# 简介

IoTVideoSDK是基于Cloudlink™ P2P的智能家居平台工具集。SDK将监控、录像、回放、设备控制、设备通讯、设备报警等功能进行封装，方便合作伙伴集成。

**SDK及Demo示例**

| 名称 | 位于 | URL |
| --- | --- | --- |
| IoTVideo-iOS Demo | IoTVideo-iOS/Demo/ | 前往 |
| IoTVideo-iOS SDK | IoTVideo-iOS/SDK/Frameworks/ | 前往 |

# 快速开始

## 第一步：集成

这一步我们学习如何快速地将IoTVideoSDK集成到您的项目中并配置工程依赖，按照如下步骤进行配置，就可以完成SDK的集成工作。

> 详见 集成指南

## 第二步：接入准备

在开始使用SDK之前我们还需要获得`productId`,`ivCid`,`accessId`和`ivToken`。

> **productId**： app的产品id(从平台注册时获取)
> **ivCid**： 客户id(从平台注册时获取)
> **accessId**： 是外部访问IoTVideo云平台的唯一性身份标识
> **ivToken**： 登录成功后IoTVideo云服务器返回的`ivToken`。

### 1. 获取ProductId和ivCid

在注册腾讯`IoTVideo`云平台时获取。

### 2. 获取accessId和ivToken

用户自有账号体系可以采用云云对接的方式实现账户体系相关业务。
详见《 厂商云对接IoTVideo平台接口定义 》

## 第三步：SDK初始化

### 1. 初始化

腾讯云平台上获取cid 和 productId,在您的 App 调用 SDK 相关功能之前（建议在AppDelegate类中）进行如下设置：

在`AppDelegate`中导入`IotVideo`并调用注册方法

- Swift

```
import IoTVideo

func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
    // Override point for customization after application launch.
    IoTVideo.sharedInstance.setupIvCid("xxx", productId: "xxxxxxxxxx",
userInfo: nil)

    return true
}
```

- Objective-C

```
#import <IoTVideo/IoTVideo.h>

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions {
    // Override point for customization after application launch.
    [IoTVideo.sharedInstance setupIvCid:@"xxx" productId:@"xxxxxxxxxxxx"
userInfo:nil];

    return YES;
}
```

## 2. 注册

账号注册登录获取到accessId、ivToken后，调用sdk注册接口

- Swift

```
import IoTVideo
IoTVideo.sharedInstance.register(withAccessId: accessId, ivToken: ivToken)
```

- Objective-C

```
#import <IoTVideo/IoTVideo.h>

[IoTVideo.sharedInstance registerWithAccessId:@"xxxxxxxx" ivToken:@"xxxxxx"];
```

⚠️注意：对设备的操作都依赖于上述四个参数的加密校验，非法参数将无法操作设备

# 第四步：配网

通过SDK初始化 我们已经可以正常使用SDK，现在我们可以为设备配置上网环境。

## 1.设备联网

设备配网模块用来为设备配置上网环境，目前支持以下配网方式:

- 有线配网
- 扫码配网
- AP配网

⚠️注意：并非任意设备都支持以上所有配网方式，具体支持的配网方式由硬件和固件版本决定。

> 详见《设备配网》

## 2.设备绑定

⚠️注意：设备配网完成后请参考《厂商云对接IoTVideo平台接口定义》绑定接口，将设备绑定至账户。

# 第五步：监控

使用内置的多媒体模块可以轻松实现设备监控。

```
// 1.创建监控播放器
let monitorPlayer = IVMonitorPlayer(deviceId: device.deviceID)
// 2.设置播放器代理（回调）
monitorPlayer.delegate = self
// 3.添加播放器渲染图层
videoView.insertSubview(monitorPlayer.videoView!, at: 0)
monitorPlayer.videoView?.frame    = videoView.bounds
// 4.预连接，获取流媒体头信息
monitorPlayer.prepare()
// 5.开始播放，启动推拉流、渲染模块
monitorPlayer.play()
// 6.开启/关闭语音对讲（只支持MonitorPlayer/LivePlayer）
monitorPlayer.startTalk()
monitorPlayer.stopTalk()
// 7.停止播放，断开连接
monitorPlayer.stop()
```

> 详见《多媒体》

# 第六步：消息管理

```
import IVCore.IVMessageMgr

// 设备ID的字符串
```

```
let deviceId = dev.deviceId
// 模型路径的字符串
let path = "CO._cameraOn"
// 模型参数的字符串
let json = "{\"ctlVal\":1}"

// 1.物模型获取
IVMessageMgr.sharedInstance.getDataOfDevice(deviceId, path: path) { (json, error) in
    // do something here
}

// 2.物模型设置
IVMessageMgr.sharedInstance.setDataToDevice(deviceId, path: path, json: json) { (json, error) in
    // do something here
}
```

> 详见《消息管理》

# 第七步：云存储

云存储功能详见《厂商云对接IoTVideo平台接口定义》的云存储接口。

jj

# 集成指南

本文主要介绍如何快速地将IoTVideoSDK集成到您的项目中，按照如下步骤进行配置，就可以完成 SDK 的集成工 作。

## 开发环境要求

- Xcode 8.0+
- iOS 9.0+

## 一、 集成 SDK

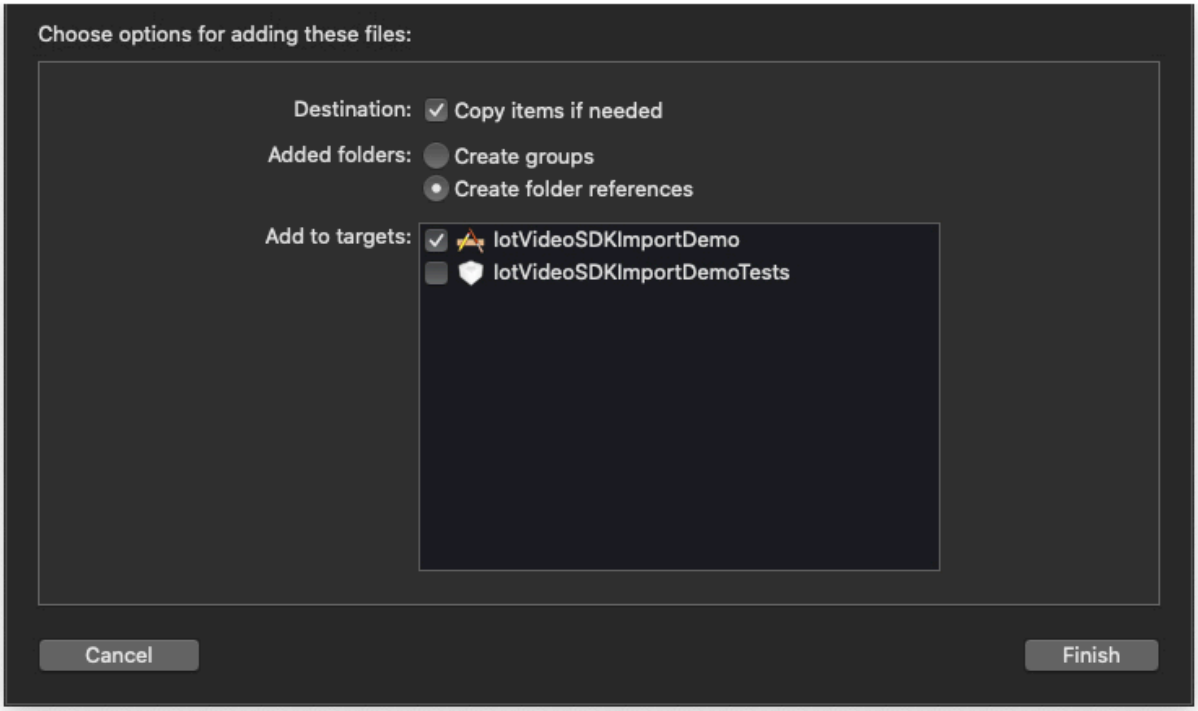
## 方法1: cocopods(推荐)

// TODO

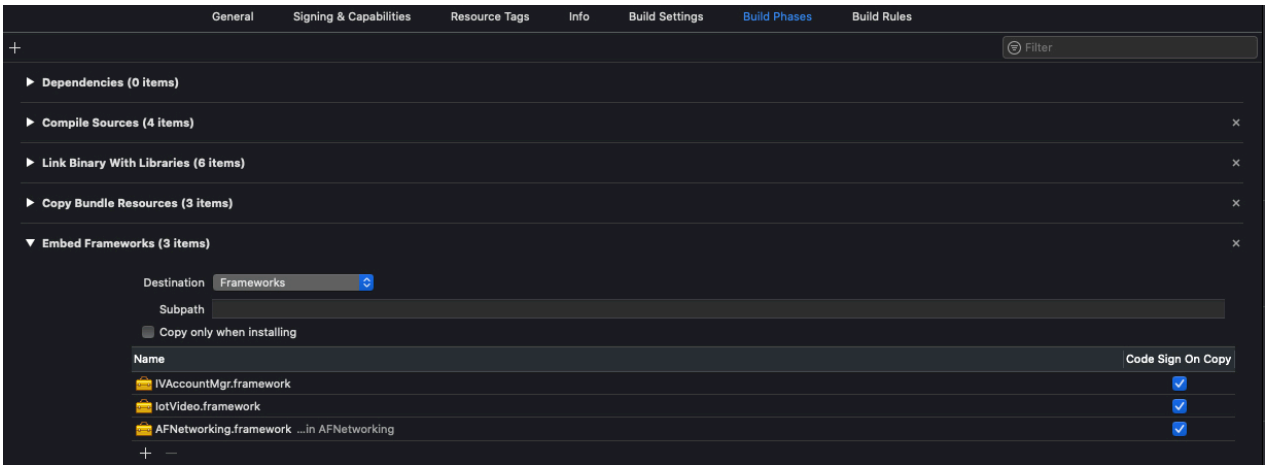
## 方法2：手动导入

添加IoTViedoSDK 的 Framework - IoTVideo.framework
- IVAccountMgr.framewok *//(可选)使用Gwell账号密码登录时：必须导入*

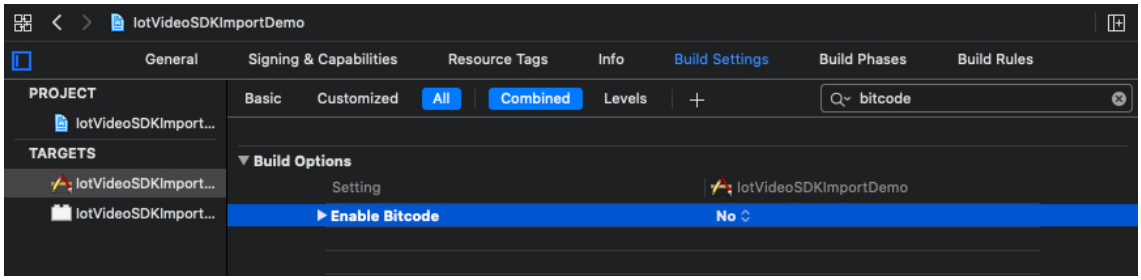
**手动导入SDK时，拖入项目时需要选择Copy items if needed 和 create folder references，如下图所示**




**需要设置TATGETS -> Build Phases -> Embed Frameworks为 Embed & sign**




- **添加系统的 Framework**
  - AudioToolbox.framework
  - VideoToolbox.framework
  - CoreMedia.framework
  - libz.tbd
- **添加其他第三方库依赖**
  - AFNNetWorking 3.0+ //*添加方法参考GitHub*
- **其他设置**
  - 关闭bitcode： TARGETS -> Build Settings -> Build Options -> Enable Bitcode -> NO




# 二、 接入准备

在开始使用SDK之前我们还需要获得`productId`,`ivCid`,`accessId`和`ivToken`。

> **productId**： app的产品id(从平台注册时获取)
> **ivCid**： 客户id(从平台注册时获取)
> **accessId**： 是外部访问IoTVideo云平台的唯一性身份标识
> **ivToken**： 登录成功后IoTVideo云服务器返回的`ivToken`。

## 1、 `productId`、`ivCid`在注册腾讯`IoTVideo`云平台时获取。

## 2、使用以下方式获得`accessId`和`ivToken`：

采用云云对接的方式实现账户体系相关，详见《厂商云对接IoTVideo平台接口定义》

# 三. SDK初始化

通过接入准备我们已经从IoTVideo云服务器获取了`productId`,`ivCid`,`accessId`和`ivToken`。

要使用IoTVideo SDK必先初始化后，所有的接口才能使用。

## 1、在`AppDelegate`中导入`IotVideo`并调用注册方法

- Swift

```
import IoTVideo

func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
    // Override point for customization after application launch.
    IoTVideo.sharedInstance.register(withProductId: "xxxxxxxxxxxx", ivCid:
"xxx", userInfo: nil)

    return true
}
```

- Objective-C

```
#import <IoTVideo/IoTVideo.h>

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Override point for customization after application launch.
    [IoTVideo.sharedInstance registerWithProductId:@"xxxxxxxxxxxx" ivCid:@"xxx" userInfo:nil];

    return YES;
}
```

### 2、在获取到登录授权信息后，注册ivToken和accessId

- Swift

```
import IoTVideo

IoTVideo.sharedInstance.setupToken(ivToken, accessId: accessId)
```

- Objective-C

```
#import <IoTVideo/IoTVideo.h>

[IoTVideo.sharedInstance setupToken:ivToken accessId:accessId];
```

⚠️注意：对设备的操作都依赖于上述四个参数的加密校验，非法参数将无法操作设备

# 设备配网

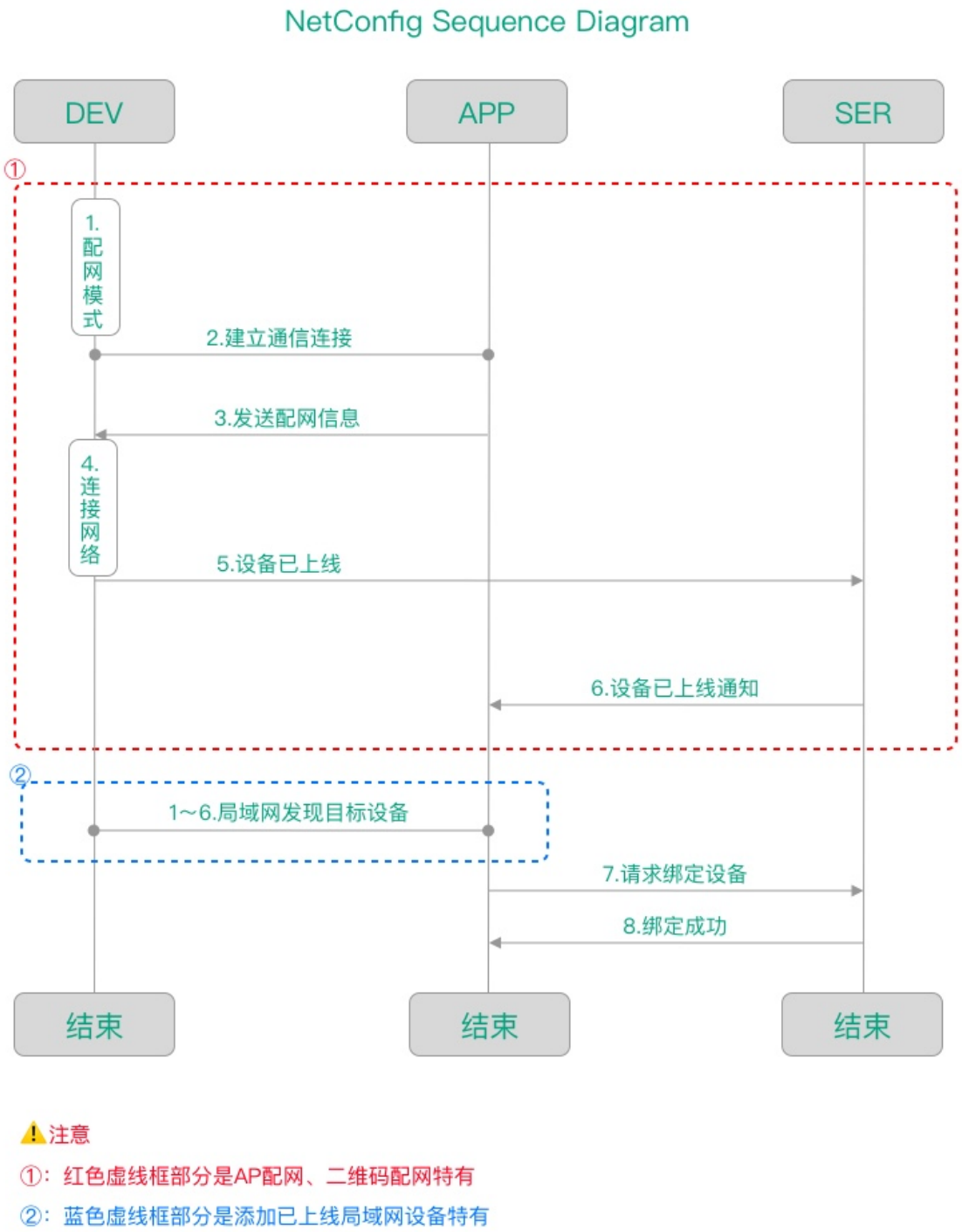
## 简介

设备配网模块用来为设备配置上网环境，目前支持以下配网方式:

- 有线配网
- 扫码配网
- AP配网

⚠️注意：并非任意设备都支持以上所有配网方式，具体支持的配网方式由硬件和固件版本决定。

**通用配网流程**




**有线配网（添加已在线局域网设备，需设备硬件支持）**

部分设备可通过自带网口使用有线上网，省去了配网环节，APP可通过局域网搜索到目标设备，使用设备ID向服务器发起绑定请求。
流程大致如下：

> 1. APP连接到与设备同一网络下的Wi-Fi
> 2. APP搜索到目标设备，取得目标设备ID
> 3. APP向服务器发起绑定目标设备的请求
> 4. 账户绑定设备成功
> 5. 配网结束

**AP配网（需设备硬件支持）**

AP配网原理是APP连接设备发射的热点，使设备与APP处于同一局域网，并在局域网下实现信息传递。
流程大致如下：

> 1. 设备复位进入配网模式并发射Wi-Fi热点
> 2. APP连接设备的热点（进入局域网）
> 3. APP向设备发送配网信息（Wi-Fi信息）
> 4. 设备收到配网信息并连接指定网络
> 5. 设备上线并向服务器注册
> 6. APP收到设备已上线通知
> 7. APP向服务器发起绑定目标设备的请求
> 8. 账户绑定设备成功
> 9. 配网结束

**二维码配网**

二维码配网原理是APP使用配网信息生成二维码，设备通过摄像头扫描二维码获取配网信息。
流程大致如下：

> 1. 设备复位进入配网模式，摄像头开始扫描二维码
> 2. APP使用配网信息生成二维码
> 3. 用户使用设备扫描二维码
> 4. 设备获取配网信息并连接指定网络
> 5. 设备上线并向服务器注册
> 6. APP收到设备已上线通知
> 7. APP向服务器发起绑定目标设备的请求
> 8. 账户绑定设备成功
> 9. 配网结束

# 使用示例

**1.有线配网**

```
import IVCore.IVLanNetConfig

// 1.获取局域网设备列表
let deviceList: [IVLANDevice] = IVLanNetConfig.getDeviceList()
// 2.取得目标设备
let dev = deviceList[indexPath.row]
// 3.绑定设备
```

绑定设备方法请参考《厂商云对接IoTVideo平台接口定义》的绑定接口。

**2.AP配网**

```
import IVCore.IVLanNetConfig

// 1.向服务器请求配网ID

// 2.连接设备热点如`AP-99999999`
```

```
// 3.发送配网信息
IVLanNetConfig.sendWifiName("TP-LINK12345", wifiPwd: "12345678", toDevice:
"99999999") { (success, error) in
    if success {
        // 发送成功
    } else {
        // 发送失败
    }
}

// 4.等待设备上线通知...

// 5.绑定设备
```

绑定设备方法请参考《厂商云对接IoTVideo平台接口定义》的绑定接口。

**3.二维码配网**

接入方自定义传递给设备的数据格式，可使用内置工具类生成二维码，也可自行生成二维码

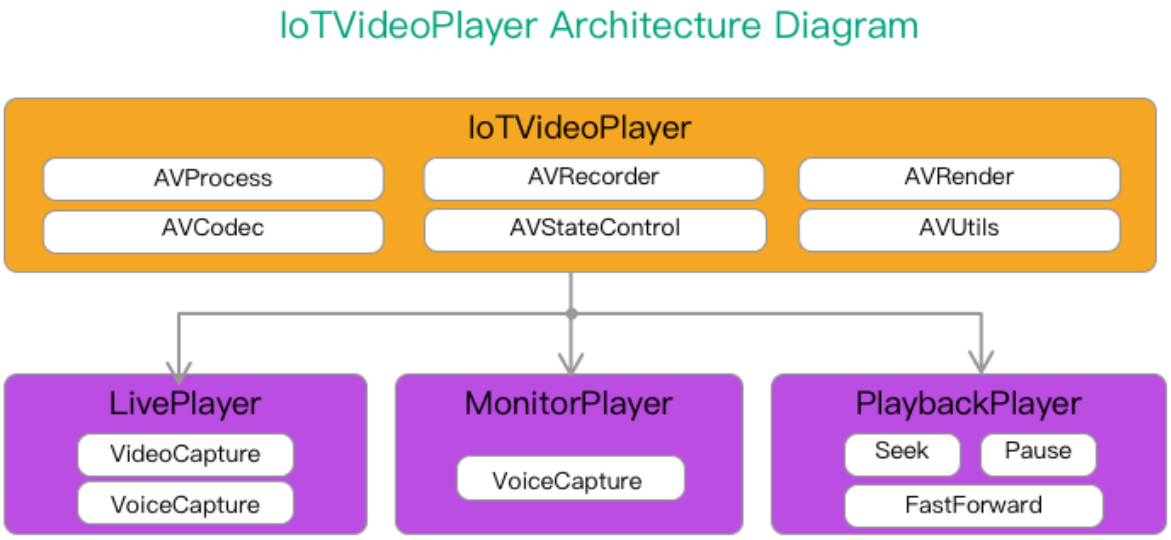
```
import IoTVideo.IVQRCodeNetConfig

// 1.生成二维码图片

 IVQRCodeNetConfig.createQRCode(withWifiName: wifiName, wifipwd: wifiPwd,
qrSize: size, completionHandler: { (image, error)  in
     // get the image
 })

// 2.用户使用设备扫描二维码....

// 3.等待设备上线通知...

// 4.绑定设备
```

绑定设备方法请参考《厂商云对接IoTVideo平台接口定义》的绑定接口。

# 多媒体

## 简介

多媒体模块为SDK提供音视频能力，包含实时监控、实时音视频通话、远程回放、录像、截图等功能。





**播放器核心(IVPlayer)**

IVPlayer是整个多媒体模块的核心，主要负责以下流程控制：

- 音视频通道建立
- 音视频流的推拉
- 协议解析
- 封装和解封装
- 音视频编解码
- 音视频同步
- 音视频渲染
- 音视频录制
- 播放状态控制

其中，*音视频编解码* 和 *音视频渲染* 流程允许开发者自定义实现（⚠️*播放器已内置实现，不推荐自定义实现*）

**监控播放器(MonitorPlayer)**

MonitorPlayer是基于IotVideoPlayer封装的监控播放器，主要增加以下功能：

- 语音对讲

**音视频通话播放器(LivePlayer)**

LivePlayer是基于IotVideoPlayer封装的音视频通话播放器，主要增加以下功能：

- 语音对讲
- 双向视频

**回放播放器(PlaybackPlayer)**

PlaybackPlayer是基于IotVideoPlayer封装的回放播放器，主要增加以下功能：

- 暂停/恢复
- 跳至指定位置播放
- 快进

**播放器功能对比**

| 功能 | 监控播放器 | 回放播放器 | 音视频通话 |
| --- | --- | --- | --- |
| 视频播放 | ✓ | ✓ | ✓ |
| 音频播放 | ✓ | ✓ | ✓ |
| 暂停/恢复 | x | ✓ | x |
| 跳至指定位置播放 | x | ✓ | x |
| 总时长 | x | ✓ | x |
| 当前播放进度 | x | ✓ | x |
| 播放器状态变更通知 | ✓ | ✓ | ✓ |
| 静音 | ✓ | ✓ | ✓ |
| 快进 | x | ✓ | x |
| 画面缩放模式设置 | ✓ | ✓ | ✓ |
| 播放器截图 | ✓ | ✓ | ✓ |
| 边播边录 | ✓ | ✓ | ✓ |
| 对讲 | ✓ | ✓ | ✓ |
| 分辨率切换 | ✓ | x | x |
| 双向视频 | x | x | ✓ |

# 使用示例

**1.导入播放器所在模块头文件**

```
import IVCore
```

**2.创建播放器实例**

⚠️*注意：以下使用`xxxxPlayer`泛指支持该功能的播放器*

```
// 监控播放器
let monitorPlayer = IVMonitorPlayer(deviceId: device.deviceID)
// 音视频通话播放器
let livePlayer = IVLivePlayer(deviceId: device.deviceID)
// 回放播放器
let playbackPlayer = IVPlaybackPlayer(deviceId: device.deviceID, startTime:
time)
```

**3.设置播放器代理（回调）**

```
xxxxPlayer.delegate = self
```

**4.添加摄像头预览图层(只支持LivePlayer)**

```
previewView.layer.addSublayer(livePlayer.previewLayer)
livePlayer.previewLayer.frame = previewView.bounds
```

**5.添加播放器渲染图层**

```
videoView.insertSubview(xxxxPlayer.videoView!, at: 0)
xxxxPlayer.videoView?.frame    = videoView.bounds
```

**6.预连接，获取流媒体头信息**

```
xxxxPlayer.prepare()
```

**7.开始播放，启动推拉流、渲染模块**

```
xxxxPlayer.play()
```

**8.开启/关闭语音对讲（只支持MonitorPlayer/LivePlayer）**

```
xxxxPlayer.startTalk()
xxxxPlayer.stopTalk()
```

**9.开启/切换/关闭摄像头（只支持LivePlayer）**

```
//打开摄像头
livePlayer.openCamera()
//切换摄像头
livePlayer.switchCamera()
//关闭摄像头
livePlayer.closeCamera()
```

**10.指定时间播放(只支持PlaybackPlayer)**

```
playbackPlayer.seekToTime(10);
```

**11.暂停/恢复播放(只支持PlaybackPlayer)**

```
//暂停
playbackPlayer.pause()
//恢复
playbackPlayer.resume()
```

**12.快进/取消快进(只支持PlaybackPlayer)**

```
//快进
playbackPlayer.fastForward();
//停止快进
playbackPlayer.cancelFastForward()
```

**13.停止播放，断开连接**

```
xxxxPlayer.stop()
```

# 高级功能

**自定义音视频编解码**

> ⚠️注意：该协议已默认由核心播放器实现。如无必要，无需另行实现。

可选实现播放器核心（IVPlayer）中的以下音视频编解码器即可自定义编解码器的目的：

```
@property (nonatomic, strong, nullable) id<IVVideoDecodable> videoDecoder;
@property (nonatomic, strong, nullable) id<IVVideoEncodable> videoEncoder;
@property (nonatomic, strong, nullable) id<IVAudioDecodable> audioDecoder;
@property (nonatomic, strong, nullable) id<IVAudioEncodable> audioEncoder;
```

**自定义音视频渲染器**

> ⚠️注意：该协议已默认由核心播放器实现。如无必要，无需另行实现。

可选实现播放器核心（IVPlayer）中的以下音视频渲染器即可自定义渲染器的目的：

```
@property (nonatomic, strong, nullable) id<IVAudioRenderable> audioRender;
@property (nonatomic, strong, nullable) UIView<IVVideoRenderable> *videoView;
```

# 消息管理

# 简介

消息管理模块负责APP与设备、服务器之间的消息传递，主要包含以下功能：

- 注册离线消息
  - 离线推送token上报
- 在线消息回调
  - 接收到事件消息（EVT）：告警、分享、系统通知
  - 接收到状态消息（ST）
- 控制/操作设备（CO）
- 设置设备参数（SP）
- 获取设备状态（ST）
- 获取设备参数（SP）
- 自定义消息透传

# 使用示例

**1.导入消息管理类所在模块**

```
import IVCore.IVMessageMgr
// 或
import IVCore
```

**2.状态和事件消息通知**

```
import IoTVideo.IVMessageMgr

class MyViewController: UIViewController, IVMessageDelegate {
    override func viewDidLoad() {
        super.viewDidLoad()
        // 设置消息代理
        IVMessageMgr.sharedInstance.delegate = self
    }

    // MARK: - IVMessageDelegate

    // 接收到事件消息（EVT）：  告警、分享、系统通知
    func didReceiveEvent(_ event: String, topic: String) {
        // do something here
    }

    // 接收到状态消息（ST）
    func didUpdateStatus(_ json: String, path: String, deviceId: String) {
        // do something here
    }
}
```

**3.获取物模型消息**

```
import IVCore.IVMessageMgr

// 设备ID的字符串
let deviceId = dev.deviceId
// 模型路径的字符串
let path = "CO._cameraOn"

IVMessageMgr.sharedInstance.getDataOfDevice(deviceId, path: path) { (json,
error) in
    // do something here
}
```

**4.发送物模型消息**

```
import IVCore.IVMessageMgr

// 设备ID的字符串
let deviceId = dev.deviceId
// 模型路径的字符串
let path = "CO._cameraOn"
// 模型参数的字符串
let json = "{\"ctlVal\":1}"

IVMessageMgr.sharedInstance.setDataToDevice(deviceId, path: path, json: json) {
(json, error) in
    // do something here
}
```

# 高级功能

**1.透传数据给设备**

```
    /// 透传数据给设备（无数据回传）
    /// 使用在不需要数据回传的场景，如发送控制指令
    /// @note 完成回调条件：收到ACK 或 消息超时
    /// @param deviceId 设备ID
    /// @param data 数据内容
    /// @param completionHandler 完成回调
    func sendData(toDevice deviceId: String, data: Data, withoutResponse
completionHandler: IVMsgDataCallback? = nil)


    /// 透传数据给设备（有数据回传）
    /// 使用在预期有数据回传的场景，如获取信息
    /// @note 完成回调条件：收到ACK错误、消息超时 或 有数据回传
    /// @param deviceId 设备ID
    /// @param data 数据内容
```

```
    /// @param completionHandler 完成回调
    func sendData(toDevice deviceId: String, data: Data, withResponse
completionHandler: IVMsgDataCallback? = nil)


    /// 透传数据给设备
    /// 可使用在需要数据回传的场景，如获取信息
    /// @note 可以等待有数据回传时才完成回调，如忽略数据回传可简单使用
`sendDataToDevice:data:completionHandler:`代替。
    /// @param deviceId 设备ID
    /// @param data 数据内容
    /// @param timeout 自定义超时时间，默认超时时间可使用@c `IVMsgTimeoutAuto`
    /// @param expectResponse 【YES】预期有数据回传 ； 【NO】忽略数据回传
    /// @param completionHandler 完成回调
    func sendData(toDevice deviceId: String, data: Data, timeout: TimeInterval,
expectResponse: Bool, completionHandler: IVMsgDataCallback? = nil)
```

**2.透传数据给服务器**

```
    /// 透传数据给服务器
    /// @param url 服务器路径
    /// @param data 数据内容
    /// @param completionHandler 完成回调
    func sendData(toServer url: String, data: Data?, completionHandler:
IVMsgDataCallback? = nil)


    /// 透传数据给服务器
    /// @param url 服务器路径
    /// @param data 数据内容
    /// @param timeout 超时时间
    /// @param completionHandler 完成回调
    func sendData(toServer url: String, data: Data?, timeout: TimeInterval,
completionHandler: IVMsgDataCallback? = nil)
```

# 云存储

云存储功能详见《厂商云对接IoTVideo平台接口定义》的云存储接口。