实时互动-工业能源版

现场设备 SDK

腾讯云

文档目录

现场设备 SDK
基本介绍

最近更新时间：2023-05-12 18:00:03

适用范围

本 SDK 用于现场设备接入，适用于 Linux 64位系统，提供 so 库形式接入。

主要功能

本 SDK 面向远程实时操控场景提供低时延音视频通话和控制/状态数据传输，主要功能如下：

视频处理

包含视频的采集、缩放、剪裁、拼接等功能。

视频编码

对于 Jetson 平台支持 H264、H265、AV1 等硬件编码类型，通用 x86_64 和 aarch64 平台支持 H264 软编。

视频传输

支持 H264\H265\AV1 等编码视频流的传输，具备抗弱网和低延迟传输能力。

二进制数据传输

支持二进制数据的透传，可向通话方传输二进制数据。

推流自管理

SDK 可以自管理推流状态，在拉流端有观看时进行推流，减少不必要的网络带宽使用。

断网自重连

SDK 可以自管理断网状态，在出现断网时，自动处理断网异常，尝试状态恢复。

基本概念

设备 ID

用于标识现场设备，具有唯一性。如果网络中有相同设备id的设备连网，会出现 MQTT 信令反复被踢下线又重连的情况。

视频流 ID

用于标识现场设备的视频流编号，编号从0开始递增，与 Json 配置中的 streams_config 数组元素编号一致。远端设备拉取视频流时，会通过设备 ID+ 视频流 ID 来指定要拉取的视频流。

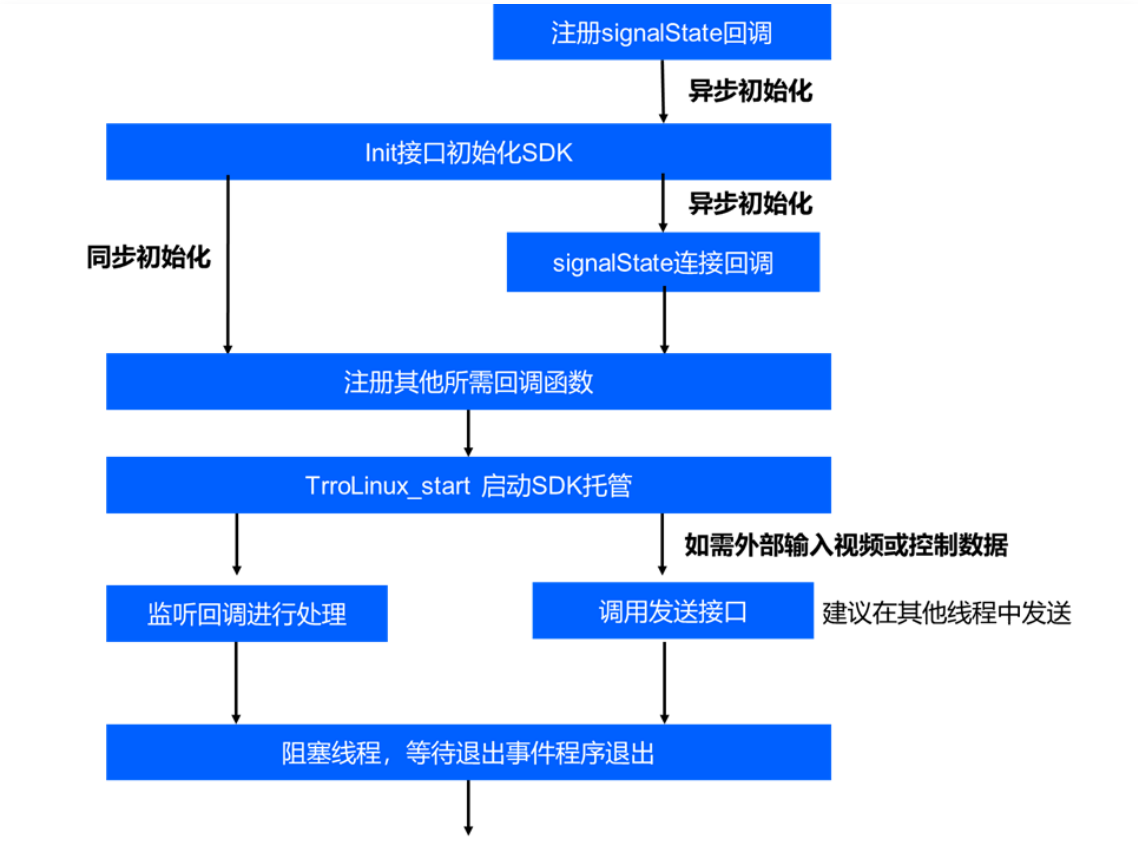
配置文件

config.json 文件，内含 SDK 的初始化配置。

SDK API 调用流程

最近更新时间：2024-04-18 14:15:01

API 调用流程图



调用步骤建议

> **注意：**
> signalStae 信令回调注册需要在初始化函数之前调用，其他回调函数可以在初始化之后注册。

步骤1：加载配置文件初始化

注册 signalState 信令回调接口，通过 Json 文件加载或输入 Json 字符串加载等方式，调用 Init 接口初始化 SDK。

步骤2：注册所需的回调函数

根据需求，注册控制消息回调、日志回调、连接状态回调、视频链路信息回调、延迟信息回调等函数，处理对应的状态信息和事件。

步骤3：启动视频 SDK 进入托管状态

调用 TrroLinux_start，进入托管状态。此时，SDK 会监听各类请求，自动进行视频采集、视频推流，并处理断网等事件。

步骤4：按需调用发送函数

当有外部视频数据或二进制数据输入需求时，在所需发送时刻，调用对应的发送接口输入要发送的视频数据或二进制控制/状态数据。

Sample 代码示例

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <thread>
#include "trro_field.h"

//外部输入视频数据
void externalVideoInput(int stream_id) {
    int data_width = 1280;
    int data_height = 720;
    char *data = (char*)malloc(data_width*data_height*3/2);
     //YUV420视频数据, sample中使用全0模拟数据（绿图）示意
    memset(data, 0, data_size);
    while(true) {
        TRRO_externalVideoData(stream_id, data, data_width, data_height, 0);
        usleep(33 * 1000);
    }
    free(data);
}

//主线程
int main() {
    //注册signal回调
    TRRO_registerSignalStateCallback(nullptr, [](void *context, SignalState state) {
        if(state == kTrroReady) {
            //服务器连接成功，注册回调函数，并启动SDK托管
            printf("init success \n");
            TRRO_registerControlDataCallback(nullptr, [](void * context, const char*
controlid, const char* msg, int len, int cmd) {
                printf("receive control data from %s: %s\n", controlid, msg);
            });
            TRRO_registerOnState(nullptr, [](void* context, int stream_id, int state) {
                printf("stream_id: %d, state: %d\n", stream_id, state);
            });
```

```
        TRRO_registerOnErrorEvent(nullptr, [](void* context, int error_code, const
char* error_msg) {
            printf("error_code %d, error_msg %s\n", error_code, error_msg);
        });
        TRRO_registerVideoCaptureCallback(nullptr, [](void *context, const char* data,
int width, int height, int type, int stream_id) {});
        TRRO_registerLatencyCallback(nullptr, [](void *context, int stream_id, int vcct)
{
            printf("LatencyCallbac context: %p, stream id %d, vcct %d\n", context,
stream_id, vcct);
        });
        TRRO_registerMediaState(nullptr, [](void* context, int stream_id, int fps, int
bps, int rtt, long long lost, long long packets_send, int stun) {
            printf("stream %d, fps %d, bps %d, rtt %d, lost %lld, packets_send %lld,
stun %d\n", stream_id, fps, bps, rtt, lost, packets_send, stun);
        });
        TRRO_registerOperationPermissionRequest(nullptr, [](void* context, const
char* remote_devid, int permission) {
            printf("remote devid %s permission %d\n", remote_devid, permission);
        });
        int ret = TRRO_start();
        //init 连接服务成功后, 可以开启 TRRO SDK 托管
        if(TRRO_SUCCED == ret) {
            printf("start succeed\n");
        } else if (ret == -TRRO_INIT_LICENSE_FILE_ERROR || ret == -
TRRO_INIT_LICENSE_CHECK_FAILED) {
            printf("start license error, wait for regist or exit error:%d\n", ret);
        } else {
            // 需注意配置问题
            printf("start config error, please check config it, error:%d\n", ret);
        }
    }
    if (state == kTrroAuthFailed) {
        printf("device_id or password is incorrect\n");
    }
    if (state == kTrroKickout) {
        printf("the device is kicked by server, may be there is another device using
the same device id\n");
    }
});

// 建议最后一个参数使用-1，采用非阻塞模式启动，等待信令连接成功回调
// 可按需替换为其他初始化函数，如使用配置字符串初始化
int ret = TRRO_initGwPathWithLicense("./config.json", "./license.txt", -1);
if(TRRO_SUCCED != ret) {
```

```
        if (ret == -TRRO_SIGNAL_CONNECT_OUTTIME) {
            printf("init process: wait for connecting\n");
        } else {
            printf("init fail ret %d\n", ret);
        }
    }

    // 启动外部输入视频数据线程，输入视频数据到流0，需要流0协议为outside
    // std::thread t1(externalVideoInput, 0);
    // t1.detach();

    //防止程序退出
    while(true){
        sleep(30000);
    }
    return 0;
}
```

C/C++ 现场设备 SDK API

最近更新时间：2024-03-18 17:59:11

API 描述

适用于 Linux 64位系统。头文件引用 trro_field.h，API 可参考头文件中接口注释。下面列举常用接口描述。

API 概览

初始化接口

函数列表	描述
TRRO_initGwJson	使用 json 字符串初始化
TRRO_initGwPath	使用配置文件路径初始化
TRRO_initGwPathWithLicense	使用配置路径及 license 路径初始化（私有化）
TRRO_initGwJsonWithLicense	使用 json 字符串和 license 路径初始化（私有化）

状态获取类接口

函数列表	描述
TRRO_registerSignalStateCallback	注册信令状态回调
TRRO_registerOnState	注册视频连接状态回调
TRRO_registerLatencyCallback	注册延迟状态回调
TRRO_registerMediaState	注册媒体状态回调
TRRO_registerEncodeFrameInfoCallback	注册外部编码建议信息回调（外部编码时使用）

启停类接口

函数列表	描述
TRRO_start	启动 SDK
TRRO_stop	停止 SDK

视频输入输出接口

函数列表	描述
TRRO_externalVideoData	输入外部源视频流（图像数据）
TRRO_registerVideoCaptureCallback	注册视频采集数据回调函数
TRRO_externalEncodeVideoData	输入外部源视频流（编码数据）

视频采集接口

函数列表	描述
TRRO_startVideoCapture	开始摄像头采集
TRRO_stopVideoCapture	停止摄像头采集

消息类接口

函数列表	描述
TRRO_sendControlData	向远端设备发送数据
TRRO_registerControlDataCallback	注册远端设备控制消息回调

权限控制类接口

函数列表	描述
TRRO_registerOperationPermissionRequest	注册远端设备操控权限请求通知回调
TRRO_setOperationPermission	设置远端设备操作权限

重配置接口

函数列表	描述
TRRO_reinitRtc	重配置现场设备视频流，仅支持重置normal/combine/rtsp_enc等扩展采集协议重配置

音频控制类接口

函数列表	描述

TRRO_audioMute	静音设置

日志/错误信息类接口

函数列表	描述
TRRO_registerLogCallback	注册日志回调函数
TRRO_registerOnErrorEvent	注册错误事件回调
getErrorMsg	错误码对应错误信息获取接口

录制相关接口

函数列表	描述
TRRO_startRecorder	启动录制
TRRO_sendRecordVideoData	发送录制数据
TRRO_switchRecorderFile	切换录制文件
TRRO_stopRecorder	结束录制

初始化接口

使用说明：可根据是否使用本地 license（公有云无需本地 license）以及配置输入类型（文件/字符串），选择对应的初始化接口进行初始化，只需使用一个。

使用 json 字符串初始化

使用说明：用户需确保 json 字符串格式正确，此接口用于 SDK 加载配置进行初始化。

```
/*
* @name :  TRRO_initGwJson
* @brief : 使用字符串初始化
* @input : json_str       配置文件json字符串
* @input : mode            0 同步模式，一直等待
*                      -1 异步模式，初始化成功后通知TRRO_onSignalState
* @return : 1 for success, other failed
*/
int TRRO_initGwJson(const char * json_str, int mode = 0);
```

参数	含义

json_str	配置信息 josn字符串
mode	返回模式。 • 0：同步阻塞模式 • 1：异步模式

使用配置文件路径初始化

使用说明：用户需确保 json 配置文件格式正确，此接口用于 SDK 加载配置进行初始化。

```
/*
* @name :  TRRO_initGwPath
* @brief : 使用配置文件初始化
* @input : cfg_path       json配置文件路径名，e.g. "./config.json"
* @input : mode           0  同步模式一直等待
*                    -1 异步模式，初始化成功后TRRO_onSignalState信息
* @return : 1 for success, other failed
*/
int TRRO_initGwPath(const char * cfg_path, int mode = 0);
```

参数	含义
cfg_path	配置文件路径
mode	返回模式。 • 0：同步阻塞模式 • 1：异步模式

使用配置文件路径及 license 文件路径初始化

使用说明：用户需确保 json 配置文件格式正确，license 文件路径正常，此接口用于 SDK 加载配置进行初始化。

```
/*
* @name :  TRRO_initGwPathWithLicense
* @brief : 使用配置文件和本地license初始化
* @input : cfg_path       json配置文件路径名，e.g. "./config.json"
* @input : license_path    license文件路径名，e.g. "./license.txt"
* @input : mode           0 同步模式一直等待
*                    -1 异步模式 初始化成功后通知TRRO_onSignalState
* @return : 1 for success, other failed
*/
```

```
int TRRO_initGwPathWithLicense(const char * cfg_path, const char * license_path, int
mode = 0);
```

参数	含义
cfg_path	配置文件名
license_path	本地 license 文件路径
mode	返回模式 • 0：同步阻塞模式 • 1：异步模式

使用配置字符串和 license 文件路径初始化

使用说明：用户需确保 json 字符串和 license 内容格式正确，此接口用于 SDK 加载配置进行初始化。

```
/*
* @name :  TRRO_initGwJsonWithLicense
* @brief : 使用字符串和本地license初始化
* @input : json_str        配置文件json字符串
* @input : license_path    license文件路径名，e.g. "./license.txt"
* @input : mode            初始化模式
*                      0 同步模式一直等待
*                     -1 异步模式, 初始化成功后通知TRRO_onSignalState
* @return : 1 for success, other failed
*/
int TRRO_initGwJsonWithLicense(const char * json_str, const char * license_path, int m
ode = 0);
```

参数	含义
json_str	配置文件 json 字符串
license_path	license 路径
mode	返回模式。 • 0：同步阻塞模式 • 1：异步模式

状态获取类接口

使用说明：按需注册所需功能的回调接口。

注册信令状态回调（异步初始化回调）

使用说明：此接口用于注册信令状态回调接口，异步模式初始化时需使用，根据 Ready 状态判断初始化成功。

```
/*
* @name : TRRO_registerSignalStateCallback
* @brief : 注册信令服务连接状态回调
* @input : context          上下文
          callback          回调函数
* @return : 1 for success, other failed
*/
int TRRO_registerSignalStateCallback(void *context, TRRO_onSignalState *callback);
```

参数	含义
context	上下文指针，回调时会返回该指针用于定位
callback	回调函数

```
回调定义
enum SignalState {
    kTrroReady = 0, /**< 连接建立成功 */
    kTrroLost = 1,  /**< 连接断开，内部会进行自动重连 */
    kTrroReup = 2,  /**< 自动重连成功 */
    kTrroKickout = 3,
    kTrroAuthFailed = 4,  /**< 用户名或者密码错误 */
};

/*
* @name : TRRO_onSignalState
* @brief : 信令连接状态回调
* @input : context       上下文
*          state         kTrroReady          连接建立成功
*                        kTrroLost           连接断开，内部会进行自动重连
*                        kTrroReup           自动重连成功
* @return : void
*/
typedef void TRRO_onSignalState(void *context, SignalState state);
```

参数	含义
context	回调上下文，返回注册时传入的 context

state	信令连接状态

注册视频连接状态回调

使用说明：此接口用于注册视频流连接状态回调，此接口可根据需求选择是否调用。

```
/**
* @name TRRO_registerOnState
* @brief 注册视频连接状态回调函数
* @param[in]   context          上下文
* @param[in]   callback         回调函数
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerOnState(void* context, TRRO_OnState *
callback);
```

参数	含义
context	上下文指针，回调时返回该上下文用于定位
callback	回调函数

```
回调函数定义
enum TrroState {
    kDisconnect = 0, /**< 断连 */
    kConnecting = 1, /**< 连接中 */
    kConnected = 2,  /**< 已连接 */
    kDisconnecting = 3, /**< 已断连 */
};

/**
* @name TRRO_onState
* @brief 视频连接状态回调
* @param[in]    context      上下文
* @param[in]    stream_id    视频流id
* @param[in]    state        TrroState连接状态
* @return void
*/
typedef void TRRO_OnState(void* context, int stream_id, int state);
```

注册时延回调

使用说明：此接口用于在现场设备侧获取推流到拉流的视频数据处理延时，可按需选择使用。

```
/**
* @name TRRO_registerLatencyCallback
* @brief 注册时延回调函数
* @param[in]   context          上下文
* @param[in]   callback         回调函数
* @return  1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerLatencyCallback(void *context,
TRRO_onLatencyReport *callback);
```

参数	含义
context	上下文指针，回调时会返回该指针用于定位
callback	回调函数

```
/**
* @name TRRO_onLatencyReport
* @brief 延迟信息回调
* @param[in]    context      上下文
* @param[in]    stream_id    流ID
* @param[in]    vcct         视频控制闭环时延， 等于视频上行延迟（不含采集）+控制下行延
迟
* @return void
*/
typedef void TRRO_onLatencyReport(void *context, int stream_id, int vcct);
```

参数	含义
context	上下文指针，回调时会返回该指针用于定位
stream_id	流 ID
vcct	视频控制闭环时延， 等于视频上行延迟（不含采集）+控制下行延迟

注册媒体状态回调

使用说明：此接口用于在现场设备获取已连接视频流的媒体状态，可按需选择使用。

```
/**
* @name  TRRO_registerMediaState
* @brief 注册网络状态
* @param[in]   context          上下文
```

```
* @param[in]   callback            回调函数
* @return  1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerMediaState(void* context,
TRRO_onMediaState * callback);
```

参数	含义
context	上下文指针，回调时会返回该指针用于定位
callback	回调函数

```
/**
* @name TRRO_onMediaState
* @brief 媒体传输状态回调
* @param[in]    context      回调上下文指针
* @param[in]    stream_id    流ID
* @param[in]   fps          每秒帧数目
* @param[in]   bps          每秒数据量
* @param[in]   rtt          封包来回时间
* @param[in]   lost         总丢包数目
* @param[in]   packets_send 总发送数目
* @param[in]   stun         穿网模式 0：host, 1：srflx, 2：prflx, 3：relay
* @return void
*/
typedef void TRRO_onMediaState(void* context, int stream_id, int fps, int bps, int rtt,
long long lost, long long packets_send, int stun);
```

参数	含义
context	上下文指针，回调时会返回该指针用于定位
stream_id	流 ID
fps	每秒帧数目
bps	每秒数据量
rtt	封包来回时间
lost	总丢包数目
packets_send	总发送数目
stun	穿网模式

	• 0：host • 1：srflx • 2：prflx • 3：relay

注册编码建议信息回调

使用说明：此接口配合外部编码流输入数据接口共同使用，用户需可根据此接口的回调信息，对输入的编码流进行处理。

```
/**
* @name TRRO_registerEncodeFrameInfoCallback
* @brief 注册编码建议信息回调函数
* @param[in]    context            上下文
* @param[in]    callback           回调函数
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerEncodeFrameInfoCallback(void *context,
TRRO_onEncodeFrameInfo *callback);
```

参数	含义
context	上下文指针，回调时返回该上下文指针用于定位
callback	回调函数

回调函数定义

```
/**
* @name TRRO_onEncodeFrameInfo
* @brief 编码建议信息回调，适用于外部输入编码帧场景
* @param[in]    context       上下文
* @param[in]    stream_id     流ID
* @param[in]    type          回调类型， 0：强制关键帧请求 ， 1 码率更新请求
* @param[in]    bitrate       type为1时有效，表示建议输入的编码数据码率，单位kbps
* @return void
*/
typedef void TRRO_onEncodeFrameInfo(void *context, int stream_id, int type, int
bitrate);
```

参数	含义
context	回调上下文，返回注册时传入的 context

strem_id	流 ID
type	回调类型。 • 0：强制关键帧请求 • 1：码率更新请求
bitrate	当 type 为1时有效，表示建议输入的编码数据码率，单位 kpbs

启停类接口

启动接口

使用说明：此接口用于启动现场设备 SDK，初始化成功后方可调用，即同步模式初始化返回成功，或者异步模式初始化 TRRO_onSignalState 通知连接 Ready。

```
/**
* @name TRRO_start
* @brief 启动音视频传输业务， 需要等待初始化成功后调用（同步模式init返回成功 或 异步模式初始化TRRO_onSignalState通知连接Read）
* @param[in] void
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_start();
```

停止接口

使用说明：此接口用于停止现场设备 SDK。

```
/**
* @name TRRO_stop
* @brief 销毁SDK，释放sdk资源
* @param[in]  void
* @return void
*/
extern "C" TRRO_EXPORT void TRRO_stop();
```

视频输入输出接口

输入外部源视频流（图像数据）

使用说明：此接口用于提供用户输入视频的原始图像数据流，注意输入数据源需与配置文件中的流数目、流配置节点相匹配。

输入的数据源 width、height 需与 现场设备配置说明 中的 width、height 一致，同时配置文件中“protocol”节点值修改为“outside”。

```
/**
* @name   TRRO_externalVideoData
* @brief  外部图像输入接口
* @param[in]   stream_id     流ID
* @param[in]   data          消息体
* @param[in]   width         数据源宽
* @param[in]   height        数据源高
* @param[in]   type          数据源类型，当前支持Trro_ColorYUVI420 Trro_ColorJPEG
Trro_ColorYUYV
* @param[in]   dataSize      数据大小，为0时会自动根据格式计算大小，发送JPEG等无特定大
小数据时需指定
* @param[in]   text          图像要叠加字符串(experimental)
* @param[in]   point_x       叠加文字起始x坐标，最左侧为0 (experimental)
* @param[in]   point_y       叠加文字起始y坐标，最顶部为0 (experimental)
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_externalVideoData(int stream_id, const char *
data, int width, int height, int type, int dataSize = 0, const char* text = "", int point_x
= 0, int point_y = 0);
```

参数	含义
stream_id	视频流 ID，从0开始，第2路为1，以此递增
data	图像数据
width	图像宽
height	图像高
type	图像类型。 • 0：YUVI420 • 4：YUYV • 5：JPEG，推荐 YUVI420 格式
dataSize	图像数据大小，0为自动计算，发送 jpeg 等无固定大小图像格式需指定
text	保留
point_x	保留

point_y	保留

注册视频采集数据回调函数

使用说明：此接口用于获取SDK内部采集的相机数据源，供上层业务使用。

```
/**
* @name  TRRO_registerVideoCaptureCallback
* @brief 注册视频采集数据回调函数
* @param[in]   context          上下文
* @param[in]   callback         回调函数
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerVideoCaptureCallback(void* context,
TRRO_onVideoCaptureData * callback);
```

参数	含义
context	上下文指针，回调时会返回该指针用于定位
callback	回调函数

```
/**
* @name TRRO_onVideoCaptureData
* @brief 采集视频帧回调
* @param[in]    context      上下文指针，返回注册时传入的context
* @param[in]    data         视频数据
* @param[in]    width        宽
* @param[in]    height       高
* @param[in]    type         视频格式,0 YUV420, 4 YUYV
* @param[in]    stream_id    流号
* @return void
*/
typedef void TRRO_onVideoCaptureData(void *context, const char* data, int width, int
height, int type, int stream_id);
```

参数	含义
remote_devid	设置权限的对端设备 ID
data	视频数据
width	视频数据宽

height	视频数据高
type	视频格式： • 0 YUV420 • 4 YUYV
stream_id	流号

输入外部源视频流（编码数据）

> **说明：**
> 建议采用单 slice 编码 I 帧带 sps/pps。

使用说明：此接口用于提供用户输入已编码视频流进行传输，注意输入数据源需与配置文件中的流数目、流配置节点相匹配。输入视频流编码类型（例如 H264）需要与配置文件中对应流的 codec 一致，关键帧中包含 SPS/PPS 信息，同时配置文件中“protocol”节点值修改为“outenc”。

```
/**
* @name TRRO_externalEncodeVideoData
* @brief 外部编码流输入，编码流codec需要与配置codec一致
* @param[in]    stream_id          流ID
* @param[in]    data               消息体
* @param[in]    width              数据源宽
* @param[in]    height             数据源高
* @param[in]    size               数据大小
* @param[in]    type               数据源类型，参考FrameType
* @return  1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_externalEncodeVideoData(int stream_id, const
char * data, int width, int height, int size, FrameType type);
```

参数	含义
stream_id	视频流 ID，从0开始，第2路为1，以此递增
data	视频流数据
width	视频款
height	视频高
size	视频帧大小
type	编码类型。

	• 0：P 帧 • 1：I 帧（需包含 SPS/PPS 信息）

消息类接口

向远端设备发送数据

使用说明：此接口用于发送二进制数据到远端设备，配合远端设备 SDK 数据回调接口使用，此接口可根据需求选择是否调用。目前限制单次发送数据大小700字节，频率限制100次/秒。

```
/**
* @name   TRRO_sendControlData
* @brief  向远端设备发送数据
* @param[in] msg          消息体
* @param[in] len          消息体长度
* @param[in] qos          0:unreliable 1:reliable
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_sendControlData(const char* msg, int len, int qos
= 0);
```

参数	含义
msg	二进制消息内容
len	消息长度
qos	发送 qos 类型。 • 0：非可靠传输 • 1：可靠传输

注册远端设备控制消息回调

使用说明：此接口用于注册远端设备 SDK 发送的控制数据回调函数，可根据需要实现。

```
/**
* @name TRRO_registerControlDataCallback
* @brief 注册远端设备消息回调函数
* @param[in]   context          上下文指针，回调时会返回该上下文指针
* @param[in]   callback         回调函数
* @return 1 for success, other failed
*/
```

```
extern "C" TRRO_EXPORT int TRRO_registerControlDataCallback(void* context,
TRRO_onControlData * callback);
```

参数	含义
context	上下文指针，回调时会返回该上下文指标用于定位
callback	回调函数

回调函数定义

```
/**
* @name TRRO_onControlData
* @brief 接收远端设备消息回调
* @param[in]    context       上下文指针，返回注册时传入的context
* @param[in]    controller_id  远端设备ID
* @param[in]    msg            消息体内容
* @param[in]    len            消息体长度
* @param[in]    qos            消息qos类型 0:unreliable, 1:reliable
* @return void
*/
typedef void TRRO_onControlData(void *context, const char *controller_id, const char*
msg, int len, int qos);
```

参数	含义
context	回调上下文，返回注册时传入的 context
controller_id	远端设备 ID
msg	控制消息字符串
len	字符串长度
qos	消息 qos 类型： • 0：不可靠传输 • 1：可靠传输

权限控制类接口

注册远端设备操控权限请求通知回调

使用说明：此接口用于在现场设备接收权限控制请求，可按需选择使用。

```
/**
```

```
* @name  TRRO_registerOperationPermissionRequest
* @brief 注册远端设备操控权限请求通知回调
* @param[in]   context            上下文
* @param[in]   callback           回调函数
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerOperationPermissionRequest(void
*context, TRRO_onOperationPermissionRequest *callback);
```

参数	含义
context	上下文指针，回调时会返回该指针用于定位
callback	回调函数

```
/**
* @name TRRO_onOperationPermissionRequest
* @brief 远端设备操控权限申请通知
* @param[in]    remote_devid       请求权限的remote deviceId
* @param[in]    permission         请求的权限，参考TrroPermission，  0: guest 只有观看权
限，1: master 完全控制权限
* @return void
*/
typedef void TRRO_onOperationPermissionRequest(void* context, const char*
remote_devid, int permission);
```

参数	含义
context	上下文指针，回调时会返回该指针用于定位
remote_devid	请求权限的 remote deviceId
permission	请求的权限，参见 TrroPermission • 0: guest 只有观看权限 • 1: master 完全控制权限

设置远端设备操作权限

使用说明：此接口用于在改现场设备设置远端设备的操作权限。

```
/**
 * @name  TRRO_setOperationPermission
```

```
 * @brief 设置远端设备操控权限，目前同时只能有一个远端设备有master权限，若已有远端设备是master权限，调用该接口设置master权限，会自动取消之前设备的master权限然后设置新设备；
 * @param[in]  remote_devid        设置权限的对端设备id
 * @param[in]  permission          参考 TrroPermission， 0 guest，只有观看权、1 master， 完全控制权限
 * @return 1 for success, other failed
 */
extern "C" TRRO_EXPORT int TRRO_setOperationPermission(const char* remote_devid, int permission);
```

参数	含义
context	上下文指针，回调时会返回该指针用于定位
callback	回调函数

重配置接口

重置RTC配置

使用说明：重配置视频流配置

```
/**
* @name  TRRO_reinitRtc(Experimental)
* @brief 重配置，当前仅支持重置normal/combine/rtsp_enc等扩展采集协议配置
* @param[in] config   重配置json数据
*/
extern "C" TRRO_EXPORT int TRRO_reinitRtc(const char * config);
```

参数	含义
config	重配置 json 数据

音频控制接口

静音设置

使用说明：关闭音频传输

```
/**
* @name  TRRO_audioMute(Experimental)
* @brief Mute 拉流端的音频 仅Server模式
* @param[in] userid  拉流端id
```

```
* @param[in] mute    是否mute
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_audioMute(const char* userid, bool mute);
```

参数	含义
userid	拉流端 ID
mute	是否 mute

日志/错误信息类接口

注册日志回调函数

使用说明：此接口用于注册日志回调函数，供上层业务获取日志。

```
/**
* @name  TRRO_registerLogCallback
* @brief 注册日志回调函数
* @param[in]   context            上下文
* @param[in]   callback           回调函数
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerLogCallback(void *context,
TRRO_OnLogData *callback);
```

参数	含义
context	上下文指针，回调时会返回该指针用于定位
callback	回调函数

```
/**
* @name TRRO_OnLogData
* @brief 日志回调
* @param[in]    context       上下文
* @param[in]    msg           日志内容
* @param[in]    level         日志级别,参考枚举TrroLogLevel
* @return void
*/
typedef void TRRO_OnLogData(void *context, const char *msg, int level);
```

参数	含义
context	上下文指针，回调时会返回该指针用于定位
msg	日志内容
level	日志级别,参考枚举 TrroLogLevel

注册错误事件回调

使用说明：此接口用于注册错误事件回调，当推流端出现异常时触发回调，可按需选择使用。

```
/**
* @name TRRO_registerOnErrorEvent
* @brief 注册错误事件回调函数
* @param[in] context              上下文
* @param[in] callback             回调函数
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerOnErrorEvent(void* context,
TRRO_OnErrorEvent * callback);
```

参数	含义
context	上下文指针，回调时会返回该指针用于定位
callback	回调函数

```
/**
* @name TRRO_OnErrorEvent
* @brief 错误信息回调
* @param[in] context        上下文
* @param[in] error_code     错误码  为负值
* @param[in] error_msg      错误信息
* @return void
*/
typedef void TRRO_OnErrorEvent(void* context, int error_code,  const char*
error_msg);
```

参数	含义
context	上下文指针，回调时会返回该指针用于定位
error_code	错误码

error_msg	错误信息

错误信息查询

使用说明：查询错误码对应的错误信息。

```
/**
* @name  getErrorMsg
* @brief 根据错误码，返回错误信息。
* @param[in] errorCode   错误码
* @return errorMsg   排查帮助信息
*/
extern "C" TRRO_EXPORT const char* getErrorMsg(int errorCode);
```

参数	含义
errorCode	要查询的错误码
errorMsg	对应错误信息

录制类接口

启动录制

使用说明：创建视频录制器，用于现场设备本地录制视频流。

```
/**
* @name TRRO_startRecorder(Experimental)
* @brief 启动录制
* @param[in]   recorderID          录制ID (若想录制SDK采集流，id需设置成对应stream，
config文件中stream_config需加上 “record_on”:1)
* @param[in]   format              录制格式：0 ： 264
* @param[in]   width               编码宽
* @param[in]   heigh               编码高
* @param[in]   jump                录制跳帧数（隔几帧录一帧）
* @param[in]   fps                 编码帧率
* @param[in]   bps                 码率
* @param[in]   filename            文件名
* @param[in]   config              录制选项（保留）
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_startRecorder(int recorderID, int format, int width,
int heigh, int jump, int fps, int bps,
                        const char* filename,  const char* config);
```

参数	含义
recorderID	录制 ID (若想录制SDK采集的视频流，ID 需设置成对应视频流ID， 并在对应视频流的stream_config配置中加上 “record_on” :1，对应视频流数据会自动导入匹配ID的recorder)
format	录制格式：0 ： 264
width	编码宽
heigh	编码高
jump	录制跳帧数（隔几帧录一帧）
fps	编码帧率
bps	码率
filename	文件名
config	录制选项（保留）

发送录制数据

使用说明：输入要录制的视频数据。录制外部输入视频流时，配合 TRRO_startRecorder 接口和 TRRO_stopRecorder 接口使用。

```
/**
* @name TRRO_sendRecordVideoData(Experimental)
* @brief 发送录制数据
* @param[in]   recorderID          录制ID
* @param[in]   data                录制数据
* @param[in]   width               数据源宽
* @param[in]   height              数据源高
* @param[in]   size                录制数据大小
* @param[in]   format              视频源格式 1 yuv420 , 4 YUYV
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_sendRecordVideoData(int recorderID, const char*
data, int width, int height, int format);
```

参数	含义
recorderID	录制 ID

data	录制数据
width	编码宽
heigh	编码高
size	录制数据大小
format	编码帧率
bps	视频源格式 1 yuv420 , 4 YUYV

切换录制文件

使用说明：切换录制文件，配合录制功能使用。

```
/**
 * @name    TRRO_switchRecorderFile(Experimental)
 * @brief   切换录制文件
 * @param[in]    recorderID     录制ID
 * @param[in]    filename       文件名
 * @return   1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_switchRecorderFile(int recorderID, const char*
filename);
```

参数	含义
recorderID	录制 ID
filename	切换后的录制文件名

结束录制

使用说明：停止录制，配合录制功能使用。

```
/**
* @name   Trro_TRRO_OnStatestopRecorder(Experimental)
* @brief  停止录制
* @param[in]   recorderID      录制ID
* @return  1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_stopRecorder(int recorderID);
```

参数	含义
recorderID	录制 ID

视频采集接口

开始采集

使用说明：用于采集相机数据并通过回调函数将相机数据回调给用户。

```
/**
 * @name TRRO_onVideoCaptureFrame
 * @brief 外部调用 TRRO_startVideoCapture 时的数据回调
 * @param[in] context      上下文
 * @param[in] capture_id   采集视频的当前源 ID
 * @param[in] data         采集的视频数据
 * @param[in] length       采集的视频数据长度
 * @param[in] width        采集的视频宽度
 * @param[in] height       采集的视频高度
 * @param[in] color_format 采集的视频格式
 * @return void
 */
typedef void TRRO_onVideoCaptureFrame(void *context, unsigned long long
capture_id, const char* data, int length, int width, int height, TrroColor color_format);


/**
* @name  TRRO_startVideoCapture
* @brief 开始摄像头采集
* @param[in] context   上下文
* @param[in] url       采集路径，摄像头为 /dev/video*, rtsp 时是 rtsp 的地址
* @param[in] protocol  kRTSP 为网络摄像头 rtsp协议采集，kV4L2_MMAP 为 USB/GMSL 等
摄像头 V4L2 协议采集，kV4L2_DMA用于nvidia jetson平台V4L2采集
* @param[in] TrroColor 采集视频格式，V4L2 协议为摄像头图像格式，RTSP 协议采集为目标回
调格式
* @param[in] width     采集视频宽度
* @param[in] height    采集视频高度
* @param[in] fps       采集视频帧率
* @param[in] callback  采集视频回
* @param[in, out] capture_id 采集标识符，用于区分不同的采集，传入为指针形式，支持值指
定或者自动生成(当(*capture_id) > 0时使用指定传入值，为0使用sdk自动生成值)
* @return 开始采集失败则返回错误码，返回1表示成功ocol, TrroColor color_format, int
width, int height, int fps, TRRO_onVideoCaptureFrame callback, unsigned long long*
capture_id);
```

参数	含义
context	上下文
url	采集路径，摄像头为 /dev/video*, rtsp 时是 rtsp 的地址
protocol	kRTSP 为网络摄像头 rtsp 协议采集，kV4L2_MMAP 为 USB/GMSL 等摄像头 V4L2 协议采集，kV4L2_DMA 用于 nvidia jetson 平台 V4L2 采集
TrroColor	采集视频格式，V4L2 协议为摄像头图像格式，RTSP 协议采集为目标回调格式
width	采集视频宽度
height	采集视频高度
fps	采集视频帧率
callback	采集视频回调
capture_id	采集标识符，用于区分不同的采集，传入为指针形式，支持值指定或者自动生成（当（*capture_id）> 0时使用指定传入值，为0使用 sdk 自动生成值）

停止采集

使用说明：用于采集相机数据并通过回调函数将相机数据回调给用户。

```
/**
* @name  TRRO_stopVideoCapture
* @brief 停止摄像头采集
* @param[in] capture_id 采集视频源 ID(不能为0)
* @return 停止采集失败则返回错误码，返回1表示成功
*/
extern "C" TRRO_EXPORT int TRRO_stopVideoCapture(unsigned long long capture_id);
```

参数	含义
capture_id	采集视频源 ID（不能为0）

现场设备配置说明

最近更新时间：2024-02-20 10:59:32

JSON 配置文件样例

文件名：config.json
文件位置：$(workspace)/config.json
文件类型：json
注意："//"后注释在使用时要删除。最小节点如下

```
{
 "device_id":"dev1", //修改为控制台中创建的现场设备ID
 "device_name":"vin234",
 "device_streams":1, //如果是多路输入,修改这里的流数目，并增加streams_config数组中的元素个数
 "cloud_mode":"public",
 "certificate":"./device.pem",
 "projectid" : "xxxxx", //修改为控制台中创建的项目ID
 "password": "xxxxx", //修改为控制台中创建的密码
 "streams_config": [
 {
    "fps":30,
    "bps":2000,
    "width":1920,
    "height":1080,
    "camera":0, //修改为实际接入的相机，对应/dev/videox
    "protocol":"v4l2" //如果为外部输入yuv数据，修改为outside。 如果为外部输入编码流数据，修改为outenc
 }
 ]
}
```

JSON 配置根节点含义

节点名	类型	含义	缺省值	备注	参数范围
device_id	字符串型（String）	设备 ID，用于注册到服务器的唯一编码	-	必填	-
device_name	字符串型（String）	设备名称，用于辨识设	-	必填	-

		备信息			
force_login	数字型（Number）	是否强制登录	0	同名 ID 登录是否剔除之前的设备登录	• 0：已有相同 ID 登录时，当前设备登录被剔除 • 1：已有相同 ID 登录时，当前设备会强制登录，将之前设备剔除
server_ip	字符串型（String）	信令服务端 IP	–	公有云账号无需指定，私有化必填	–
server_port	数字型（Number）	信令服务端端口号	–	公有云账号无需指定，私有化必填	–
rtc_server_ip	字符串型（String）	媒体服务端 IP	–	公有云账号无需指定，私有化必填	–
rtc_server_port	数字型（Number）	媒体服务端端口号	–	公有云账号无需指定，私有化必填	–
projectid	字符串型（String）	项目 ID	–	公有云必填	–
cloud_mode	字符串型（String）	云模式	"private"	可选	• public：公有云 • private：私有云
password	字符串型（String）	设备密钥	–	公有云必填	–
certificate	字符串型（String）	服务器公钥证书	–	公有云必填，device.pem 文件路径	–
log_enable	布尔型（Boolean）	日志开关	0	可选	• 0：不使能 • 1：info 日志 • 2：debug + info 日志
sdk_mod	字符串型	流传输模式	"ser	可选	–

e	（String）		ver"		
min_port	数字型（Number）	本地传输端口最小值	50000	可选	-
max_port	数字型（Number）	本地传输端口最大值	50100	可选	-
audio_enable	数字型（Number）	是否采集音频	0	可选	• 0：不采集 • 1：采集
audio_receive	数字型（Number）	是否接收音频	1	可选	• 0：不接收 • 1：接收
device_streams	数字型（Number）	设备视频流数目	-	必填	1-8
streams_config	JSON 数组（JSON Array）	流配置数组	-	必选	数量与 device_streams 一致
network_bind	字符串数组（String Array）	指定网卡ip	-	可选	有指定网卡或多网需求时设置，`"network_bind":["ip1","ip2"]`，配置指定通信网卡的本地 IP
output_path	字符串型（String）	文件存储路径	"./"	可选	配置 sdk 运行过程中日志和临时文件的输出路径
log_file_num	数字型（Number）	日志文件存储数目	7	可选	-
use_local_conf	数字型（Number）	是否使用本地配置	0	可选（仅限私有化）	• 0：使用远端配置 • 1：使用本地配置

use_local_license	数字型（Number）	是否使用本地license	0	可选（仅限私有化）	• 0：使用远端 license • 1：使用本地 license
fec_enable	数字型（Number）	开启 FEC	0	可选	• 0：关闭本地 FEC • 1：开启本地 FEC
up_fec_rate	数字型（Number）	指定上行 fec 冗余度	0	可选	• 0：自动冗余度 • >0: 指定冗余度%，最大100
down_fec_rate	数字型（Number）	指定下行 fec 冗余度	0	可选	• 0：自动冗余度 • >0: 指定冗余度%，最大100

streams_config 流配置含义

节点名	类型	含义	默认值	备注	参数范围
fps	数字型（Number）	视频帧率	–	必选	–
bps	数字型（Number）	流传输码率	–	必选，根据画质需求及实际网络负载配置。码率越高，画质越清晰，传输数据量越大	–
min_bps	数字型（Number）	期望最低传输码率	默认值与 bps 相同	可选，根据最低画质需求填写。	–
width	数字型（Number）	视频源宽度	–	必选，相机采集为相机分辨率，外部输入视频时，根据输入分辨率填写	–
height	数字型（Number）	视频源高度	–	必选，相机采集为相机分辨率，外部输入视频	–

				时，根据输入分辨率填写	
encode_width	**数字型（Number）**	编码传输的视频宽度	默认与width一致	可选	-
encode_height	**数字型（Number）**	编码传输的视频高度	默认与height一致	可选	-
min_width	**字符串型（String）**	最小调整编码分辨率宽	与encode_width一致	可选	开启动态分辨率时设置
min_fps	**数字型（Number）**	最低 fps	20	可选	调整动态帧率时设置
forece_min	数字型（Number）	是否开启强制最低码率	0	可选	• 0：动态控制最低码率 • 1：将期望最低码率作为最低码率
major	**数字型（Number）**	是否配置为主流	0	可选	• 0：否 • 1：是，高优先保障主流
protocol	**字符串型（String）**	采集协议	“v4l2”	可选	• “v4l2”：由SDK采集 v4l2相机数据 • “outenc”：客户通过接口输入编码流数据 • "outside"：客户通过接口输入图像数据
camera	**数字型**	v4l2相机编号	0	填写数字X对应	-

	（Number）			设备/dev/videoX，当protocol 字段配置为“v4l2”时，此字段生效	
codec	数字型（Number）	编码类型	0	可选	• 0：h264 • 1：h265 • 2：av1
encoder_type	数字型（Number）	视频编码方式	1	可选	• 0：软编优先 • 1：硬编优先
frame_type	数字型（Number）	yuyv-uyvy图像反转	0	可选	• 0：不反转 • 1：反转
record_on	数字型（Number）	是否开启本地存储	0	可选	• 0：不开启 • 1：开启
capture_acc	数字型（Number）	是否开启采集加速	0	可选	• 0：不开启 • 1：开启
need_mirror	数字型（Number）	是否开启镜像	0	可选	• 0：不开启 • 1：开启
always_push	数字型（Number）	是否开启强制推流	0	可选	• 0：不开启 • 1：开启

streams_config 流配置 protocol 扩展及网络相机采集配置

1. 拉取 rtsp 相机场景参见 streams_config 节点配置

参考配置：

```
"streams_config": [
    {
      "fps":25,
      "bps":2000,
      "width":1920,
      "height":1080,
```

```
        "protocol":"normal",
        "cameras": [
          {
            "width":1920,
            "height":1080,
            "protocol":1,
            "url":"rtsp://xxxxx"
          }
        ]
      }
  ]
```

2. 多画面拼接场景参考 streams_config 节点配置

```
"streams_config": [
      {
      "fps":25,
      "bps":2000,
      "width":2560,
      "height":1440,
      "protocol":"combine",
      "cameras": [
        {
          "width":640,
          "height":480,
          "camera":1,
          "protocol":2,
          "url":"/dev/video0"
        },
        {
          "width":640,
          "height":480,
          "camera":1,
          "protocol":2,
          "url":"/dev/video1"
        },
        {
          "width":640,
          "height":480,
          "camera":1,
          "protocol":2,
          "url":"/dev/video2"
        },
        {
          "width":640,
```

```
            "height":480,
            "camera":1,
            "protocol":2,
            "url":"/dev/video3"
        }
    ],
    "combine_config": {
        "basic_width":1920,
        "basic_height":1080,
        "pattern": [
            {
                "x1":0,
                "y1":0,
                "x2":640,
                "y2":480
            },
            {
                "x1":640,
                "y1":0,
                "x2":1280,
                "y2":480
            },
            {
                "x1":0,
                "y1":480,
                "x2":640,
                "y2":960
            },
            {
                "x1":640,
                "y1":480,
                "x2":1280,
                "y2":960
            }
        ]
    }
    }
]
```

节点名	类型	含义	默认值	备注	参数范围
protocol	String	图像处理类型	"normal"	默认不处理	• normal：解码后正常传输 • rtsp_enc：透传编码流

					• combine：混流拼接
cameras	JSON Array	视频源相机	–	涉及多个相机组合时，需配置多个相机节点	–
width (cameras 节点中)	Number	相机采集宽度	–	–	–
height (cameras 节点中)	Number	相机采集高度	–	–	–
protocol (cameras 节点中)	Number	采集协议类型	1	–	• 1：rtsp 采集 • 2：v4l2 mmap 采集 • 3：v4l2 dma 采集
url (cameras 节点中)	String	采集数据源地址	–	配合相机采集协议类型	• protocol=1：网络相机拉流 rtsp URL • protocol=2/3：/dev/videox

错误码及排查

最近更新时间：2024-02-07 10:08:21

错误码的定义位于 trro_field.h 头文件中。函数返回值为错误码的负值，如错误码为16777215，返回值为-16777215。错误码具体含义及排查方法如下表所示：

模块	错误码定义	错误码16进制	错误码10进制	排查方向
通用模块	TRRO_SUCCED	0x00000001	1	成功执行
	TRRO_COMMON_ERROR	0x00FFFFFF	16777215	通用失败
配置文件加载模块（排查配置文件内容）	TRRO_CONFIG_ERROR	0x01FFFFFF	33554431	配置模块通用失败
	TRRO_CONFIG_PARSE_FAILED	0x01000002	16777218	关节节点解析异常，检查配置文件中最小启动节点（快速入门）是否完整
	TRRO_CONFIG_ILLEGAL	0x01000003	16777219	检查配置文件中，width、height、protocol 与接口输入参数是否匹配
	TRRO_CONFIG_UNEXIST	0x01000004	16777220	配置文件不存在或长度为0
	TRRO_CONFIG_CER_FAILED	0x01000005	16777221	使用公有云模式（cloud_mode 节点设置 public）启动，但没有证书文件 device.pem 或证书文件校验失败
	TRRO_CONFIG_LIC_FAILED	0x01000006	16777222	使用私有模式启动（cloud_mode 节点设置 private）启动，但 license.txt 文件不存在
	TRRO_CONFIG_STREAMS_SIZE_ERROR	0x01000007	16777223	超过最大支持流数目
	TRRO_CONIFG_	0x0100000	167772	端口范围配置非法

	PORT_RANGE_ILLEGAL	8	24	
	TRRO_CONFIG_LOG_PERMISSON_DENIED	0x01000009	16777225	打开 log 文件权限不足
初始化模块（排查接口调用顺序及配置文件内容）	TRRO_INIT_ERROR	0x02FFFFFF	50331647	初始化失败
	TRRO_INIT_INPUT_ILLEGAL	0x02000002	33554434	初始化失败或配置的设备 ID、端口范围非法
	TRRO_INIT_PARSE_FAILED	0x02000003	33554435	初始化解析配置节点异常
	TRRO_INIT_CREAT_MEDIAMODE_FAILED	0x02000004	33554436	媒体模块创建失败
	TRRO_INIT_PRM_ERROR	0x02000005	33554437	参数错误
	TRRO_INIT_INVALUDE_INPUT	0x02000006	33554438	初始化参数非法
	TRRO_INIT_REPEAT	0x02000007	33554439	重复初始化
排查控制台设备授权是否正常（设备列表）	TRRO_INIT_LICENSE_CHECK_FAILED	0x02000010	33554448	LICENSE 校验失败
	TRRO_INIT_LICENSE_FILE_ERROR	0x02000011	33554449	LICENSE 文件错误
	TRRO_INIT_LICENSE_CHECK_TIME_FAILED	0x02000012	33554450	授权过期
	TRRO_INIT_LICENSE_CHECK_DEVICE_FAILED	0x02000013	33554451	硬件验证失败
	TRRO_INIT_LICENSE_CHECK_STREM_FAILED	0x02000014	33554452	授权流数目小于设备配置流数目

	TRRO_INIT_LICENSE_CHECK_ID_FAILED	0x02000015	33554453	设备 ID 验证失败
	TRRO_INIT_PUBLIC_LICENSE_CHECK_TIMEOUT	0x02000100	33554688	公有云 LICENSE 验证超时
	TRRO_INIT_PUBLIC_LICENSE_CHECK_NOT_BIND	0x02000101	33554689	公有云设备未绑定 LICENSE
	TRRO_INIT_PUBLIC_LICENSE_CHECK_NOT_ENOUGH	0x02000102	33554690	公有云 LICENSE 数量不足
	TRRO_INIT_PUBLIC_LICENSE_CHECK_OVERTIME	0x02000103	33554691	公有云 LICENSE 已过期
信令服务器模块（排查网络及服务器状态）	TRRO_SIGNAL_ERROR	0x03FFFFFF	67108863	信令模块异常
	TRRO_SIGNAL_REGIST_FAILED	0x03000002	50331650	注册 MQTT 信令服务器失败
	TRRO_SIGNAL_STATUS_ABNORMAL	0x03000003	50331651	MQTT 信令服务连接异常
	TRRO_SIGNAL_MESSAGE_FAILED	0x03000004	50331652	信令消息处理失败
	TRRO_SIGNAL_CONNECT_OUTTIME	0x03000005	50331653	服务器连接超时，如果设置为异步启动可忽略此错误码
	TRRO_SIGNAL_DEVICEID_OR_PASSWORD_INCORRECT	0x03000006	50331654	信令服务器连接用户名或者密码错误
流采集模	TRRO_CAPTURE	0x04FFFFF	838860	采集模块异常

块（检查相机的接入、分辨率配置是否正常）	_ERROR	F	79	
	TRRO_CAPTURE_OPENDEVICE_FAILED	0x04000002	67108866	打开设备失败，检查相机是否有其他程序占用
	TRRO_CAPTURE_GETSOURCE_FAILED	0x04000003	67108867	读取相机数据超时失败
	TRRO_CAPTURE_UNKNOWN_CAPTURETYPE	0x04000004	67108868	未知的相机数据源类型
流传输模块	TRRO_CONNECT_ERROR	0x05FFFFFF	100663295	连接异常
	TRRO_CONNECT_OUTTIME	0x05000002	83886082	超时断连
消息传输模块	TRRO_MESSAGE_ERROR	0x06FFFFFF	117440511	消息发送失败
	TRRO_MESSAGE_CHANNEL	0x06000002	100663298	无可用消息发送通道
	TRRO_MESSAGE_BYTE_EXCEED	0x06000003	100663299	发送字节数超出限制
存储模块	TRRO_STOR_ERROR	0x07FFFFFF	134217727	存储异常，检查接口调用顺序
	TRRO_STOR_UNENABLE	0x07000001	117440513	存储功能未使能
	TRRO_STOR_IDEXIST	0x07000002	117440514	重复开启存储
	TRRO_STOR_ID_ILLEGAL	0x07000003	117440515	存储 ID 非法
	TRRO_STOR_PARAM_ILLEGAL	0x07000004	117440516	存储数据源输入入参非法
	TRRO_STOR_UNSET_FILENAME	0x07000005	117440517	存储文件名为空
外采模块	TRRO_EXTERNA	0x08000000	134217	外采模式错误

	L_RESIZE	1	729	
扩展模块	TRRO_ERROR_CALLBACK_CAMERA	0x09010000	151060480	相机异步回调错误
	TRRO_ERROR_CALLBACK_MIC	0x09020000	151126016	麦克风异步回调错误
	TRRO_ERROR_CALLBACK_BANDWIDTH_LIMIT	0x09030000	151191552	评估网络带宽无法满足最低码率需求
保留	TRRO_UNSUPPORT	0x0F000001	251658241	该调用或函数功能暂不支持