams.

  - To make an animated sticker, you can modify the materials in the demo. Make a sequence frame image for the animated sticker, open the config.json file, and set the following parameters:

    ```
    "du": 2.04, // The duration for playing the animated sticker once.
    "n": "qizi", // The name of the animated sticker. Make sure the name of the folder for making the animated sticker is the same as the name of the sticker. The name of the sticker contains the name followed by the sequence number, such as qizi0.
    "c": 68.0, // The number of animation frames, which is the number of images included in an animated sticker.
    "kerneframe": 51, //The keyframe. Specify an image as the keyframe. For example, the 51st frame is specified as the keyframe in the demo. Make sure that the specified frame exists.
    "frameArry": [
      {"time":0,"pic":0},
      {"time":0.03,"pic":1},
      {"time":0.06,"pic":2},
      ],
    // The parameters of the animated sticker. The preceding settings indicate that the first frame (qizi0) is displayed in 0 seconds after the start, and the second frame (qizi1) is displayed in 0.03 seconds after the start.In this way, all frames are specified in the animation.
    ```

    Note:

    Set other fields as described by the .json file in the demo.

  - The following code provides an example on how to add an animated sticker:

    ```
    /**
    * Add an animated sticker.
    * @param path The path of the animated sticker, which must contain config.json.
    * @param x The starting position on the x-axis. Value range: (0-1.0f).
    * @param y The starting position on the y-axis. Value range: (0-1.0f).
    * @param w The width of the screen. Value range: (0-1.0f).
    * @param h The height of the screen. Value range: (0-1.0f).
    * @return id The ID of the sticker. You must specify the sticker ID if you want to remove a sticker.
    */
    mAlivcLivePusher.addDynamicsAddons("Path of the sticker", 0.2f, 0.2f, 0.2f, 0.2f);
    ```

  - The following code provides an example on how to remove an animated sticker:

    ```
    mAlivcLivePusher.removeDynamicsAddons(int id);
    ```

- **Debugging tools**

  DebugView is a UI debugging tool that allows you to diagnose issues. DebugView provides a draggable and always-on-top window for debugging. For example, you can query the logs of stream ingest, monitor the metrics for stream ingest performance in real time, and generate line charts of main performance metrics. The following code provides an example:

  ```
  AlivcLivePusher.showDebugView(context); // Show DebugView.
  AlivcLivePusher.hideDebugView(context); // Hide DebugView.
  ```

> **Note** In the current release, do not call an interface that invokes DebugView.

- **Other interfaces**

```
/*In custom mode, you can change the minimum bitrate and the maximum bitrate in real time. */
mAlivcLivePusher.setTargetVideoBitrate(800);
mAlivcLivePusher.setMinVideoBitrate(400);
/*Specify whether auto-focus is supported.*/
mAlivcLivePusher.isCameraSupportAutoFocus();
/*Specify whether the flash is supported. * /
mAlivcLivePusher.isCameraSupportFlash();
/*Obtain the status of stream ingest.*/
mAlivcLivePusher.isPushing();
/*Obtain the ingest URL.*/
mAlivcLivePusher.getPushUrl();
/*Obtain the debugging information about stream ingest performance. For more information about the p
arameters of stream ingest performance, see API references or comments in the code.*/ */
mAlivcLivePusher.getLivePushStatsInfo();
/*Obtain the version number of Push SDK for Android. */
 mAlivcLivePusher.getSDKVersion();
/*Set a log level to filter debugging information.*/
mAlivcLivePusher.setLogLevel(AlivcLivePushLogLevelAll);
/*Obtain the status of Push SDK for Android.*/
AlivcLivePushStats getCurrentStatus();
/*Obtain the last error code. If no error code is returned, the output displays ALIVC_COMMON_RETURN_S
UCCESS.*/
AlivcLivePushError getLastError();
```

Screen recording

- **Configure the screen recording mode**

  Screen recording supports the following modes:

  - Record screen with the camera disabled: When you request screen recording permission, call the setMediaProjectionPermissionResultData interface in AlivcLivePushConfig and specify the Intent object returned by MediaProjectionManager.createScreenCaptureIntent as the parameter.
  - Record screen with the camera enabled and enable camera preview on the streamer side: Viewers can view the content that is recorded with the camera.
    a. When you request the screen recording permission, call the setMediaProjectionPermissionResultData interface in AlivcLivePushConfig and specify the Intent object returned by MediaProjectionManager.createScreenCaptureIntent as the parameter.
    b. Pass surfaceView into the StartCamera interface and call this interface.

- Record screen with the camera enabled and disable camera preview on the streamer side: Viewers can still view the content that is recorded from the camera.
  a. When you request screen recording permission, call the setMediaProjectionPermissionResultData interface in AlivcLivePushConfig and specify the Intent object returned by MediaProjectionManager.createScreenCaptureIntent as the parameter.
  b. Call the StartCamera interface without passing surfaceView into the method.
  c. Call the startCameraMix interface to tune the position of the camera view on the viewer side.

- **Enable screen recording**

  Screen recording uses MediaProjection. When you request screen recording permission, call the setMediaProjectionPermissionResultData interface in AlivcLivePushConfig and specify the Intent object returned by MediaProjectionManager.createScreenCaptureIntent as the parameter. By default, the camera is disabled during screen recording. The following code provides an example on how to configure this setting in the configuration:

  ```
  mAlivcLivePushConfig.setMediaProjectionPermissionResultData(resultData)
  ```

- **Camera preview**

  You can call the camera preview interface after you have enabled screen recording. The following code provides an example:

  ```
  mAlivcLivePusher.startCamera(surfaceView); // Enable camera preview.
  mAlivcLivePusher.stopCamera(); // Disable camera preview.
  ```

  Note:

  - We recommend that you set the aspect ratio of the surfaceView to 1:1 for the camera preview in screen recording mode. In this way, you do not need to adjust the aspect ratio of surfaceView when you rotate your screen.
  - If you do not set the aspect ratio of the surfaceView to 1:1 for the camera preview, you must adjust the aspect ratio, disable camera preview, and then enable camera preview.
  - If the streamer does not require preview, set the surfaceview parameter to null.

- **Stream mixing**

  You can enable this feature when the streamer does not require camera preview but viewers do. This feature is typically used in game live streaming. When the streamer does not want to stream the game content, the streamer can display the camera view on top of the game content. The following code provides an example:

  ```
  /**
  * @param x The starting position on the x-axis. Value range: (0-1.0f).
  * @param y The starting position on the y-axis. Value range: (0-1.0f).
  * @param w The width of the screen. Value range: (0-1.0f).
  * @param h The height of the screen. Value range: (0-1.0f).
  * @return
  */
  mAlivcLivePusher.startCameraMix(x, y, w, h); // Enable camera and screen stream mixing.
  mAlivcLivePusher.stopCameraMix(); // Disable camera and screen stream mixing.
  ```

- **Set screen rotation**

In screen recording mode, you can configure the screen to rotate between portrait mode and landscape mode. The following code provides an example:

```
mAlivcLivePusher.setScreenOrientation(0);
```

Note:

When you change the orientation of the screen, you must enable OrientationEventListener at the application layer and pass the orientation setting to this interface.

- **Privacy settings**

  This feature allows the streamer to protect privacy during live stream recording. The streamer can enable this feature when the streamer enters a password and then disable the feature. The following code provides an example:

  ```
  mAlivcLivePusher.pauseScreenCapture(); // Enable privacy protection.
  mAlivcLivePusher.resumeScreenCapture(); // Disable privacy protection.
  ```

  Note:

  If you set the setPausePushImage parameter in config, the specified image is displayed during the pause of live stream recording. If you do not set this parameter, the last frame is displayed during the pause of live stream recording.

Note

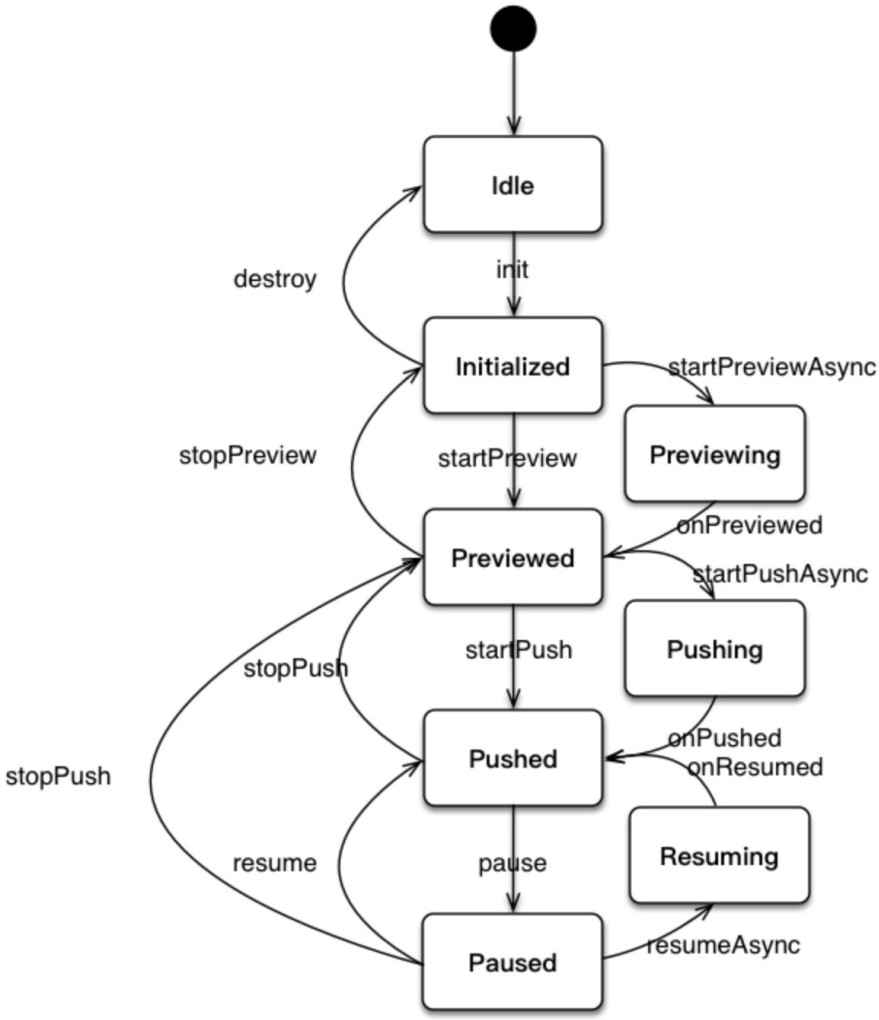
- Obfuscation rules

  Check the obfuscation configurations. Make sure that the package names of Push SDK for iOS are removed from the obfuscation list.

  ```
  -keep class com.alivc.** { *;}
  ```

- API call
  - You can call both synchronous and asynchronous interfaces. However, we recommend that you avoid using asynchronous interfaces because this consumes the resources of the main thread.
  - Push SDK for Android throws exceptions when you fail to call the required interface or call interfaces in an invalid order. You must handle try catch exceptions to prevent unexpected quits.

- The following figure shows the call procedure to be followed.



- Package size
  - SDK size: The architectures of armeabi-v7a and arm64-v8a are supported. The total size of Push SDK for Android is 8.7 MB.
  - After you integrate Push SDK for Android, the size of the APK file increases by 7.4 MB.
- Limits
  - You must configure screen rotation before stream ingest. You cannot rotate the screen during stream ingest.
  - You must disable auto-screen rotation for stream ingest in landscape mode.
  - In hardware encoding mode, the output resolution must be multiples of 16 due to the compatibility of the encoder. For example, if you set the resolution to 540p, the output resolution is 544 x 960. You must scale the screen size of the player based on the output resolution to prevent black edges.
- Version upgrade instruction

  Push SDK for Android is updated to V4.1.0. Make sure the ApsaraVideo Player SDK is integrated.

  AliyunPlayer is updated to V5.4.1-full. To support RTS playback, you must use AlivcArtc V5.4.1.

```
implementation 'com.aliyun.sdk.android:AliyunPlayer:5.4.1-full'
implementation 'com.aliyun.sdk.android:AlivcArtc:5.4.1'
```

4.5. Handling of exceptions and special scenarios

This topic describes the exceptions and special scenarios that you may encounter when you use Push SDK for Android. This topic also describes how to handle such exceptions and special scenarios.

Handle AlivcLivePushErrorListener callbacks

- When you receive an onSystemError callback message, which indicates a system error, stop live streaming.
- When you receive an onSDKError callback message, which indicates an SDK error, destroy the current AlivcLivePusher object and then create another one. Alternatively, you can call the restartPush or restartPushAsync method to restart the current AlivcLivePusher object.
- You must pay special attention to the fired callbacks that are related to microphone and camera permission issues. The ALIVC_PUSHER_ERROR_SDK_CAPTURE_CAMERA_OPEN_FAILED error code indicates that the mobile app requires permissions to use the microphone. The ALIVC_PUSHER_ERROR_SDK_CAPTURE_MIC_OPEN_FAILED error code indicates that the mobile app requires permissions to use the camera.

Handle AlivcLivePushNetworkListener callbacks

- An onNetworkPoor callback message is sent to you when the network speed is slow. The message indicates that the network cannot support stream ingest, but the stream is still being ingested and not interrupted. When the network speed resumes to the normal level, an onNetworkRecovery callback message is sent to you. At this point, you can configure custom business logic. For example, you can notify your users of network recovery in a visualized manner on the mobile app UI.
- The onConnectFail, onReconnectError, or onSendDataTimeout callback is fired when a specific network error occurs. In this case, destroy the current AlivcLivePusher object and then create another one. Alternatively, you can call the reconnectAsync method to reconnect to the network. We recommend that you test the network connectivity and the ingest URL before you perform the reconnection operation.
- The SDK fires the onReconnectStart callback to reconnect to the network when the SDK receives a reconnection request. The reconnection request may be issued by the SDK or the developer that calls the reconnectAsync method. To reconnect to the network, the mobile app reinitiates a connection to the Real-Time Messaging Protocol (RTMP) server.

  After the RTMP connection is established, an onReconnectSuccess callback message is sent to you. However, the message does not mean that the mobile app resumes stream ingest. Stream ingest may fail to be resumed if a network issue persists. In this case, the SDK keeps reconnecting to the RTMP server.
- An onPushURLAuthenticationOverdue callback message indicates the expiration of the ingest URL authentication. The SDK sends such a message to you only after you enable ingest URL authentication. An ingest URL contains the auth_key field. This callback is fired 1 minute before the ingest URL expires. After you receive such a message, you must specify a new ingest URL to ensure that the ingest process is not interrupted due to the expiration of the original ingest URL. Sample code:

```
String onPushURLAuthenticationOverdue(AlivcLivePusher pusher) {
  return "New ingest URL rtmp://";
}
```

Handle AlivcLivePusherBGMListener callbacks

- The onOpenFailed callback is fired when the background music (BGM) fails to be played. In this case, check whether the audio file and the file path specified for the startBGMAsync method are valid. Then, call the startBGMAsync method to replay the BGM.
- The onDownloadTimeout callback is fired when the BGM playback times out. This issue generally occurs when the BGM is specified by using an online URL. This callback instructs the streamer to check the status of the network. You can call the startBGMAsync method to replay the BGM.

Handle network disconnection

- Short-period network disconnection and network switchover: The SDK attempts to reconnect to the network when a network fluctuation or a network switchover occurs. You can use the AlivcLivePushConfig class to specify the reconnection timeout period and the maximum number of reconnection attempts allowed. After the SDK reconnects to the network, stream ingest is resumed. If you use ApsaraVideo Player, we recommend that you perform a reconnection operation 5 seconds after you receive a timeout notification.
- Long-period network disconnection: The SDK fails to reconnect to the network if a reconnection request times out or the number of reconnection attempts exceeds the upper limit. In this case, the onReconnectError callback is fired. After the network is recovered, call the reconnectAsync method to reconnect the SDK to the network. You must also reconnect the SDK to ApsaraVideo Player.
  - We recommend that you externally monitor the network.
  - Use the server to handle communication failures between the streamer and player. For example, when network disconnection occurs on the streamer, the server receives a callback message of stream ingest interruption from Alibaba Cloud CDN. The server pushes the callback message to the player. Then, the player takes relevant measures to handle the stream ingest interruption. The server uses the same procedure to resume stream ingest.
  - Stop and then restart ApsaraVideo Player to reconnect the player to the ingest URL. To achieve this objective, call the stop and prepareAndPlay methods in sequence.

    ```
    mPlayer.stop();
    mPlayer.prepareAndPlay(mUrl);
    ```

    > **Note** For more information about ApsaraVideo Player, see Implementation.

Switch the mobile app to the background or lock the screen

- When the mobile app is switched to the background or the screen is locked, you can call the pause() or resume() method of the AlivcLivePusher class to pause or resume stream ingest.
- For non-system audio and video calls, the SDK continues recording and ingesting the audio stream. You can call the mAlivcLivePusher.setMute() method to specify whether to enable audio recording when the mobile app is switched to the background or your screen is locked. You can set the setMute field to true or false.

Set bitrates for ingested streams

The SDK supports dynamic bitrate conversion. You can use the AlivcLivePushConfig class to change the default bitrate. Different services require different video quality. The resolution, smoothness, and definition of an output video vary based on the bitrate of the ingested stream.

- Video definition: The larger the bitrate of the ingested stream is, the higher the video definition is. We recommend that you specify a large bitrate to ensure the playback of high-definition videos.
- Video smoothness: The larger the bitrate of the ingested stream is, the higher network bandwidth is required. A large bitrate with low network bandwidth may affect the smoothness of the videos.

5.iOS Push SDK

5.1. Update Push SDK from V4.0.2 to V4.1.0

This topic describes how to update Push SDK for iOS from V4.0.2 to V4.1.0.

Procedure

Remove the class libraries and resource files of Push SDK V4.0.2 from your project. Then, add the class libraries and resource files of Push SDK V4.1.0 to your project.

1. In the libs directory, use *AlivcLivePusher.framework* and *AlivcLibRtmp.framework* of Push SDK V4.1.0 to replace *AliLivesdk.framework* of Push SDK V4.0.2.
2. Update the ApsaraVideo Player SDK to AliyunPlayer. For more information, see SDK and the SDK used in the demo.
3. If you want to implement the Queen smart retouching feature in your app, view the usage instructions in the demo. This facilitates the integration of the Queen SDK and UI.
4. Modify specific interfaces, as described in the Comparison of key interfaces section of this topic.
5. Modify specific interfaces for the major process, as described in the Changes in the interfaces for major process section of this topic.

Changes in the interfaces for major process

1. **Create an engine**

AliLiveEngine in V4.0.2	AlivcLivePusher in V4.1.0

AliLiveEngine in V4.0.2	AlivcLivePusher in V4.1.0

// Create an AliLiveConfig object.
Import a header file. #import <AliLiveSdk/AliLiveSdk.h>
Create an AliLiveEngine object. AliLiveConfig *config = [[AliLiveConfig alloc] init];
config.videoProfile = AliLiveVideoProfile_540P;
config.videoFPS = 20;
myConfig.pauseImage = [UIImage imageNamed:@"background_img.png"];
myConfig.accountID = @"";
AliLiveEngine *engine = [[AliLiveEngine alloc] initWithConfig:myConfig];
[engine setAudioSessionOperationRestriction:AliLiveAudioSessionOperationRestrictionDeactivateSession];
[engine setRtsDelegate:self];
[engine setStatusDelegate:self];
	Run the following command to import the header file for the view controller that requires the streamer: `#import <AlivcLivePusher/AlivcLivePusherHeader.h>`. The following code provides an example:
	// Initialize the AlivcLivePushConfig class. You can also use initWithResolution to initialize the class.
	AlivcLivePushConfig *config = [[AlivcLivePushConfig alloc] init];
	// The default resolution is 540p. The supported maximum resolution is 720p.
	config.resolution = AlivcLivePushResolution540P;
	// We recommend that you set the frame rate to 20 fps.
	config.fps = AlivcLivePushFPS20;
	// Enable adaptive bitrate streaming. The default value is true.
	config.enableAutoBitrate = true;
	// The default keyframe interval is 2. Higher keyframe intervals cause higher latency. We recommend that you set the keyframe interval to a number between 1 and 2.
	config.videoEncodeGop = AlivcLivePushVideoEncodeGOP_2;
	// The unit of the reconnection interval is milliseconds. The default reconnection interval is 2 seconds. The reconnection interval must be 1 second or greater. We recommend that you use the default value.
	config.connectRetryInterval = 2000;
	// The default value of the PreviewMirror parameter is false. We recommend that you use the default value.
	config.previewMirror = false;
	// The default orientation is portrait. You can change the direction to landscape left or landscape right.
	config.orientation = AlivcLivePushOrientationPortrait;

2. **create a preview**

V4.0.2	V4.1.0
Start the preview.self.engine startPreview:self.renderView];	The following modes are supported for the preview: ○ ALIVC_LIVE_PUSHER_PREVIEW_SCALE_FILL: Fill the entire window. The preview is distorted when the video aspect ratio and the window aspect ratio are different. ○ ALIVC_LIVE_PUSHER_PREVIEW_ASPECT_FIT: Maintain the video aspect ratio during the preview. The blank edge is displayed in black by default when the video aspect ratio and the window aspect ratio are different. ○ ALIVC_LIVE_PUSHER_PREVIEW_ASPECT_FILL: Split the video to adapt to the window aspect ratio when the video aspect ratio and the window aspect ratio are different. You can specify these modes in AlivcLivePushConfig. You can also use the setpreviewDisplayMode interface during preview and stream ingest. [self.livePusher startPreview:self.view];

3. Start stream ingest

V4.0.2	V4.1.0
[self.engine startPushWithURL:self.pushUrl];	[self.livePusher startPushWithURL:@"Ingest URL for test (rtmp://.....)"];

4. Stop stream ingest

V4.0.2	V4.1.0
[self.engine stopPush]; [self.engine stopPreview]; [self.engine destorySdk]; self.engine = nil;	[self.livePusher destory]; self.livePusher = nil; /*Obtain the status of stream ingest. */ AlivcLivePushStatus status = [self.livePusher getLiveStatus];

Comparison of key interfaces

- Basic interfaces

v4.0.2	v4.1.0	Description
getSdkVersion	getSdkVersion	Queries the version of Push SDK.
initWithConfig	initWithConfig	Creates a stream ingest object.
destorySdk	destroy	Destroys the stream ingest object.
○ setStatusDelegate ○ setRtsDelegate ○ setVidePreProcessDelegate ○ setDataStatsDelegate	○ AlivcPublisherViewDelegate ○ AlivcLivePusherInfoDelegate ○ AlivcLivePusherErrorDelegate	Sets the callback for the audio and video streams of a fan during stream ingest over Real-Time Communication (RTC). For more information, see AliLiveRtsDelegate to set video preprocessing callbacks. Sets the callback for parameters that are related to live streaming media.
setNetworkDelegate	AlivcLivePusherNetworkDelegate	Sets the callback for the network status during stream ingest.
setLogDirPath	N/A: Custom logs are written.	Sets the path for storing the log files of Push SDK. To prevent log loss, you must call this operation before other API operations. In addition, make sure that the specified path exists and is writable.
setLogLevel	N/A	Sets the log level.

- **Basic stream ingest interfaces**

V4.0.2	V4.1.0	Description
startPreview	startPreview	Starts the preview. This interface is called on the streamer side.
stopPreview	stopPreview	Stops the preview. This interface is called on the streamer side.
pausePush	pause	Stops the camera from collecting live streams and ingests standby streams. This interface is supported only for stream ingest over RTMP. You must call the startPush interface before the pausePush interface to prevent an invalid call order.
resumePush	resume()	Enables the camera to collect live streams again and stops ingesting standby streams. This interface is supported only for stream ingest over RTMP. You must call the pausePush interface before the resumePush interface to prevent an invalid call order.
startPush	startPushWithURL	Starts stream ingest.

V4.0.2	V4.1.0	Description
stopPush	stopPush	Stops stream ingest.
isPublishing	isPushing	Queries whether streams are being ingested.
getPublishUrl	getPushURL	Queries the current ingest URL.

- **Video-related interfaces**

V4.0.2	V4.1.0	Description
setPreviewMode	setpreviewDisplayMode	Sets the preview mode.
switchCamera	switchCamera	Switches between the front camera and the rear camera.
setCameraZoom	setZoom	Sets the zoom factor of the camera and specifies whether to enable the flash.
isCameraExposurePointSupported	setExposure	Queries whether an exposure point can be set for the camera.
setCameraFocusPoint	setAutoFocus	Sets the focus point of the camera.

- **Audio-related interfaces**

V4.0.2	V4.1.0	Description
setMute	setMute	Specifies whether the frames collected from the local audio are mute frames.
isAudioOnly	isAudioOnly	Queries whether the ingested streams are audio-only streams.
enableEarBack	setBGMEarsBack	Enables in-ear monitoring. To prevent echoes, we recommend that you enable in-ear monitoring after you insert the headset.
playBGM	startBGMAsync	Plays background music.
stopBGM	stopBGM	Stops playing background music.
pauseBGM	pauseBGM	Pauses the playback of background music.
resumeBGM	resumeBGM	Resumes the playback of background music.
setBGMVolume	setBGMVolume	Sets the volume of background music.

5.2. Demo compilation

This topic describes the requirements for the runtime environment of the Push SDK demo for iOS. This topic also describes how to compile the demo.

Environment requirements

Item	Requirement
iOS version	iOS 8.0 or later
Phone model	iPhone 5 or later
CPU architecture	ARM64 or ARMv7
Integration tool	Xcode 8.0 or later
bitcode	Disabled

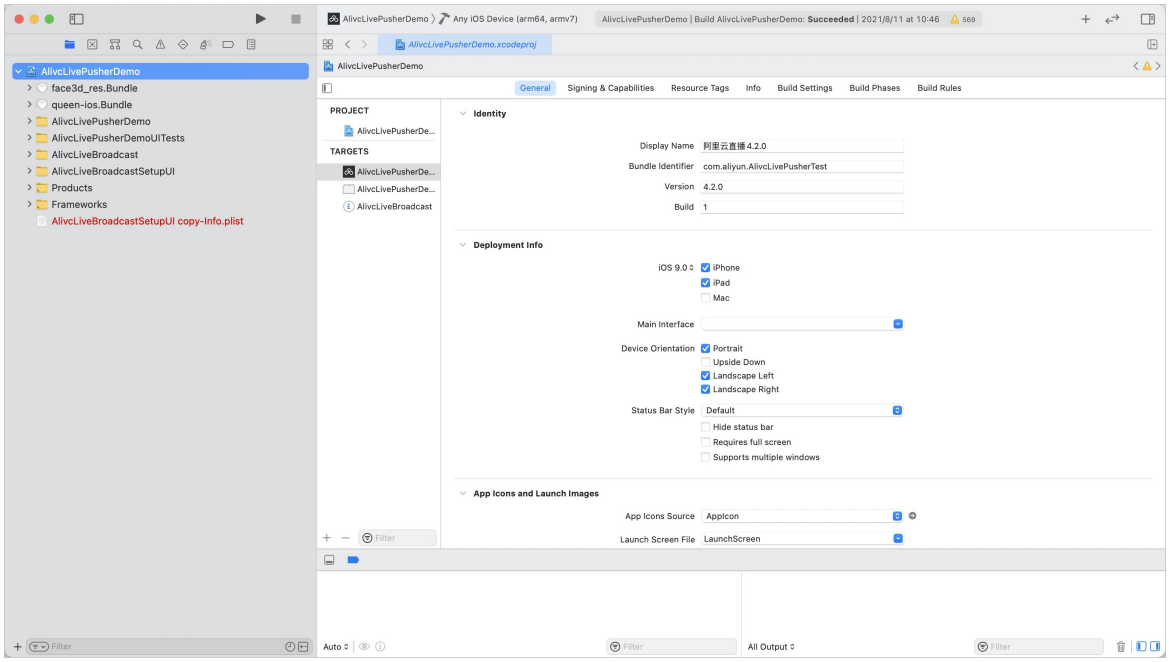
Run the demo

1. Download the demo package.

   Download the demo package of the required version by using the download URL provided in the SDK download and release notes topic.

2. Import the demo project.

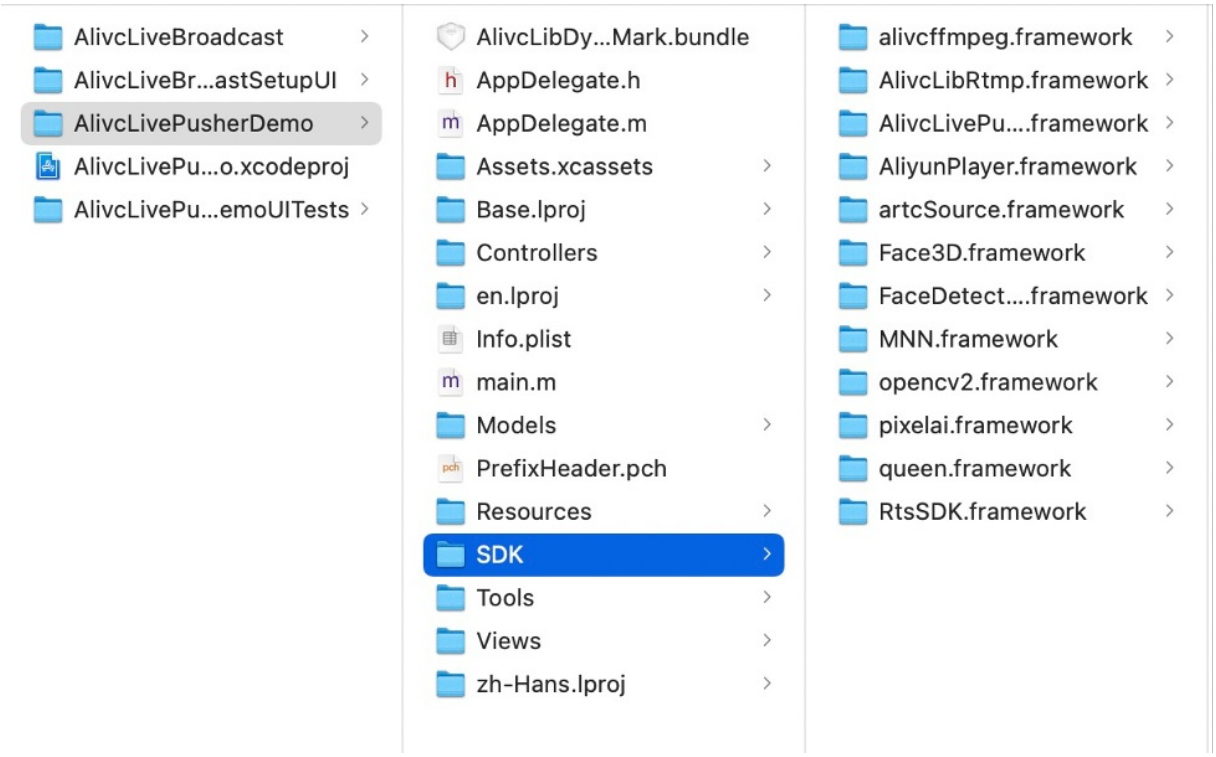
   Use Xcode to open the AlivcLivePusherDemo demo project.



3. Run the demo project to test the features.

You must specify a valid Real-Time Messaging Protocol (RTMP) URL to start stream ingest. After the stream is ingested, you can use the ApsaraVideo Player SDK, FFPlay, or VLC to watch the stream.

Directory structure of the demo for iOS



The following table describes the files contained in the SDK folder.

File	Description
AlivcLivePusher.framework	Push SDK.
AlivcLibRtmp.framework	RTMP stream ingest SDK.
queen.framework	Library for retouching.
• FaceDetection.framework • MNN.framework • Face3D.framework • AlivcLibFace.framework • opencv2.framework • pixelai.framework	Libraries for facial recognition.
AlivcLibFaceResource.bundle	Resource file for facial recognition.
AliyunPlayer.framework	Library of ApsaraVideo Player.
• alivcffmpeg.framework • artcSource.framework	Dependent third-party libraries of ApsaraVideo Player.
RtsSDK.framework	RTS stream ingest SDK.

File	Description
AlivcLibArtp.framework and AlivcLibBeauty.framework	Libraries for stream ingest required by SDK integration.

5.3. SDK integration

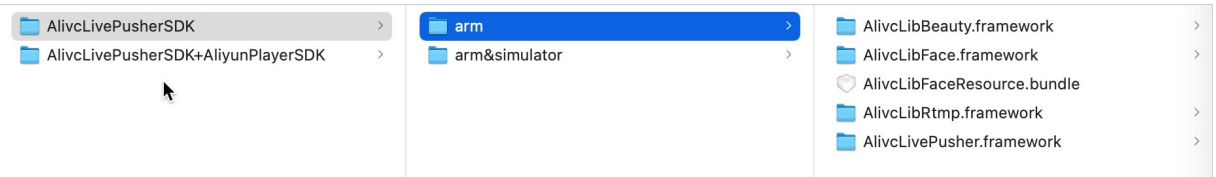
This topic describes how to integrate Push SDK for iOS.

Environment requirements

Item	Requirement
iOS version	iOS 8.0 or later
Phone model	iPhone 5 or later
CPU architecture	ARM64 or ARMv7
Integration tool	Xcode 8.0 or later
bitcode	Disabled

Download the SDK package

Download the SDK package of the required version by using the download URL provided in the SDK download and release notes topic. Push SDK for iOS is contained in the AlivcLivePusher folder, as shown in the following figure.



The following table describes the differences among the subfolders.

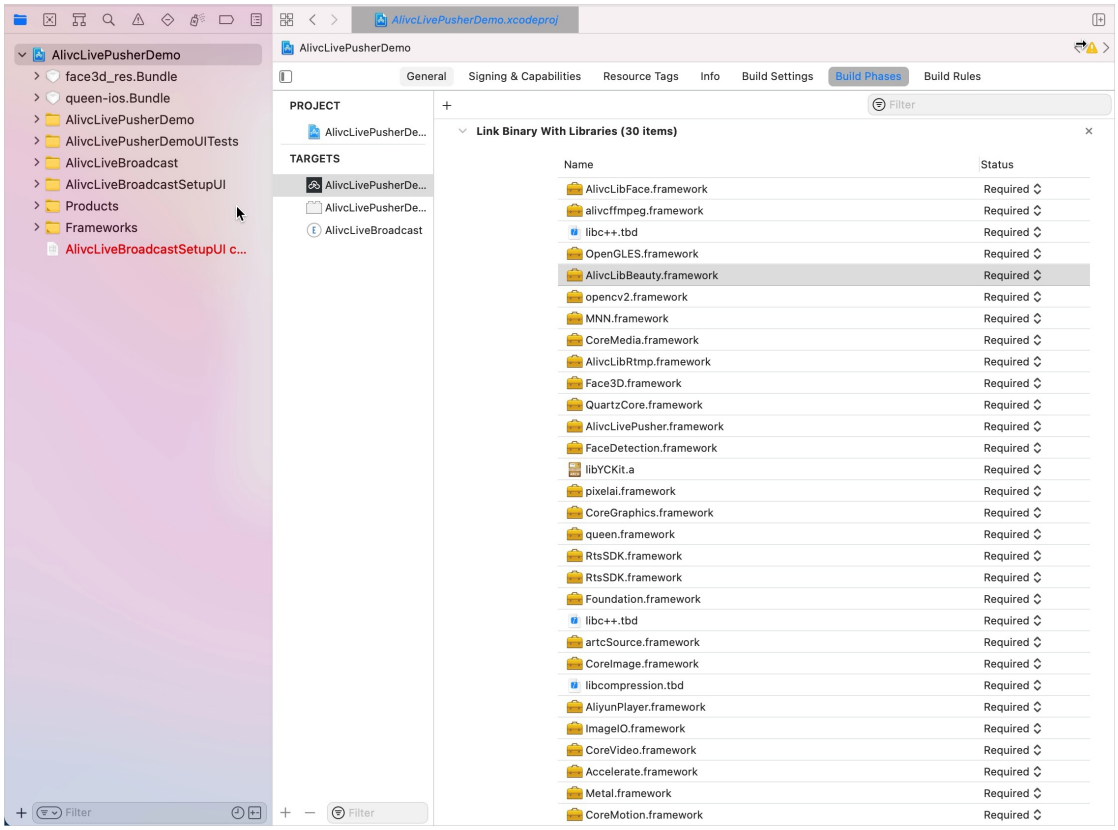
Subfolder directory	Description
AlivcLivePusherSDK/arm	The edition of Push SDK that excludes ApsaraVideo Player and applies only to arm architectures.
AlivcLivePusherSDK/arm&simulator	The edition of Push SDK that excludes ApsaraVideo Player and applies to both arm and simulator architectures.
AlivcLivePusherSDK+AliyunPlayerSDK/arm	The edition of Push SDK that includes ApsaraVideo Player and applies only to arm architectures.
AlivcLivePusherSDK+AliyunPlayerSDK/arm&simulator	The edition of Push SDK that includes ApsaraVideo Player and applies to both arm and simulator architectures.

> **Note**
>
> - The SDK provides the background music (BGM) feature. If you need to use the BGM feature, use an edition that includes ApsaraVideo Player. Otherwise, use an edition that excludes ApsaraVideo Player.
> - AlivcLibFaceResource.bundle is a resource file for facial recognition. If you need advanced features such as retouching, you must import this file to your project.
> - An edition that applies only to arm architectures supports debugging on physical devices. An edition that applies to both arm and simulator architectures supports debugging on physical devices and simulators. You must use an edition that applies only to arm architectures to publish your project online.

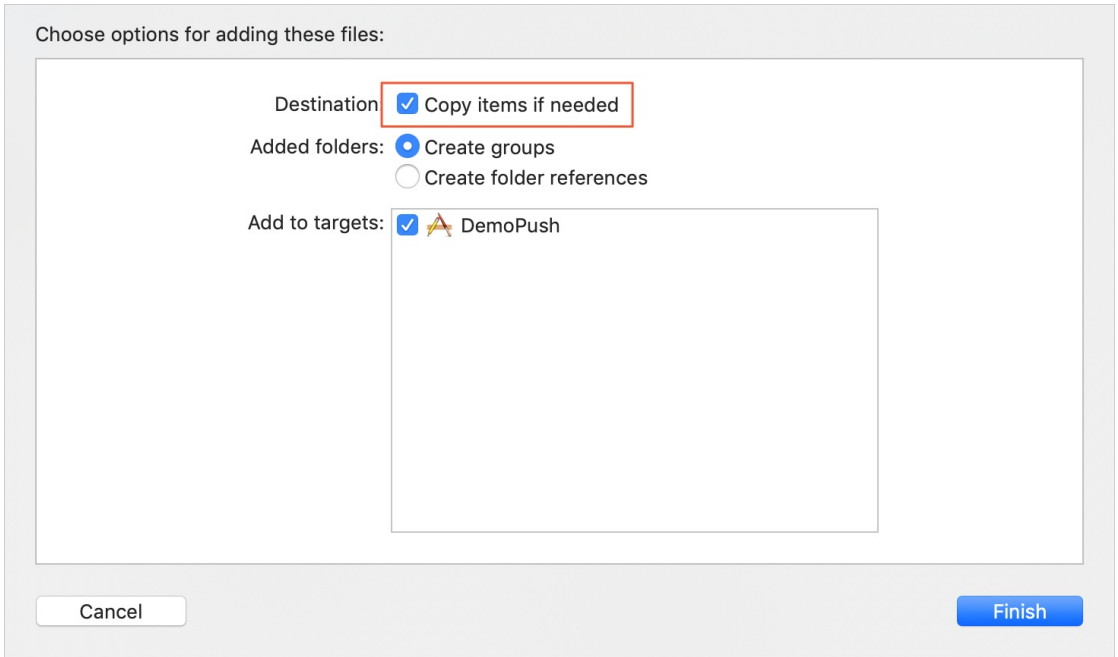
Integrate the SDK

1. Manually import the required files.
   i. Create a demo project named DemoPush in Xcode.
   ii. Add the following files to the demo project in Xcode.
   - AlivcLibRtmp.framework
   - AlivcLivePusher.framework
   - queen.framework
   - AlivcLibBeauty.framework
   - AlivcLibFace.framework
   - AlivcLibFaceResource.bundle
   - Openvc2.framework
   - Face3D.framework
   - FaceDetection.framework
   - MNN.framework
   - pixelai.framework
   - RtsSDK.framework

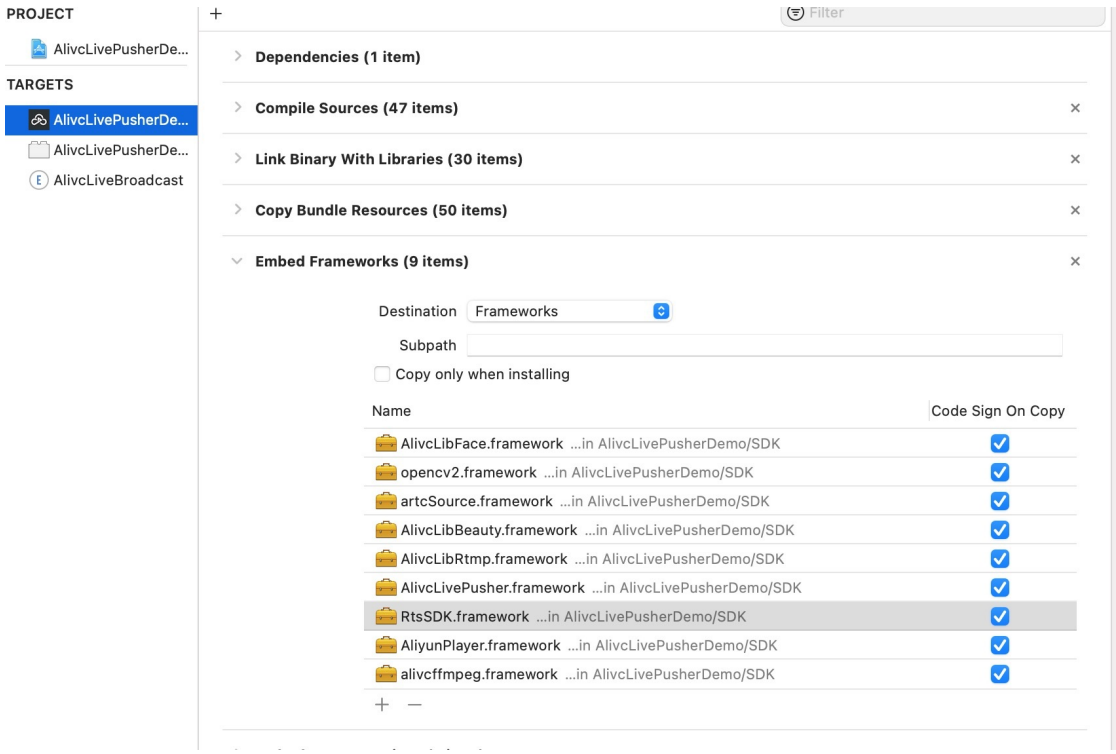
   If you want to use an SDK edition that includes ApsaraVideo Player, you must also add the following files:
   - AliyunPlayer.framework
   - artcSource.framework
   - alivcffmpeg.framework



iii. Select Copy items if needed and click **Finish**.

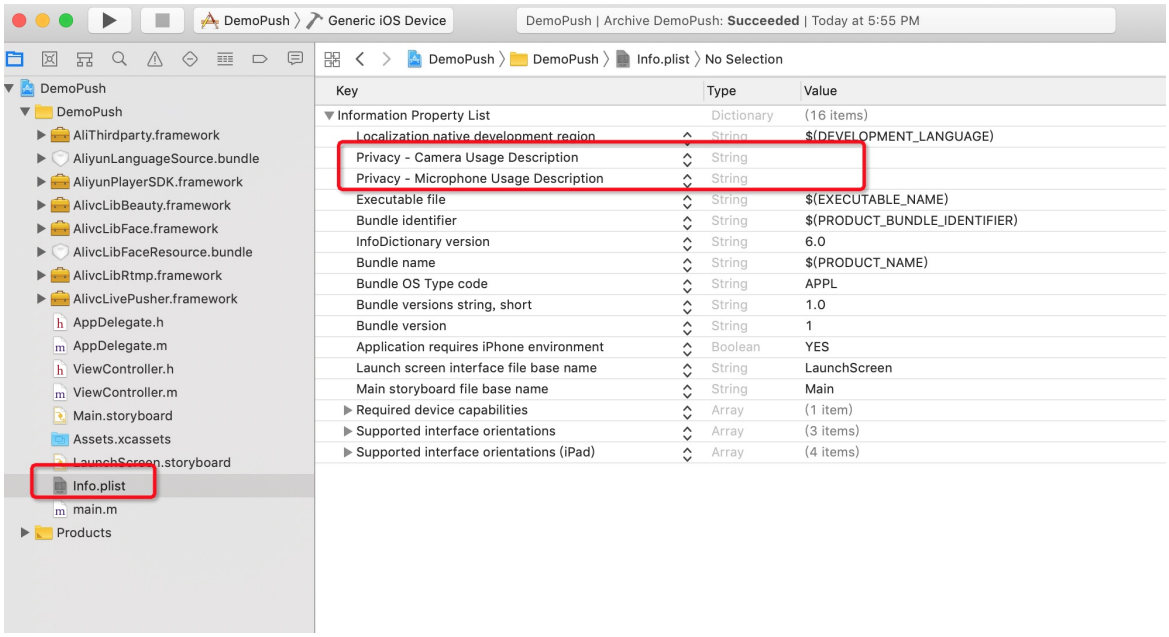


iv. After you import the SDK files, add SDK dependencies in the **Embedded Binaries** section of the **General** tab in **Xcode**.

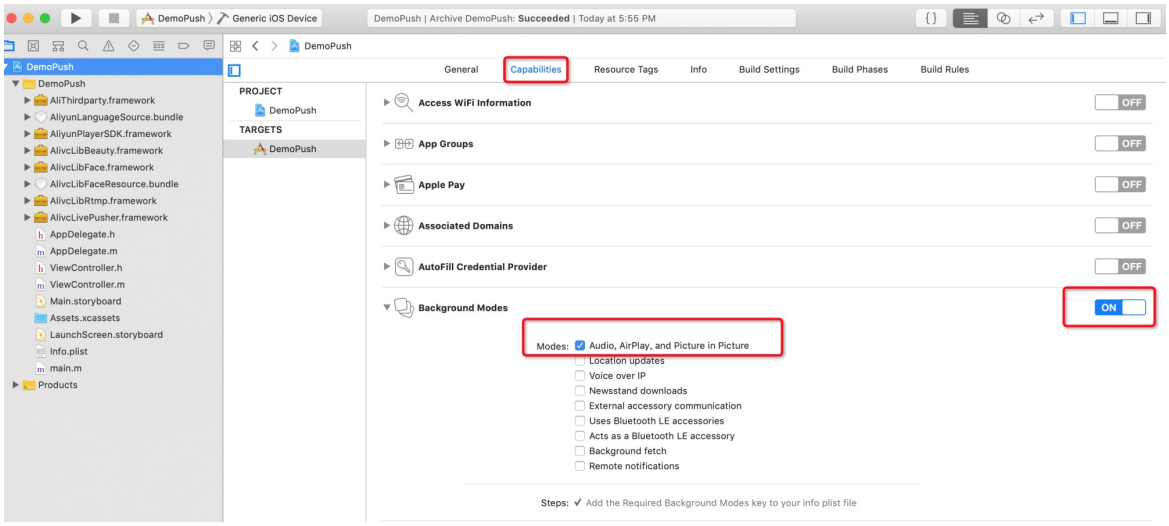


2. Configure the permissions to access features and resources.

   Add the following permissions on the microphone and camera in the Info.plist file: `Privacy - Microphone Usage Description` and `Privacy - Camera Usage Description`.



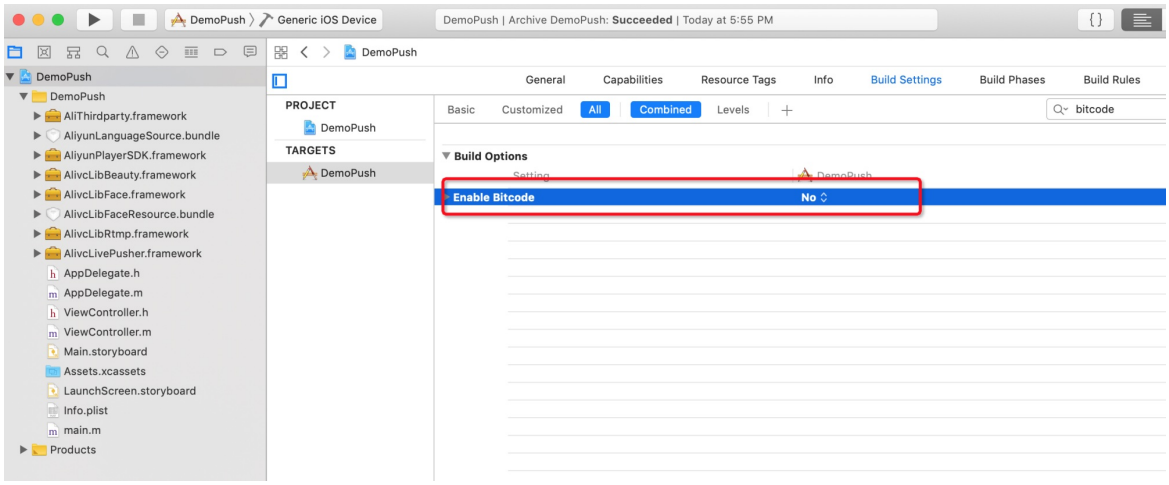
If you want to continue stream ingest when the mobile app is switched to the background, you must enable audio recording in the background.



> **Note** To use the recording feature and the camera, you must add the related permissions to the Info.plist file.

3. Disable the support for Bitcode.

   Push SDK for iOS does not support Bitcode. Therefore, you must set the Enable Bitcode parameter to No for the demo project.



4. Use the SDK by referring to the related topics.
   - For information about the interfaces provided in the SDK, see Interfaces in the SDK for iOS V4.1.0 or the **api** folder in the SDK package.
   - For information about the sample code to use specific interfaces, see SDK usage.

5.4. Use Push SDK for iOS

This topic describes how to use the interfaces in Push SDK for iOS and the procedure for using Push SDK for iOS. This topic also provides examples on how to use Push SDK for iOS to implement specific features.

Features of Push SDK for iOS

- Supports the stream ingest over Real-Time Messaging Protocol (RTMP).

- Adopts H.264 for video coding and Advanced Audio Coding (AAC) for audio coding.
- Supports custom configurations for features such as bitrate control, resolution, and display mode.
- Supports a variety of camera operations.
- Supports real-time retouching and allows you to customize retouching effects.
- Supports using animated stickers as animated watermarks and allows you to add and remove animated watermarks.
- Supports live stream recording.
- Supports external audio and video input in different formats such as YUV and pulse-code modulation (PCM).
- Supports mixing streams.
- Supports the ingest of audio-only and video-only streams and stream ingest in the background.
- Supports background music and allows you to manage background music.
- Supports capturing snapshots from streams.
- Supports automatic reconnection and exception handling.

Procedure

You can use Push SDK for iOS by following these steps:

1. Set the stream ingest parameters.
2. Create a stream ingest preview.
3. Start to ingest streams.
4. Adjust ingest parameters in real time.

Set the stream ingest parameters

You can configure stream ingest parameters in the AlivcLivePushConfig class. Each parameter has a default value. For more information about the default values and valid values, see Interfaces in the SDK for iOS V4.1.0 or comments in the code.

You can modify the values of parameters as needed. To modify these parameters in stream ingest, see the parameters and methods of the AlivcLivePusher class.

This section provides sample code for specific configurations.

- **Basic configurations**

  Run the following command to import the header file in the view controller that requires AlivcLivePusher: `#import <AlivcLivePusher/AlivcLivePusherHeader.h>` . The following code provides an example:

```
AlivcLivePushConfig *config = [[AlivcLivePushConfig alloc] init]; // The configuration class used to initialize the stream ingest configurations. You can also use initWithResolution.
config.resolution = AlivcLivePushResolution540P; // The resolution is set to 540p by default. The maximum resolution is 720p.
config.fps = AlivcLivePushFPS20; // We recommend that you set the frame rate to 20 fps.
config.enableAutoBitrate = true; // Enable adaptive bitrate. The default value is true.
config.videoEncodeGop = AlivcLivePushVideoEncodeGOP_2; // The default value is 2. The longer the interval between key frames, the higher the latency. We recommend that you set this value to a number between 1 and 2.
config.connectRetryInterval = 2000; // Reconnection interval in milliseconds. The default reconnection interval is 2 seconds. The reconnection interval must not be shorter than 1 second. We recommend that you use the default value.
config.previewMirror = false; // The default value is false. We recommend that you use the default value.
config.orientation = AlivcLivePushOrientationPortrait; // The default screen orientation is portrait. You can change the orientation to landscape left or landscape right.
```

Note:

- All these parameters have default values. We recommend that you use the default values.
- Considering the performance of most mobile phones and network bandwidth requirements, we recommend that you set the resolution to 540p. Most mainstream live streaming apps use 540p.
- After adaptive bitrate streaming is disabled, the bitrate is fixed at the initial value and is not automatically adjusted between the specified target bitrate and the minimum bitrate. If the network is unstable, this setting may cause stuttering. Use the adaptive bitrate streaming feature with caution.

- **Bitrate control**

  Push SDK for iOS provides three bitrate control modes. You can specify a mode by setting the qualityMode parameter.

  - **AlivcLivePushQualityModeResolutionFirst**: quality first. Push SDK for iOS configures bitrate parameters to ensure the quality of video streams first.
  - **AlivcLivePushQualityModeResolutionFirst**: smoothness first. Push SDK for iOS configures bitrate parameters to ensure the smoothness of video streams first.
  - **AlivcLivePushQualityModeCustom**: custom mode. Push SDK for iOS configures bitrate parameters based on your custom settings.

  The following code provides an example:

  - The following code provides an example on how to configure quality-first or smoothness-first mode:

    ```
    config.qualityMode = AlivcLivePushQualityModeResolutionFirst; // The default mode is quality-first. You can change this mode to smoothness-first or custom mode.
    ```

    Note:

    If you use the quality-first or smoothness-first mode, you do not need to set the initialVideoBitrate, minVideoBitrate, and targetVideoBitrate parameters. Push SDK for iOS automatically ensures the quality or smoothness of video streams when the network is unstable.

- The following code provides an example on how to configure the custom mode:

```
config.qualityMode = AlivcLivePushQualityModeCustom; // Select the custom mode.
config.targetVideoBitrate = 1200; // The maximum bitrate is 1,200 kbit/s.
config.minVideoBitrate = 400; // The minimum bitrate is 400 kbit/s.
config.initialVideoBitrate = 900; // The initial bitrate is 900 kbit/s.
```

Note:

If you use the custom mode, you must configure the maximum, minimum, and initial bitrates.

- Initial bitrate: The initial bitrate is the bitrate when the live streaming starts.
- Minimum bitrate: When the network bandwidth is low, the bitrate is gradually reduced to the minimum value to avoid stuttering.
- Maximum bitrate: When the network bandwidth is high, the bitrate is gradually increased to the maximum value to improve the quality of the video stream.

We recommend that you use the following parameter values for the custom mode:

Resolution	initialVideoBitrate	minVideoBitrate	targetVideoBitrate
360P	800	200	600
480P	1000	300	700
540P	1200	400	900
720P	1500	600	1200

- **Adaptive resolution streaming**

After adaptive resolution streaming is enabled, the resolution is automatically reduced to ensure the smoothness and quality of video streams in poor network conditions. The adaptive resolution streaming feature is not supported by all players. If you need to use this feature, we recommend that you use ApsaraVideo Player. The following code provides an example:

```
config.enableAutoResolution = YES; // Enable adaptive resolution streaming. The default value is NO.
```

Note:

Adaptive resolution streaming is supported only when the qualityMode parameter is set to the quality-first or smoothness-first mode.

- **Retouching**

Push SDK for iOS provides basic and advanced retouching effects. Basic retouching effects include skin whitening, polishing, and shining. Advanced retouching effects include skin whitening, polishing, and shining, eye widening, and face resizing and slimming. These effects are based on facial recognition. The following code provides an example:

```
# pragma mark - "Retouching effect types and related API interfaces"
/**
 * @brief Specifies whether a beautification type is enabled.
 * @param type The parameter contained in the QueenBeautyType parameter.
 * @param isOpen YES: on. NO: off.
 *
 */
```

```
- (void)setQueenBeautyType:(kQueenBeautyType)type enable:(BOOL)isOpen;
/**
 * @brief Set the retouching parameters.
 * @param param The type of the retouching effect. This parameter is included in the QueenBeautyParam
s.
 * @param value Required. The valid value ranges from 0 to 1. Set the value to 0 if the original value is sma
ller than 0. Set the value to 1 if the original value is greater than 1.
 */
- (void)setQueenBeautyParams:(kQueenBeautyParams)param
value:(float)value;
# pragma mark - "API interfaces for filters"
/**
 * @brief The filter material. Before you specify the filter material, set the kQueenBeautyTypeLUT parame
ter.
 * @param imagePath The path of the filter material.
 */
- (void)setLutImagePath:(NSString *)imagePath;
# pragma mark - "API interfaces for shaping"
/**
 * @brief The shaping type. Before you specify the shaping type, set the kQueenBeautyTypeFaceShape pa
rameter.
 * @param faceShapeType Required. For more information about the configurations, see the QueenBeaut
yFaceShapeType parameter.
 * @param value This parameter is required.
 */
- (void)setFaceShape:(kQueenBeautyFaceShapeType)faceShapeType
value:(float)value;
# pragma mark - "API interfaces for makeup effects"
/**
 * @brief Specify the makeup type and the path of makeup material. Before you specify a makeup effect, s
et the kQueenBeautyTypeMakeup parameter.
 * @param makeupType Specify the makeup type.
 * @param imagePaths Specify the path of the makeup material.
 * @param blend Specify the makeup types to be mixed for retouching.
 */
- (void)setMakeupWithType:(kQueenBeautyMakeupType)makeupType
paths:(NSArray<NSString *> *)imagePaths
blendType:(kQueenBeautyBlend)blend;
/**
 * @brief Specify the makeup types and the path of the makeup materials.
 * @param makeupType Specify the makeup types.
 * @param imagePaths Specify the paths of the makeup materials.
 * @param blend Specify the makeup types to be mixed for retouching.
 * @param fps Specify the frame rate.
 */
- (void)setMakeupWithType:(kQueenBeautyMakeupType)makeupType
paths:(NSArray<NSString *> *)imagePaths
blendType:(kQueenBeautyBlend)blend fps:(int)fps;
/**
 * @brief Set the transparency property for the makeup effect. You can specify the gender for this parame
ter.
 * @param makeupType Specify the makeup type.
 * @param isFeMale Specify whether the gender is female. The value YES indicates female. The value NO in
dicates male.
```

```
 * @param alpha Set the transparency for the makeup effect.
 */
- (void)setMakeupAlphaWithType:(kQueenBeautyMakeupType)makeupType
female:(BOOL)isFeMale alpha:(float)alpha;
/**
 * @brief Set the makeup types to be mixed.
 * @param makeupType Specify the makeup type.
 * @param blend Specify the makeup types to be mixed for retouching.
 */
- (void)setMakeupBlendWithType:(kQueenBeautyMakeupType)makeupType
blendType:(kQueenBeautyBlend)blend;
/**
 * @brief Clear all makeup effects.
 */
- (void)resetAllMakeupType;
```

- **Ingest of images**

  Push SDK for iOS supports the ingest of an image when your app is switched to the background or the bitrate is low. This enhances the user experience. When your app is switched to the background, stream ingest is paused by default. In this case, only an image and audio streams can be ingested. For example, you can ingest an image that displays a message reminding the leave of the streamer. The following code provides an example:

  ```
  config.pauseImg = [UIImage imageNamed:@"image.png"]; // Specify the image for background stream ing
  est.
  ```

  In addition, you can specify a static image for stream ingest in poor network conditions. After that, the specified image is ingested when the bitrate is low. This avoids stuttering. The following code provides an example:

  ```
  config.networkPoorImg = [UIImage imageNamed:@"picture.png"]; // Specify the image for stream ingest i
  n poor network conditions.
  ```

- **Watermarks**

  Push SDK for iOS allows you to add one or more watermarks in the PNG format. The following code provides an example:

  ```
  NSString *watermarkBundlePath = [[NSBundle mainBundle] pathForResource:
  [NSString stringWithFormat:@"watermark"] ofType:@"png"]; // Specify the path of the watermark.
  [config addWatermarkWithPath: watermarkBundlePath
      watermarkCoordX:0.1
      watermarkCoordY:0.1
      watermarkWidth:0.3]; // Add a watermark.
  ```

  Note:

  - The values of the coordX, coordY, and width parameters are relative. For example, a value of 0.1 for the CoordX parameter indicates that the left edge of the watermark is displayed at the 10% position on the x-axis of the streaming image. Therefore, if the streaming resolution is 540 × 960, the value for the watermarkCoordX parameter is 54.
  - The height of the watermark is calculated based on the input width in a proportional aspect ratio.
  - If you want to add a text watermark, you can transform the text into an image and then call the addWaterMark interface to add the image as a watermark.

- To ensure the clarity and smooth edges of the watermark image, we recommend that you use a source image in the watermark output size. If the resolution of the output video is 544 × 940 and the watermark width is 0.1f, we recommend that you use the following width for the source image: 544 × 0.1f = 54.4.

- **Preview mode**

```
mAlivcLivePushConfig.setPreviewDisplayMode(AlivcPreviewDisplayMode.ALIVC_LIVE_PUSHER_PREVIEW_ASPECT_FIT);
```

Note:

- ALIVC_LIVE_PUSHER_PREVIEW_SCALE_FILL: Specify the video to fill the entire preview view. If the aspect ratio of the video does not match the view, the preview image is deformed.
- ALIVC_LIVE_PUSHER_PREVIEW_ASPECT_FIT: Specify the initial aspect ratio of the video when the preview is displayed. If the aspect ratio of the video does not match the view, black edges appear on the preview view, which is set by default.
- ALIVC_LIVE_PUSHER_PREVIEW_ASPECT_FILL: Change the aspect ratio of the video to fit for the preview view. If the aspect ratio of the video does not match the view, the video is cropped to fit the preview view.

You can specify these modes in AlivcLivePushConfig. You can also set the setpreviewDisplayMode parameter during preview and stream ingest.

This configuration takes effect only for the preview mode. The actual resolution of the output video follows the default configurations in AlivcLivePushConfig. The preview mode does not affect the actual resolution. You can select a preview mode to adapt to different screen sizes of mobile phones.

Use AlivcLivePusher

AlivcLivePusher is the core class of Push SDK for iOS. This class provides parameters for video preview, stream ingest callback, and stream ingest control. You can also use this class to modify parameters during stream ingest.

This section provides sample code for specific configurations.

- **Initialization**

Use the initWithConfig method to initialize the configured stream ingest parameters. The following code provides an example:

```
self.livePusher = [[AlivcLivePusher alloc] initWithConfig:config];
```

Note:

AlivcLivePusher does not support multiple instances. Therefore, Push SDK for iOS provides a destroy method for each init method.

- **Register stream ingest callbacks**

Stream ingest callbacks are grouped into Info, Error, and Network callbacks.

- Info: the callbacks that are used for notifications and status detection.
- Error: the callbacks that are used when errors occur.
- Network: the callbacks that are used to manage network services.

You can use the delegate method to receive the specified callbacks.

```
[self.livePusher setInfoDelegate:self];
[self.livePusher setErrorDelegate:self];
[self.livePusher setNetworkDelegate:self];
```

- **Start preview**

  You can start preview after you initialize the livePusher object. Create a view instance in the UIView class or a class that inherits from UIView to start preview. The following code provides an example:

  ```
  [self.livePusher startPreview:self.view];
  ```

- **Start stream ingest**

  You can start stream ingest only after preview succeeds. Therefore, you must set the onPreviewStarted callback for AlivcLivePusherInfoDelegate by adding the following code to the callback:

  ```
  [self.livePusher startPushWithURL:@"Ingest URL for test (rtmp://.....)"];;
  ```

  Note:

  - Push SDK for iOS provides the asynchronous interface startPushWithURLAsync to start stream ingest.
  - Push SDK for iOS supports the URLs of the streams that are ingested over RTMP. For more information about how to obtain ingest URLs, see Ingest and streaming URLs.
  - Start stream ingest with a valid URL. Then, use a player, such as ApsaraVideo Player, FFplay, or VLC, to test stream pulling. For more information about how to obtain source URLs, see Ingest and streaming URLs.

- **Other stream ingest configurations**

  Push SDK for iOS allows you to control stream ingest. For example, you can start, stop, restart, pause, and resume stream ingest, stop preview, and destroy stream ingest objects. You can add buttons as needed to perform these operations. The following code provides an example:

```
/*You can pause the ongoing stream ingest. If you pause the ongoing stream ingest, the system pauses th
e video preview and the video stream ingest at the last frame, and continues the ingest of audio-only stre
ams. */
[self.livePusher pause];
/*You can resume the paused stream ingest. Then, the system resumes the audio and video preview and t
he stream ingest. */
[self.livePusher resume];
/*You can stop the ongoing stream ingest. */
[self.livePusher stopPush];
/*You can stop the ongoing preview. However, this operation does not take effect for the ongoing stream
ingest. When the preview is stopped, the preview view is frozen at the last frame. */
[self.livePusher stopPreview];
/*You can restart stream ingest that is in the streaming status or when the method receives an error callb
ack. When an error occurs, you can only use this method or reconnectPushAsync to restart stream ingest.
You can also use the destroy method to destroy the stream ingest object. After you complete the precedi
ng operation, restart all ALivcLivePusher resources that are required for operations such as the preview a
nd stream ingest. */
[self.livePusher restartPush];
/*You can use this method in the streaming status or when the method receives callbacks caused by error
s in AlivcLivePusherNetworkDelegate. When an error occurs, you can only use this method or restartPush
to restart stream ingest. You can also use the destroy method to destroy the stream ingest object. After y
ou complete the preceding operation, you can restart the stream ingest over RTMP. */
[self.livePusher reconnectPushAsync];
/*After the ingest stream object is destroyed, the stream ingest and the preview are stopped, and preview
views are deleted. All resources related to AlivcLivePusher are destroyed. */
[self.livePusher destory];
self.livePusher = nil;
/*Query the status of stream ingest. */
AlivcLivePushStatus status = [self.livePusher getLiveStatus];
```

- **Real-time adjustment of retouching effects**

  Push SDK for iOS allows you to adjust retouching effects in real time during stream ingest. You can enable the retouching feature and set retouching parameters as needed. The following code provides an example:

```
[_queenEngine setQueenBeautyType:kQueenBeautyTypeSkinBuffing enable:YES];
[_queenEngine setQueenBeautyType:kQueenBeautyTypeSkinWhiting enable:YES];
[_queenEngine setQueenBeautyParams:kQueenBeautyParamsWhitening value:0.8f];
[_queenEngine setQueenBeautyParams:kQueenBeautyParamsSharpen value:0.6f];
[_queenEngine setQueenBeautyParams:kQueenBeautyParamsSkinBuffing value:0.6];
```

- **Background music**

  Push SDK for iOS allows you to manage background music. For example, you can configure the playback of background music, and the audio mixing, noise reduction, in-ear monitoring, and muting features. You can call background music interfaces only after preview starts. The following code provides an example:

```
/*Start the playback of background music. */
[self.livePusher startBGMWithMusicPathAsync:musicPath];
/*Stop the playback of background music. To change the background music, call the interface that is used to start the playback of background music. You do not need to stop the playback of the current background music. */
[self.livePusher stopBGMAsync];
/*Pause the playback of the background music. You can call this interface only after the background music starts. */
[self.livePusher pauseBGM];
/*Resume the playback of background music. You can call this interface only after the background music is paused. */
[self.livePusher resumeBGM];
/*Enable looping.*/
[self.livePusher setBGMLoop:true];
/*Configure noise reduction. When noise reduction is enabled, the system filters out non-vocal parts from collected audio. This feature may slightly reduce the volume of the human voice. Therefore, we recommend that you allow your users to determine whether to enable this feature. This feature is disabled by default.*/
[self.livePusher setAudioDenoise:true];
/*Configure in-ear monitoring. In-ear monitoring applies to the KTV scenario. When in-ear monitoring is enabled, you can hear your voice on your earphones during streaming. When in-ears monitoring is disabled, you cannot hear your voice on your earphones during streaming. This feature does not take effect if no earphones are detected. */
[self.livePusher setBGMEarsBack:true];
/*Configure audio mixing by adjusting the volumes of the background music and the human voice. */
[self.livePusher setBGMVolume:50]; // Set the volume of the background music.
[self.livePusher setCaptureVolume:50]; // Set the volume of the human voice.
/*Set muting. If you enable this feature, the background music and the human voice are muted. To separately mute background music or the human voice, call the interface that is used to configure audio mixing. */
[self.livePusher setMute:isMute?true:false];
```

- Camera-related operations

  You can perform camera-related operations only after you start preview in streaming status, paused status, or reconnecting status. For example, you can switch the camera, and configure the flash, the focal length, zooming, and the mirroring mode. If you do not start preview, the following interfaces are invalid. The following code provides an example:

```
/*Switch between the front and the rear cameras.*/
[self.livePusher switchCamera];
/*Enable or disable the flash. You cannot enable the flash for the front camera.*/
[self.livePusher setFlash:false];
/*Set the focal length to zoom in and out of images. If you set the parameter to a positive number, the sys
tem increases the focal length. If you set the value to a negative number, the system decreases the focal le
ngth. */
CGFloat max = [_livePusher getMaxZoom];
[self.livePusher setZoom:MIN(1.0, max)];
/*Configure manual focus. To configure manual focus, you must set the following two parameters: point,
which specifies the coordinates of the focus point, and autoFocus, which is used to enable or disable auto
focus. The autoFocus parameter is valid only when you call this interface. Other auto focus settings follo
w the same settings. */
[self.livePusher focusCameraAtAdjustedPoint:CGPointMake(50, 50) autoFocus:true];
/*Configure auto focus.*/
[self.livePusher setAutoFocus:false];
/*Configure the mirroring mode. Mirroring-related interfaces are PushMirror and PreviewMirror. The Push
Mirror interface is used to configure the mirroring mode for stream ingest. The PreviewMirror interface is
used to configure the mirroring mode for preview. PushMirror is valid only for playback images. PreviewM
irror is valid only for preview views. */
[self.livePusher setPushMirror:false];
[self.livePusher setPreviewMirror:false];
```

- **Live Q&A**

  To implement the live question and answer (Q&A) feature, you must insert supplemental enhancement information (SEI) into live streams and parse SEI by using the player. Push SDK for iOS provides an interface to insert SEI. This interface can be called only during stream ingest. The following code provides an example:

```
/*
msg: The SEI messages to be inserted into the live stream. The SEI messages are in the JSON format. The A
psaraVideo Player SDK can receive and parse SEI messages.
repeatCount: the number of frames to which the SEI messages are inserted. To ensure that no SEI messag
es are dropped for a frame, you must set the number of repetitions. A value of 100 indicates that SEI mess
ages are inserted into the subsequent 100 frames. ApsaraVideo Player deduplicates the same SEI message
s.
delayTime: the time period before the frames are sent. Unit: milliseconds.
KeyFrameOnly: specifies whether to send only keyframes.
*/
[self.livePusher sendMessage:@"Information about questions" repeatCount:100 delayTime:0 KeyFrameO
nly:false];
```

- **External audio and video input**

  Push SDK for iOS allows you to import external audio and video sources for stream ingest. For example, you can ingest an audio or video file.

- Configure the input of external audio and video sources in stream ingest settings. The following code provides an example:

```
config.externMainStream = true; // Enable the input of external streams.
config.externVideoFormat = AlivcLivePushVideoFormatYUVNV21; // Specify the color format for video d
ata. In this example, the color format is YUVNV21. You can also use other formats as needed.
config.externMainStream = AlivcLivePushAudioFormatS16; // Specify the bit depth format for audio dat
a. In this example, the bit depth format is S16. You can also use other formats as needed.
```

- The following code provides an example on how to import external video data:

```
/*The sendVideoData interface supports only native YUV and RGB buffer data. You can use the sendVide
oData interface to send the buffer, length, width, height, timestamp, and rotation angle of video data.*
/
[self.livePusher sendVideoData:yuvData width:720 height:1280 size:dataSize pts:nowTime rotation:0];
/*For CMSampleBufferRef video data, you can call the sendVideoSampleBuffer interface.*/
[self.livePusher sendVideoSampleBuffer:sampleBuffer]
/*You can also transform CMSampleBufferRef video data to continuous buffer data before you call the s
endVideoData interface. The following code provides an example.*/
// Query the length of samplebuffer.
- (int) getVideoSampleBufferSize:(CMSampleBufferRef)sampleBuffer {
if(!sampleBuffer) {
   return 0;
}
int size = 0;
CVPixelBufferRef pixelBuffer = CMSampleBufferGetImageBuffer(sampleBuffer);
CVPixelBufferLockBaseAddress(pixelBuffer, 0);
if(CVPixelBufferIsPlanar(pixelBuffer)) {
  int count = (int)CVPixelBufferGetPlaneCount(pixelBuffer);
  for(int i=0; i<count; i++) {
     int height = (int)CVPixelBufferGetHeightOfPlane(pixelBuffer,i);
     int stride = (int)CVPixelBufferGetBytesPerRowOfPlane(pixelBuffer,i);
     size += stride*height;
  }
}else {
  int height = (int)CVPixelBufferGetHeight(pixelBuffer);
  int stride = (int)CVPixelBufferGetBytesPerRow(pixelBuffer);
  size += stride*height;
}
CVPixelBufferUnlockBaseAddress(pixelBuffer, 0);
return size;
}
// Transform video sample buffer to native buffer.
- (int) convertVideoSampleBuffer:(CMSampleBufferRef)sampleBuffer toNativeBuffer:(void*)nativeBuffe
r
{
if(!sampleBuffer || !nativeBuffer) {
  return -1;
}
CVPixelBufferRef pixelBuffer = CMSampleBufferGetImageBuffer(sampleBuffer);
CVPixelBufferLockBaseAddress(pixelBuffer, 0);
int size = 0;
if(CVPixelBufferIsPlanar(pixelBuffer)) {
  int count = (int)CVPixelBufferGetPlaneCount(pixelBuffer);
  for(int i=0; i<count; i++) {
```

```
    int height = (int)CVPixelBufferGetHeightOfPlane(pixelBuffer,i);
    int stride = (int)CVPixelBufferGetBytesPerRowOfPlane(pixelBuffer,i);
    void *buffer = CVPixelBufferGetBaseAddressOfPlane(pixelBuffer, i);
    int8_t *dstPos = (int8_t*)nativeBuffer + size;
    memcpy(dstPos, buffer, stride*height);
    size += stride*height;
  }
}else {
  int height = (int)CVPixelBufferGetHeight(pixelBuffer);
  int stride = (int)CVPixelBufferGetBytesPerRow(pixelBuffer);
  void *buffer = CVPixelBufferGetBaseAddress(pixelBuffer);
  size += stride*height;
  memcpy(nativeBuffer, buffer, size);
}
CVPixelBufferUnlockBaseAddress(pixelBuffer, 0);
return 0;
}
```

- The following code provides an example on how to import external audio and video data:

```
/*The sendPCMData interface supports only native pulse-code modulation (PCM) buffer data. You can u
se sendPCMData to transmit the buffer, length, and timestamp of audio data.*/
[self.livePusher sendPCMData:pcmData size:size pts:nowTime];
```

- **Animated stickers**

  Push SDK for iOS allows you to add animated stickers as watermarks to live streams.

```
# pragma mark - "API interfaces related to animated stickers"
/**
 * @brief Add animated stickers. Supported formats include glTF, TAOPAI, and mediaAI.
 * @param materialPath The path of the folder that contains the animated stickers to be added.
 */
- (void)addMaterialWithPath:(NSString *)materialPath;
/**
 * @brief Remove animated stickers.
 * @param materialPath Specify the path of the folder that contains the animated stickers to be removed.
 */
- (void)removeMaterialWithPath:(NSString *)materialPath;
```

- **Debugging tools**

  DebugView is a UI debugging tool that allows you to diagnose issues. DebugView provides a draggable and always-on-top window for debugging. For example, you can query the logs of stream ingest, monitor the metrics for stream ingest performance in real time, and generate line charts of main performance metrics. The following code provides an example:

```
[AlivcLivePusher showDebugView]; // Show DebugView.
```

  > **Note** Note: In the current release, do not call an interface that invokes DebugView.

- **Other interfaces**

```
/*In custom mode, you can change the minimum bitrate and the maximum bitrate in real time. */
[self.livePusher setTargetVideoBitrate:800];
[self.livePusher setMinVideoBitrate:200]
/*Obtain the status of stream ingest.*/
BOOL isPushing = [self.livePusher isPushing];
/*Obtain the ingest URL.*/
NSString *pushURLString = [self.livePusher getPushURL];
/*Obtain the debugging information about stream ingest performance. For more information about the p
arameters of stream ingest performance, see API references or comments in the code.*/ */
AlivcLivePushStatsInfo *info = [self.livePusher getLivePushStatusInfo];
/*Obtain the version number of Push SDK for iOS. */
NSString *sdkVersion = [self.livePusher getSDKVersion];
/*Set a log level to filter debugging information.*/
[self.livePusher setLogLevel:(AlivcLivePushLogLevelDebug)];
```

Use ReplayKit for live stream recording

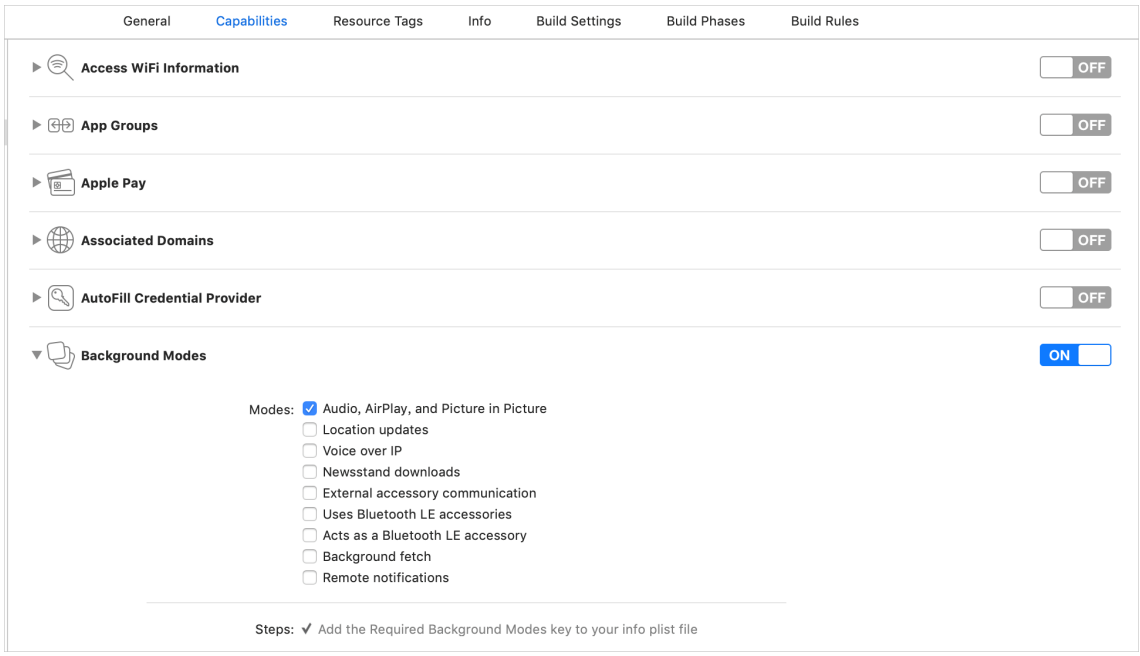
ReplayKit is an iOS 9 framework that allows you to record your screen content. In iOS 10, you can use third-party extensions to broadcast the screen. Push SDK for iOS can work with app extensions to support live stream recording in iOS 10 and later versions.

- **Create a live streaming extension**

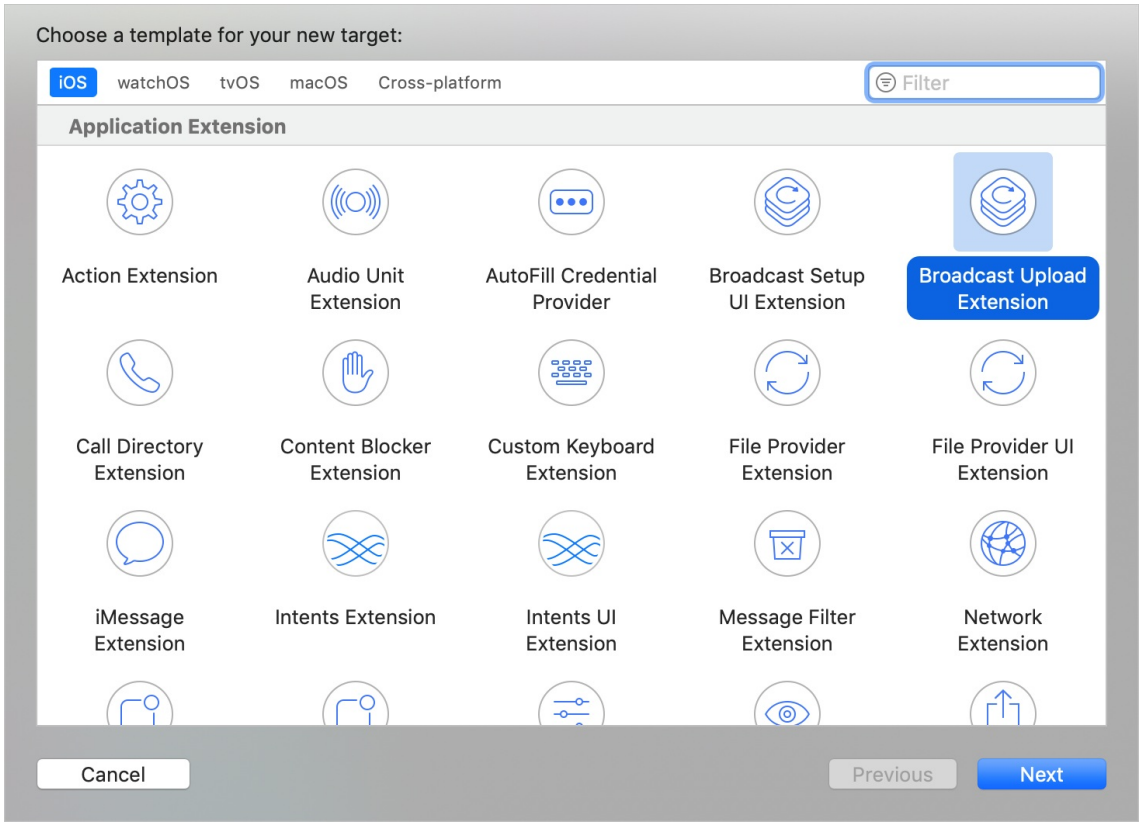
  You can use ReplayKit to create screen recorder extensions. An app extension cannot be used as an app. You cannot submit an app extension unless it is inside a containing app. However, an app extension directly communicates with a host app by using requests and responses with no support from the containing app. During the communication, the host app sends a request to launch the app extension.

  The Alibaba Cloud Live Streaming demo has provided the AlivcLiveBroadcast and AlivcLiveBroadcastSetupUI app extensions to record and broadcast screen content. To create a live streaming extension in an app, perform the following steps:

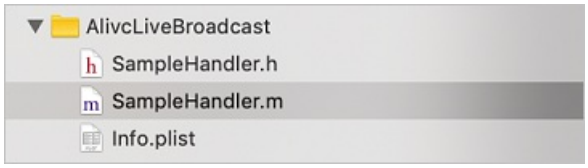
  i. Choose **New > Target** in the current project. Then, click Broadcast Upload Extension. The following page appears.



ii. Modify the **Product Name** parameter, select Include UI Extension, and then click **Finish**. After that, a live streaming extension and UI are created, as shown in the following figure.



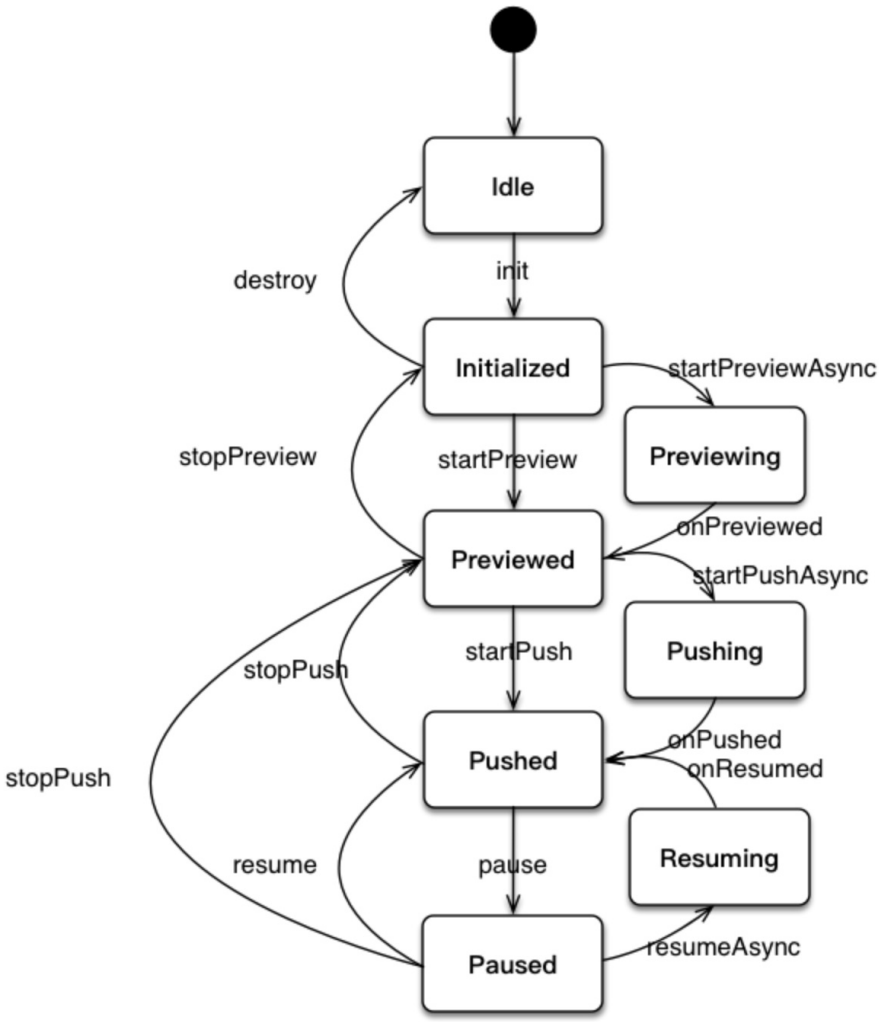
iii. Configure the Info.plist extension, as shown in the following figure.



iv. Launch the app to automatically install both the live streaming extension and the app to your mobile phone. After the installation, launch an app that supports ReplayKit, such as TowerDash. Then, click the live streaming button. You can see the icon of the extension in the drop-down menus.

- **Integrate Push SDK for iOS**
  - Add the AlivcLivePusher.framework and AlivcLibRtmp.framework dependencies to the stream ingest extension.

- Configure the UI extension parameters, including the live stream URL, resolution, and screen rotation, as shown in the following figure.



- Use AlivcLivePusher to configure live streaming settings. The SampleHandle object provides all the interfaces for live stream recording with ReplayKit. You can call these interfaces for AlivcLivePusher in Push SDK for iOS to use the relevant features.
  - **Start stream ingest**

    You can call the broadcastStartedWithSetupInfo interface to set the stream ingest parameters and start to ingest streams. The following code provides an example:

```
- (void)broadcastStartedWithSetupInfo:(NSDictionary<NSString *,NSObject *> *)setupInfo {
self.pushConfig = [[AlivcLivePushConfig alloc] init];
self.livePusher = [[AlivcLivePusher alloc];
self.pushConfig.externMainStream = true;
[self.livePusher initWithConfig:self.pushConfig];
[self.livePusher startPushWithURL:pushUrl];
}
```

- Pause stream ingest

  You can call the broadcastPaused interface to pause stream ingest. The following code provides an example:

```
- (void)broadcastPaused {
[self.livePusher pause];
}
```

- Resume stream ingest

  You can call the broadcastResumed interface to resume stream ingest. The following code provides an example:

```
- (void)broadcastResumed {
[self.livePusher resume];
}
```

- Stop stream ingest

  You can call the broadcastFinished interface to stop stream ingest. The following code provides an example:

```
- (void)broadcastFinished {
[self.livePusher stopPush];
[self.livePusher destory];
self.livePusher = nil;
}
```

- Send audio and video data

  You can call the processSampleBuffer interface to send audio streams. The following code provides an example:

```
- (void)processSampleBuffer:(CMSampleBufferRef)sampleBuffer withType:(RPSampleBufferType)sa
mpleBufferType {
switch (sampleBufferType) {
 case RPSampleBufferTypeVideo:
    // Handle video sample buffer
    [self.livePusher processVideoSampleBuffer:sampleBuffer];
    break;
 case RPSampleBufferTypeAudioApp:
    // Handle audio sample buffer for app audio
     [self.livePusher processAudioSampleBuffer:sampleBuffer withType:sampleBufferType];
    break;
 case RPSampleBufferTypeAudioMic:
    // Handle audio sample buffer for mic audio
    [self.livePusher processAudioSampleBuffer:sampleBuffer withType:sampleBufferType];
    break;
 default:
    break;
 }
}
```

Note

- Package size
  - The size of Push SDK for iOS is 20.7 MB.
  - After you integrate Push SDK for iOS, the size of the IPA package increases by 7.8 MB.
- Compatible mobile phones

  iPhone 5s and later, with iOS 8.0 or later versions.
- Limits
  - You must configure the screen rotation before stream ingest. You cannot rotate the screen during stream ingest.
  - In hardware encoding mode, the output resolution must be multiples of 16 due to the compatibility of the encoder. For example, if you set the resolution to 540p, the output resolution is 544 x 960. You must scale the screen size of the player based on the output resolution to prevent black edges.
- Version upgrade instruction

  To upgrade Push SDK for iOS from V4.0.2 to V4.1.0, you must delete the original SDK and download the latest SDK. For more information, see Update Push SDK from V4.0.2 to V4.1.0.

5.5. Handling of exceptions and special scenarios

This topic describes the exceptions and special scenarios that you may encounter when you use Push SDK for iOS. This topic also describes how to handle such exceptions and special scenarios.

Handle AlivcLivePusherErrorDelegate callbacks

- When you receive an onSystemError callback message, which indicates a system error, stop live streaming.
- When you receive an onSDKError callback message, which indicates an SDK error, destroy the current AlivcLivePusher object and then create another one. Alternatively, you can call the restartPush or restartPushAsync method to restart the current AlivcLivePusher object.
- You must pay special attention to the fired callbacks that are related to microphone and camera permission issues. The 268455940 error code indicates that the mobile app requires permissions to use the microphone. The 268455939 error code indicates that the mobile app requires permissions to use the camera.

Handle AlivcLivePusherNetworkDelegate callbacks

- An onNetworkPoor callback message is sent to you when the network speed is slow. The message indicates that the network cannot support stream ingest, but the stream is still being ingested and not interrupted. When the network speed resumes to the normal level, an onNetworkRecovery callback message is sent to you. At this point, you can configure custom business logic. For example, you can notify your users of network recovery in a visualized manner on the mobile app UI.
- The onConnectFail, onReconnectError, or onSendDataTimeout callback is fired when a specific network error occurs. In this case, destroy the current AlivcLivePusher object and then create another one. Alternatively, you can call the reconnectAsync method to reconnect to the network. We recommend that you test the network connectivity and the ingest URL before you perform the reconnection operation.

- The SDK fires the onReconnectStart callback to reconnect to the network when the SDK receives a reconnection request. The reconnection request may be issued by the SDK or the developer that calls the reconnectAsync method. To reconnect to the network, the mobile app reinitiates a connection to the Real-Time Messaging Protocol (RTMP) server.

  After the RTMP connection is established, an onReconnectSuccess callback message is sent to you. However, the message does not mean that the mobile app resumes stream ingest. Stream ingest may fail to be resumed if a network issue persists. In this case, the SDK keeps reconnecting to the RTMP server.

- An onPushURLAuthenticationOverdue callback message indicates the expiration of the ingest URL authentication. The SDK sends such a message to you only after you enable ingest URL authentication. An ingest URL contains the auth_key field. This callback is fired 1 minute before the ingest URL expires. After you receive such a message, you must specify a new ingest URL to ensure that the ingest process is not interrupted due to the expiration of the original ingest URL. Sample code:

```
- (NSString *)onPushURLAuthenticationOverdue:(AlivcLivePusher *)pusher {
return @"New ingest URL rtmp://";
}
```

Handle AlivcLivePusherBGMDelegate callbacks

- The onOpenFailed callback is fired when the background music (BGM) fails to be played. In this case, check whether the audio file and the file path specified for the startBGMAsync method are valid. Then, call the startBGMAsync method to replay the BGM.
- The onDownloadTimeout callback is fired when the BGM playback times out. This issue generally occurs when the BGM is specified by using an online URL. This callback instructs the streamer to check the status of the network. You can call the startBGMAsync method to replay the BGM.

Handle network disconnection

- Short-period network disconnection and network switchover: The SDK attempts to reconnect to the network when a network fluctuation or a network switchover occurs. You can use the AlivcLivePushConfig class to specify the reconnection timeout period and the maximum number of reconnection attempts allowed. After the SDK reconnects to the network, stream ingest is resumed. If you use ApsaraVideo Player, we recommend that you perform a reconnection operation 5 seconds after you receive an AliVcMediaPlayerPlaybackDidFinishNotification timeout notification.
- Long-period network disconnection: The SDK fails to reconnect to the network if a reconnection request times out or the number of reconnection attempts exceeds the upper limit. In this case, the `onReconnectError:error:` callback is fired. After the network is recovered, call the reconnectAsync method to reconnect the SDK to the network. You must also reconnect the SDK to ApsaraVideo Player.
  - We recommend that you externally monitor the network.
  - Use the server to handle communication failures between the streamer and player. For example, when network disconnection occurs on the streamer, the server receives a callback message of stream ingest interruption from Alibaba Cloud CDN. The server pushes the callback message to the player. Then, the player takes relevant measures to handle the stream ingest interruption. The server uses the same procedure to resume stream ingest.

- Stop and then restart ApsaraVideo Player to reconnect the player to the ingest URL. To achieve this objective, call the stop, prepare, and play methods in sequence.

```
[self.mediaPlayer stop];
AliVcMovieErrorCode err = [self.mediaPlayer prepareToPlay:[NSURL URLWithString:@"ingest URL"]];
if(err != ALIVC_SUCCESS) {
 NSLog(@"play failed,error code is %d",(int)err);
 return;
}
[self.mediaPlayer play];
```

> **Note** For more information about ApsaraVideo Player, see Implementation.

Switch the mobile app to the background or answer a phone call

The SDK provides built-in configurations for the background mode. When you switch the mobile app to the background, the video is paused at the last frame. The mobile app continues playing the audio in the background. Open your project in Xcode. On the Signing & Capabilities tab, select **Audio, AirPlay, and Picture in Picture** in the Background Modes section. This ensures that audio can be recorded when the mobile app is switched to the background. The following figure shows the configuration.

You can disable audio recording in the background to completely pause stream ingest when the mobile app is switched to the background. Stream ingest is resumed when the mobile app is switched back to the foreground. You can disable audio recording in the background by using one of the following approaches:

- Call the setMute method to enable the mute mode when the mobile app is switched to the background. This is the recommended approach.
- Call the stopPush method to pause stream ingest when the mobile app is switched to the background and call the startPushWithURL or startPushWithURLAsync method to resume stream ingest when the mobile app is switched back to the foreground.

> **Note** If you use this approach, you must configure the system to monitor changes of the UIApplicationWillResignActiveNotification and UIApplicationDidBecomeActiveNotification properties. Otherwise, an error may occur.

```
- (void)addNotifications {
[[NSNotificationCenter defaultCenter] addObserver:self
  selector:@selector(applicationWillResignActive:)
  name:UIApplicationWillResignActiveNotification
  object:nil];
[[NSNotificationCenter defaultCenter] addObserver:self
  selector:@selector(applicationDidBecomeActive:)
  name:UIApplicationDidBecomeActiveNotification
  object:nil];
}
 - (void)applicationWillResignActive:(NSNotification *)notification {
[self.livePusher stopPush];
}
 - (void)applicationDidBecomeActive:(NSNotification *)notification {
[self.livePusher startPushWithURLAsync:pushURL];
}
```

Play external audio

To play external audio on the page for stream ingest, we recommend that you use the AVAudioPlayer class because the SDK is incompatible with the AudioServicesPlaySystemSound class. After you configure external audio playback, you need to update the AVAudioSession settings. Sample code:

```
- (void)setupAudioPlayer {
  NSString *filePath = [[NSBundle
mainBundle] pathForResource:@"sound" ofType:@"wav"];
  NSURL *fileUrl = [NSURL URLWithString:filePath];
 self.player = [[AVAudioPlayer alloc] initWithContentsOfURL:fileUrl error:nil];
 self.player.volume = 1.0;
 [self.player prepareToPlay];
}
 - (void)playAudio {
  self.player.volume = 1.0;
  [self.player play];
  // Configure AVAudioSession settings.
  AVAudioSession *session = [AVAudioSession sharedInstance];
  [session setMode:AVAudioSessionModeVideoChat error:nil];
  [session overrideOutputAudioPort:AVAudioSessionPortOverrideSpeaker error:nil];
  [session setCategory:AVAudioSessionCategoryPlayAndRecord withOptions:AVAudioSessionCategoryOpti
onDefaultToSpeaker|AVAudioSessionCategoryOptionAllowBluetooth
| AVAudioSessionCategoryOptionMixWithOthers error:nil];
  [session setActive:YES error:nil];
}
```

Change the size of the view during stream ingest

Check the values of the UIView properties when you call the startPreview or startPreviewAsync method. Change the value of the frame property for all subviews in the preview. Sample code:

```
[self.livePusher startPreviewAsync:self.previewView];
for (UIView *subView in [self.previewView subviews]) {
 // ...
}
```

Adapt previews to an iPhone X

Generally, all previews can be properly displayed in full screen mode on mobile phones. However, the screen of an iPhone X has a special aspect ratio. Therefore, previews are distorted when they are displayed in full screen mode on an iPhone X. We recommend that you do not display previews in full screen mode on an iPhone X.

Set bitrates for ingested streams

The SDK supports dynamic bitrate conversion. You can use the AlivcLivePushConfig class to change the default bitrate. Different services require different video quality. The resolution, smoothness, and definition of an output video vary based on the bitrate of the ingested stream.

- Video definition: The larger the bitrate of the ingested stream is, the higher the video definition is. We recommend that you specify a large bitrate to ensure the playback of high-definition videos.
- Video smoothness: The larger the bitrate of the ingested stream is, the higher network bandwidth is required. A large bitrate with low network bandwidth may affect the smoothness of the videos.

6.Stream ingest over RTS or WebRTC

This topic describes how to ingest streams over Real-Time Streaming (RTS). This topic also provides usage notes of stream ingest over RTS.

Procedure

1. Add domain names, resolve canonical domain name (CNAME) records, and associate domain names by using the ApsaraVideo Live console or API. For more information, see Quick start.
2. Enable low-latency stream ingest in the console.

   On the Domains page, click the ingest domain that you want to configure. On the page that appears, choose **Stream Management > RTS**. On the RTS page, turn on RTS.
3. Generate an ingest URL.

   You can generate an ingest URL by using the **URL generator**. For more information, see Use the URL generator.
4. Change the protocol header of the ingest URL to `artc`.
5. Pass the ingest URL to AliLive SDK to ingest streams.

Usage notes

Stream ingest over RTS has the following limits compared with stream ingest over RTMP:

- Supports only the mono mode.
- Does not support the automatic ingest of images in poor network conditions.
- Supports only Low Complexity (LC) for audio coding.
- Supports only 48kHz for the audio sample rate.