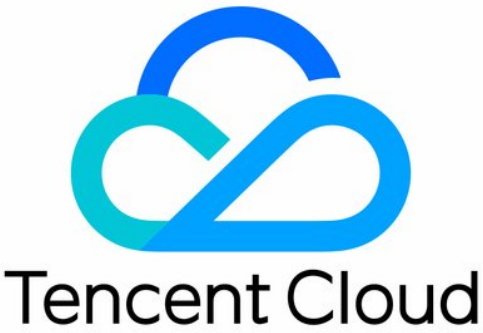
Real-time Teleoperation

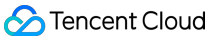
Field Device SDK

Product Documentation



Copyright Notice

Trademark Notice



Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

Contents

Field Device SDK
Basic Introduction

Last updated：2025-02-11 16:48:03

Scope of application

This SDK is used for field device access, suitable for Linux 64-bit systems, and provides access in the form of so libraries.

Main Feature

This SDK offers low-latency audio and video calls and control/status data transmission for remote real-time control scenarios. The main features are as follows:

Video Processing

Features include video collection, scaling, tailoring, concatenation, etc.

Video codec

Supports hardware encoding types including H264, H265, and AV1 on the Jetson platform, and H264 software encoding on general x86_64 and aarch64 platforms.

Video Transmission

Supports the transmission of encoded video streams such as H264/H265/AV1, with weak network resilience and low-delay transmission capabilities.

Binary Data Transmission

Supports pass-through for binary data, allowing the transmission of binary data to the call participants.

Self-managed streaming

The SDK can self-manage the streaming status and push the stream when there are viewers at the pull stream end, reducing unnecessary network bandwidth usage.

Self-reconnect upon network disconnection

The SDK can self-manage network disconnection status, automatically handling network disconnection exceptions and attempting status recovery.

Concepts

Device ID

Used to identify field devices with uniqueness. If there are devices with the same device ID connected to the network, the device will be kicked off accroding to the "force_login" configuration. If force_login is 1, then the previous device will be kicked off; otherwise the latter device will be kicked off.

Video Stream ID

Used to identify the video stream number of field devices, the number starts from 0 and increments. It is consistent with the array element numbering in the Json configuration's streams_config. When a remote device pulls a video stream, it will specify the video stream to be pulled through the device ID and video stream ID.

Configuration file

the file containing the SDK's initial configuration json strings, defaultly named “config.json”.

SDK Call Procedure

Last updated：2025-02-11 16:48:24

Recommended Steps

> **Note:**
> The signalState signaling callback registration needs to be invoked before the initialization function, other callback functions can be registered after initialization.

Step 1: Initialization

Call the Init interface to initialize the SDK with loading a JSON file or inputting a JSON string. . Before initialization, the signalState signaling callback interface should be registered first.

Step 2: Register the necessary callback functions

Register the required callback functions as needed, such as control message callback, log callback, connection status callback, video link information callback, latency information callback, to handle corresponding status information and events.

Step 3: Start SDK to enter video managed state

Invoke TRRO_start to enter the managed state. At this point, the SDK will listen to various requests, automatically carry out video capture, video streaming, and handle abnormal events such as network disconnection.

Step 4: Call the data send function as needed

When there is a need to input external video data or binary data, call the corresponding sending interface to input the video data or binary control/status data at the required sending time.

Sample Code

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

// External input video data
```

```
void externalVideoInput(int stream_id) {
    int data_width = 1280;
    int data_height = 720;
    char *data = (char*)malloc(data_width*data_height*3/2);
     // YUV420 video data, using all-zero simulated data (Green Map) in
the sample for illustration
    memset(data, 0, data_size);
    while(true) {
        TRRO_externalVideoData(stream_id, data, data_width, data_height,
0);
        usleep(33 * 1000);
    }
    free(data);
}

// Main thread
int main() {
    // Register signal callback
    TRRO_registerSignalStateCallback(nullptr, [](void *context,
SignalState state) {
        if(state == kTrroReady) {
            // Server connection successful, register callback function,
and start SDK management
            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();
            // After init successfully connects to the service, TRRO SDK
management can be started
            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 {
                // Pay attention to configuration issues
                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");
        }
    });
```

```
    // It is recommended to use -1 for the last parameter to start in
non-blocking mode and wait for the signaling connection success callback
    // You can replace it with other initialization functions as needed,
such as initializing with a configuration string
    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);
        }
    }

    // Start an external input video data thread, input video data to
stream 0, stream 0 protocol must be 'outside'
    // std::thread t1(externalVideoInput, 0);
    // t1.detach();

    // Prevent the program from exiting
    while(true){
        sleep(30000);
    }
    return 0;
}
```

C Field Device SDK API

Last updated：2025-02-11 16:48:53

API Description

Applicable to Linux 64-bit systems. Include the header file trro_field.h, and refer to the comments in the header file for API details. Below are descriptions of commonly used APIs.

> **Note:**
> APIs marked as atomic capability APIs are functional interfaces not associated with TRRO real-time interaction services. These APIs can be freely used after SDK initialization, allowing customers to reuse SDK capabilities to develop other features.

API Overview

Initialization API

Interfaces	Description
TRRO_initGwJson	Initializing with a JSON String
TRRO_initGwPath	Initializing with a Configuration File Path
TRRO_initGwPathWithLicense	Initializing with a Configuration Path and License Path (Private)
TRRO_initGwJsonWithLicense	Initializing with a JSON String and License Path (Private)

Status Retrieval APIs

Interfaces	Description
TRRO_registerSignalStateCallback	Registration Signaling Status Callback
TRRO_registerOnState	Registration of video connection status callback
TRRO_registerLatencyCallback	Registration of latency status callback
TRRO_registerMediaState	Registration of video media and network status callback
TRRO_registerAudioMediaState	Registration of audio media and network status callback

TRRO_registerEncodeFrameInfoCallback	Registration of external encoding suggestion callback (for external encoding)

Start/Stop APIs

Interfaces	Description
TRRO_start	Starting the SDK
TRRO_stop	Stopping the SDK

Audio and Video Input/Output APIs

Interfaces	Description
TRRO_externalVideoData	Inputting External Source Video Stream (Image Data)
TRRO_registerVideoCaptureCallback	Register Video Capture Data Callback Function
TRRO_externalEncodeVideoData	Inputting External Source Video Stream (Encoded Data)
TRRO_externalAudioData	Inputting External Audio Stream (PCM Data)

Camera Capture API (Atomic Capability API)

Interfaces	Description
TRRO_startVideoCapture	Start Camera Capture
TRRO_stopVideoCapture	Stop camera capture

Message APIs

Interfaces	Description
TRRO_sendControlData	Send data to remote devices
TRRO_registerControlDataCallback	Register callback for remote device control messages

Permission Control APIs

Interfaces	Description
TRRO_registerOperationPermissionRequest	Register callback for remote device permission request notification
TRRO_setOperationPermission	Set remote device operation permissions

Reconfiguration APIs

Interfaces	Description
TRRO_reinitRtc	Reconfigure field device video stream, only supports reconfiguration of extended capture protocols such as normal/combine/rtsp_enc

Audio Control APIs

Interfaces	Description
TRRO_audioMute	Mute Settings
TRRO_getDeviceCount	Retrieve the Number of Audio Capture/Playback Devices
TRRO_getDeviceName	Retrieve the Name of Audio Capture/Playback Devices
TRRO_setCurrentDevice	Change the Current Audio Capture/Playback Device

Log/Error Message APIs

Interfaces	Description
TRRO_registerLogCallback	Register Log Callback Function
TRRO_registerOnErrorEvent	Register Error Event Callback
getErrorMsg	API for Retrieving Error Information Based on Error Code

Video Recording API (Atomic Capability API)

Interfaces	Description
TRRO_startRecorder	Start Recording

TRRO_sendRecordVideoData	Sending Recording Data
TRRO_switchRecorderFile	Switching Recording File
TRRO_stopRecorder	End recording

Network Evaluation API

Interfaces	Description
TRRO_testNetworkQuality	Testing if the network quality meets the requirements for video transmission, can be used to check the network at startup

Getting Software Information

Interfaces	Description
TRRO_getSdkVersion	Get the software version number

Initialization API

Use Instructions: You can choose the corresponding initialization API based on whether to use a local license (no local license is needed for public cloud) and the input configuration type (file/string). Only one is needed.

Initializing with a JSON String

Use Instructions: Users must ensure the JSON string format is correct. This API is used for SDK configuration loading and initialization.

```
/*
* @name : TRRO_initGwJson
* @brief : Initializing with a String
* @input : json_str        Configuration file JSON string
* @input : mode            0 synchronous mode, always waiting
*                          -1 asynchronous mode, notify
TRRO_onSignalState after successful initialization
* @return : 1 for success, other failed
*/
int TRRO_initGwJson(const char * json_str, int mode = 0);
```

Parameters	Meaning
json_str	Configuration information JSON string
mode	Return mode • 0: Synchronous blocking mode • 1: Asynchronous mode

Initializing with a Configuration File Path

Use Instructions: Users must ensure the JSON configuration file format is correct. This API is used for SDK configuration loading and initialization.

```
/*
* @name : TRRO_initGwPath
* @brief : Initializing with a configuration file
* @input : cfg_path        JSON configuration file path, e.g.
"./config.json"
* @input : mode            0 synchronous mode, always waiting
*                          -1 asynchronous mode, TRRO_onSignalState
information after successful initialization
* @return : 1 for success, other failed
*/
int TRRO_initGwPath(const char * cfg_path, int mode = 0);
```

Parameters	Meaning
cfg_path	Configuration file path
mode	Return mode • 0: Synchronous blocking mode • 1: Asynchronous mode

Initializing with a Configuration File Path and License File Path

Use Instructions: Users must ensure the JSON configuration file format is correct and the license file path is valid. This API is used for SDK configuration loading and initialization.

```
/*
* @name : TRRO_initGwPathWithLicense
* @brief : Initializing with a Configuration File and Local License
```

```
* @input : cfg_path        JSON configuration file path, e.g.
"./config.json"
* @input : license_path    license file path, e.g. "./license.txt"
* @input : mode            0 synchronous mode, always waiting
*                          -1 asynchronous mode, notify
TRRO_onSignalState after successful initialization
* @return : 1 for success, other failed
*/
int TRRO_initGwPathWithLicense(const char * cfg_path, const char *
license_path, int mode = 0);
```

Parameters	Meaning
cfg_path	Configuration file name
license_path	Local license file path
mode	Return mode • 0: Synchronous blocking mode • 1: Asynchronous mode

Initializing with a Configuration String and License File Path

Use Instructions: Users must ensure the JSON string and license content formats are correct. This API is used for SDK configuration loading and initialization.

```
/*
* @name : TRRO_initGwJsonWithLicense
* @brief : Initializing with a String and Local License
* @input : json_str        Configuration file JSON string
* @input : license_path    license file path, e.g. "./license.txt"
* @input : mode            initialization mode
*                           0 synchronous mode, always waiting
*                           -1 asynchronous mode, notify
TRRO_onSignalState after successful initialization
* @return : 1 for success, other failed
*/
int TRRO_initGwJsonWithLicense(const char * json_str, const char *
license_path, int mode = 0);
```

Parameters	Meaning
json_str	Configuration file JSON string
license_path	license path
mode	Return mode • 0: Synchronous blocking mode • 1: Asynchronous mode

Status Retrieval APIs

Usage Instructions: Register the required callback APIs as needed.

Registering Signaling Status Callback (Asynchronous Initialization Callback)

Usage Instructions: This API is used to register the signaling status callback API, which is required for asynchronous initialization. The initialization success is determined by the Ready status.

```
/*
* @name : TRRO_registerSignalStateCallback
* @brief : Register signaling service connection status callback
* @input : context              Context pointer
           callback              Callback function
* @return : 1 for success, other failed
*/
int TRRO_registerSignalStateCallback(void *context, TRRO_onSignalState
*callback);
```

Parameters	Meaning
context	target remote device id
callback	the permission to be set

```
Callback Definition
enum SignalState {
    kTrroReady = 0, /**< Connection established successfully */
    kTrroLost = 1, /**< Connection lost, automatic reconnection will be
attempted internally */
    kTrroReup = 2, /**< Automatic reconnection successful */
```

```
    kTrroKickout = 3,
    kTrroAuthFailed = 4, /**< Incorrect username or password */
};

/*
* @name : TRRO_onSignalState
* @brief : Signaling connection status callback
* @input : context        Context pointer
*          state          kTrroReady              Connection
established successfully
*                         kTrroLost               Connection lost,
automatic reconnection will be attempted internally
*                         kTrroReup               Automatic
reconnection successful
* @return : void
*/
typedef void TRRO_onSignalState(void *context, SignalState state);
```

Parameters	Meaning
context	Callback context pointer, returns the context passed during registration
state	Signaling connection status

Registration of video connection status callback

Usage Instructions: This API is used to register video stream connection status callback. You can choose to call this API as needed.

```
/**
* @name TRRO_registerOnState
* @brief Register video connection status callback function
* @param[in]   context             Context pointer
* @param[in]   callback            Callback function
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerOnState(void* context,
TRRO_OnState * callback);
```

Parameters	Meaning

context	Context pointer, returned during callback for positioning
callback	the permission to be set

```
Callback Function Definition
enum TrroState {
    kDisconnect = 0, /**< Disconnected */
    kConnecting = 1, /**< Connecting */
    kConnected = 2,  /**< Connected */
    kDisconnecting = 3, /**< Disconnecting */
};

/**
* @name TRRO_onState
* @brief Video connection status callback
* @param[in]     context        Context pointer
* @param[in]     stream_id      video stream id
* @param[in]     state          TrroState connection status
* @return void
*/
typedef void TRRO_OnState(void* context, int stream_id, int state);
```

Registering Latency Callback

Use instructions: This API is used to obtain the video data processing delay from push to pull on the field device side, and can be used as needed.

```
/**
* @name TRRO_registerLatencyCallback
* @brief Register latency callback function
* @param[in]   context             Context pointer
* @param[in]   callback            Callback function
* @return  1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerLatencyCallback(void *context,
TRRO_onLatencyReport *callback);
```

Parameters	Meaning

context	target remote device id
callback	the permission to be set

```
/**
* @name TRRO_onLatencyReport
* @brief Delayed message callback
* @param[in]     context        Context pointer
* @param[in]     stream_id      Stream ID
* @param[in]     vcct           Video control closed-loop delay, equal
to video uplink delay (excluding capture) + control downlink delay
* @return void
*/
typedef void TRRO_onLatencyReport(void *context, int stream_id, int
vcct);
```

Parameters	Meaning
context	target remote device id
stream_id	Stream ID
vcct	Video control closed-loop delay, equal to video uplink delay (excluding capture) + control downlink delay

Registering the Video Media Status Callback

Use instructions: This API is used to obtain the media status of the connected video stream on the field device, and can be used as needed.

```
/**
* @name  TRRO_registerMediaState
* @brief Register network status
* @param[in]   context             Context pointer
* @param[in]   callback            Callback function
* @return  1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerMediaState(void* context,
TRRO_onMediaState * callback);
```


Parameters	Meaning
context	target remote device id
callback	Media status callback function

```
/**
* @name TRRO_onMediaState
* @brief Media transmission status callback
* @param[in]    context      Callback context pointer
* @param[in]    stream_id    Stream ID
* @param[in]   fps          Frames per second
* @param[in]   bps          Bits per second
* @param[in]   rtt          Round-trip time
* @param[in]   lost         Total packet loss
* @param[in]   packets_send Total number of packets sent
* @param[in]   stun         NAT traversal mode 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);
```

Parameters	Meaning
context	target remote device id
stream_id	Stream ID
fps	Frames per second
bps	Data per second
rtt	Packet round-trip time
lost	Total packet loss
packets_send	Total sent count
stun	NAT Traversal Mode • 0：host • 1：srflx • 2：prflx

	• 3：relay

Registering Audio Status Callback

Use instructions: This API is used to obtain the media status of the connected audio stream on the field device, and can be used as needed.

```
/**
* @name  TRRO_registerAudioMediaState
* @brief Registering Audio Status Callback
* @param[in]   context              Context pointer
* @param[in]   callback             Callback function
* @return  1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerAudioMediaState(void* context,
TRRO_onMediaState * callback);
```

Parameters	Meaning
context	target remote device id
callback	Media status callback function

Registering Encoding Suggestion Callback

Usage Instructions: This API is used together with the external encoding stream input data API. Users can process the input encoding stream based on the callback information from this API.

```
/**
* @name TRRO_registerEncodeFrameInfoCallback
* @brief Registering Encoding Suggestion Callback Function
* @param[in]    context              Context pointer
* @param[in]    callback             Callback function
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerEncodeFrameInfoCallback(void
*context, TRRO_onEncodeFrameInfo *callback);
```


Parameters	Meaning
context	Context pointer, returned during callback for positioning
callback	the permission to be set

```
Callback Function Definition
/**
* @name TRRO_onEncodeFrameInfo
* @brief Encoding suggestion callback, applicable to external input
encoding frame scenarios
* @param[in]     context        Context pointer
* @param[in]     stream_id      Stream ID
* @param[in]     type           Callback type, 0: Force keyframe
request, 1: Bitrate update request
* @param[in]     bitrate        Effective when type is 1, indicates the
suggested input encoding data bitrate, in kbps
* @return void
*/
typedef void TRRO_onEncodeFrameInfo(void *context, int stream_id, int
type, int bitrate);
```

Parameters	Meaning
context	Callback context pointer, returns the context passed during registration
strem_id	Stream ID
type	Callback Type • 0: Force keyframe request • 1: Bitrate update request
bitrate	Effective when type is 1, indicates the suggested input encoding data bitrate, in kbps

Start/Stop APIs

Start API

Use instructions: This API is used to start the field device SDK. It can be called after successful initialization, either when synchronous mode initialization returns success or when asynchronous mode initialization TRRO_onSignalState notifies the connection is Ready.

```
/**
* @name TRRO_start
* @brief Start the audio and video transmission service, to be called
after successful initialization (synchronous mode init returns success
or asynchronous mode initialization TRRO_onSignalState notifies the
connection is Ready)
* @param[in] void
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_start();
```

Stop API

Use instructions: This API is used to stop the field device SDK.

```
/**
* @name TRRO_stop
* @brief Destroy the SDK and release SDK resources
* @param[in] void
* @return void
*/
extern "C" TRRO_EXPORT void TRRO_stop();
```

Audio and Video Input/Output APIs

Inputting External Source Video Stream (Image Data)

Use instructions: This API is used to provide the raw image data stream of the video input by the user. Note that the input data source must match the number of streams and stream configuration nodes in the configuration file.
The width and height of the input data source must be consistent with the width and height in the Field Device Configuration Instructions, and the value of the "protocol" node in the configuration file should be changed to "outside".

```
/**
* @name TRRO_externalVideoData
* @brief External Image Input API
* @param[in] stream_id Stream ID
```

```
* @param[in] data Message body
* @param[in] width Data source width
* @param[in] height Data source height
* @param[in] type Data source type, currently supports
Trro_ColorYUVI420, Trro_ColorJPEG, Trro_ColorYUYV
* @param[in] dataSize Data size, automatically calculated based on
format if 0, must specify for data like JPEG with no specific size
* @param[in] text String to overlay on image (experimental)
* @param[in] point_x Starting x-coordinate for overlay text, 0 at the
leftmost (experimental)
* @param[in] point_y Starting y-coordinate for overlay text, 0 at the
top (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);
```

Parameters	Meaning
stream_id	Video stream ID, starting from 0, the second stream is 1, and so on
data	Image data
width	Image width
height	Image height
type	Image type • 0：YUVI420 • 4：YUYV • 5: JPEG, YUVI420 format recommended
dataSize	Image data size, 0 for auto calculation, must specify for image formats like JPEG with no fixed size
text	Reserved
point_x	Reserved
point_y	Reserved

Register Video Capture Data Callback Function

Use instructions: This API is used to obtain the camera data source collected by the SDK for upper-level business use.

```
/**
* @name  TRRO_registerVideoCaptureCallback
* @brief Register video capture data callback function
* @param[in]   context              Context pointer
* @param[in]   callback             Callback function
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerVideoCaptureCallback(void*
context, TRRO_onVideoCaptureData * callback);
```

Parameters	Meaning
context	target remote device id
callback	the permission to be set

```
/**
* @name TRRO_onVideoCaptureData
* @brief Video frame capture callback
* @param[in]     context        Context pointer, returns the context
passed in during registration
* @param[in]     data           Video data
* @param[in]     width          Width
* @param[in]     height         Height
* @param[in]     type           Video format, 0 YUV420, 4 YUYV
* @param[in]     stream_id      Stream number
* @return void
*/
typedef void TRRO_onVideoCaptureData(void *context, const char* data,
int width, int height, int type, int stream_id);
```

Parameters	Meaning
remote_devid	Set the target device ID for permissions
data	Video Data

width	Video data width
height	Video data height
type	Video format: • 0 YUV420 • 4 YUYV
stream_id	Stream number

Inputting External Source Video Stream (Encoded Data)

> **Notes:**
> It is recommended to use single slice encoding, and I-frames need to carry SPS/PPS.

Note: This API is used to provide the user-input encoded video stream for transmission. Note that the input data source must match the number of streams and stream configuration nodes in the configuration file. The input video stream encoding type (e.g., H264) must be consistent with the codec of the corresponding stream in the configuration file, and the keyframe must contain SPS/PPS information. Meanwhile, the "protocol" node value in the configuration file should be modified to "outenc".

```
/**
* @name TRRO_externalEncodeVideoData
* @brief External encoded stream input, the encoded stream codec needs
to be consistent with the configuration codec
* @param[in]    stream_id            Stream ID
* @param[in]    data                 Message body
* @param[in]    width                Data source width
* @param[in]    height               Data source height
* @param[in]    size                 Data size
* @param[in]    type                 Data source type, refer to
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);
```

Parameters	Meaning
stream_id	Video stream ID, starting from 0, the second stream is 1, and so on

data	Video stream data
width	Video width
height	Video height
size	Video frame size
type	Encoding type • 0: P frame • 1: I frame (must include SPS/PPS information)

Inputting External Audio Stream (PCM Data)

> **Notes:**
> In the configuration, the "audio_external" node value needs to be modified to 1.

Use instructions: This API is used to provide user input audio stream for transmission, with input as PCM data, 16-bit sampling.

```
/**
* @name  TRRO_externAudioData(Experimental)
* @brief External audio data input (PCM data), 16-bit audio sampling
* @param[in]   data         Source data
* @param[in]   data_size    Data size
* @param[in]   channel      Number of audio channels
* @param[in]   sample_rate  Audio sample rate
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_externAudioData(const char* data, int
data_size, int channel, int sample_rate);
```

Parameters	Meaning
data	Audio stream data
data_size	Data size
channel	Number of channels • 1: Mono

	• 2: Stereo
sample_rate	Sampling frequency, supports 8k/16k/24k/32k/44.1k/48kHz, for example, 48k sampling rate input is 48000

Message APIs

Send data to remote devices

Use Instructions: This API is used to send binary data to remote devices, in conjunction with the remote device SDK data callback API. This API can be called as needed. Currently, the data size limit for a single transmission is 700 bytes, with a frequency limit of 100 times per second.

```
/**
* @name TRRO_sendControlData
* @brief Send data to remote devices
* @param[in] msg Message body
* @param[in] len            Message body length
* @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);
```

Parameters	Meaning
msg	Binary message content
len	Message length
qos	Send qos type • 0: unreliable transmission • 1: Reliable Transmission

Register callback for remote device control messages

This interface is used to register the report data callback function to receive the data sent by the field device.

```
/**
* @name TRRO_registerControlDataCallback
* @brief Register Remote Device Message Callback Function
```

```
* @param[in]   context             Context pointer, returns this
context pointer during callback
* @param[in]   callback            Callback function
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerControlDataCallback(void*
context, TRRO_onControlData * callback);
```

Parameters	Meaning
context	Context pointer, the context indicator will be returned upon callback for localization
callback	the permission to be set

```
Callback Function Definition
/**
* @name TRRO_onControlData
* @brief Receive Remote Device Message Callback
* @param[in]     context        Context pointer, returns the context
passed in during registration
* @param[in]     controller_id  Remote device ID
* @param[in]     msg            Message body
* @param[in]     len            Message body length
* @param[in]     qos            Message qos type 0:unreliable,
1:reliable
* @return void
*/
typedef void TRRO_onControlData(void *context, const char
*controller_id, const char* msg, int len, int qos);
```

Parameters	Meaning
context	Callback context, returns the context passed in during registration
controller_id	Field Device ID
msg	pointer to the data buffer
len	String Length

qos	Message qos type • 0: ultra-low latency transmission, data lost is allowed • 1: Reliable Transmission

Permission Control APIs

Register callback for remote device permission request notification

This interface is used for the field device to receive permission requests from the remote device.

```
/**
* @name  TRRO_registerOperationPermissionRequest
* @brief Register callback for remote device control permission request
notification
* @param[in]   context             Context pointer
* @param[in]   callback            Callback function
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerOperationPermissionRequest(void
*context, TRRO_onOperationPermissionRequest *callback);
```

Parameters	Meaning
context	target remote device id
callback	the permission to be set

```
/**
* @name TRRO_onOperationPermissionRequest
* @brief Notification of remote device control permission request
* @param[in]     remote_devid        Remote device ID requesting
permission
* @param[in]     permission           Requested permission, refer to
TrroPermission, 0: guest (view only), 1: master (full control)
* @return void
*/
typedef void TRRO_onOperationPermissionRequest(void* context, const
char* remote_devid, int permission);
```

Parameters	Meaning

context	target remote device id
remote_devid	Remote device ID requesting permission
permission	Requested permissions, see TrroPermission • 0:guest (view-only permission) • 1:master (full control permission)

Set remote device operation permissions

This interface is used to set the remote device operation permissions on the field device.

```
/**
 * @name  TRRO_setOperationPermission
 * @brief Set remote device control permissions. Currently, only one
remote device can have master permission at a time. If a remote device
already has master permission, calling this API to set master permission
will automatically revoke the previous device's master permission and
set the new device.
 * @param[in]  remote_devid         Remote device ID to set permissions
 * @param[in]  permission           Refer to TrroPermission, 0 guest,
view only, 1 master, full control
 * @return 1 for success, other failed
 */
extern "C" TRRO_EXPORT int TRRO_setOperationPermission(const char*
remote_devid, int permission);
```

Parameters	Meaning
context	target remote device id
callback	the permission to be set

Reconfiguration APIs

Reconfiguring RTC Settings

Use instructions: Reconfiguring video stream settings

```
/**
* @name  TRRO_reinitRtc(Experimental)
```

```
* @brief Reconfiguration, currently only supports resetting extended
capture protocols such as normal/combine/rtsp_enc
* @param[in] config   Reconfigure JSON Data
*/
extern "C" TRRO_EXPORT int TRRO_reinitRtc(const char * config);
```

Parameters	Meaning
config	Reconfigure JSON Data

Audio Control Interface

Setting Mute

Use Instructions: Disable Audio Transmission

```
/**
* @name  TRRO_audioMute(Experimental)
* @brief Mute the audio of the remote deviceID
* @param[in] userid  Pullstream End ID
* @param[in] mute    Whether to mute
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_audioMute(const char* userid, bool
mute);
```

Parameters	Meaning
userid	remote deviceID
mute	Whether to mute

Retrieve the Number of Audio Capture/Playback Devices

Use instructions: Retrieve the number of audio capture/playback devices, which can be used to iterate over audio device names.

```
/**
 * MediaDeviceType device type enumeration
 */
 enum MediaDeviceType {
```

```
    ///Microphone type device
    MediaDeviceTypeMic = 0,
    ///Speaker type device
    MediaDeviceTypeSpeaker = 1,
};

/**
* @name  TRRO_getDeviceCount(Experimental)
* @brief Query the number of audio devices based on MediaDeviceType.
* @param[in] type Audio device type
* @return Number of devices
*/
extern "C" TRRO_EXPORT int TRRO_getDeviceCount(MediaDeviceType type);
```

Parameters	Meaning
MediaDeviceType	Audio device types: Microphone (MediaDeviceTypeMic), Speaker (MediaDeviceTypeSpeaker)

Querying Audio Device Names

Use instructions: Querying the names of audio capture/playback devices, which can be used to filter target audio devices

```
/**
* @name  TRRO_getDeviceName(Experimental)
* @brief Query the device name based on MediaDeviceType and index.
* @param[in] type Device type
* @param[in] index Device index, value is [0, TRRO_getDeviceCount)
* @return Device name, returns null on failure
*/
extern "C" TRRO_EXPORT const char* TRRO_getDeviceName(MediaDeviceType
type, int index);
```

Parameters	Meaning
MediaDeviceType	Audio device types: Microphone (MediaDeviceTypeMic), Speaker (MediaDeviceTypeSpeaker)

index	Audio device index, value is [0, TRRO_getDeviceCount)

Changing the Current Audio Device

Usage Instructions: Change the current audio capture/playback device. The SDK will select the default audio device based on the configuration at startup. You can call this API to change it.

```
/**
* @name  TRRO_setCurrentDevice(Experimental)
* @brief Set the current capture or playback device based on
MediaDeviceType and index.
* @param[in] type Device type
* @param[in] index Device index, value is [0, TRRO_getDeviceCount)
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_setCurrentDevice(MediaDeviceType type,
int index);
```

Parameters	Meaning
MediaDeviceType	Audio device types: Microphone (MediaDeviceTypeMic), Speaker (MediaDeviceTypeSpeaker)
index	Audio device index, value is [0, TRRO_getDeviceCount)

Log/Error Message APIs

Register Log Callback Function

Usage Instructions: This API is used to register a log callback function for upper-layer applications to obtain logs.

```
/**
* @name  TRRO_registerLogCallback
* @brief Register log callback function
* @param[in]   context             Context pointer
* @param[in]   callback            Callback function
```

```
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerLogCallback(void *context,
TRRO_OnLogData *callback);
```

Parameters	Meaning
context	target remote device id
callback	the permission to be set

```
/**
* @name TRRO_OnLogData
* @brief Log callback
* @param[in]     context        Context pointer
* @param[in]     msg            Log content
* @param[in]     level          Log level, refer to the enumeration
TrroLogLevel
* @return void
*/
typedef void TRRO_OnLogData(void *context, const char *msg, int level);
```

Parameters	Meaning
context	target remote device id
msg	Log content
level	Log level, refer to the enumeration TrroLogLevel

Register Error Event Callback

Usage Instructions: This API is used to register error event callback, which is triggered when an exception occurs on the streaming end. You can choose to use it as needed.

```
/**
* @name TRRO_registerOnErrorEvent
* @brief Register error event callback function
* @param[in] context              Context pointer
* @param[in] callback             Callback function
* @return 1 for success, other failed
```

```
*/
extern "C" TRRO_EXPORT int TRRO_registerOnErrorEvent(void* context,
TRRO_OnErrorEvent * callback);
```

Parameters	Meaning
context	target remote device id
callback	the permission to be set

```
/**
* @name TRRO_OnErrorEvent
* @brief Error message callback
* @param[in] context        Context pointer
* @param[in] error_code     Error code, negative value
* @param[in] error_msg      Error message
* @return void
*/
typedef void TRRO_OnErrorEvent(void* context, int error_code,  const
char* error_msg);
```

Parameters	Meaning
context	target remote device id
error_code	Error Code
error_msg	Error Message

Error Message Query

Use Instructions: Query the error message corresponding to the error code.

```
/**
* @name  getErrorMsg
* @brief Returns the error message based on the error code.
* @param[in] errorCode   Error code
* @return errorMsg   Troubleshooting help information
*/
extern "C" TRRO_EXPORT const char* getErrorMsg(int errorCode);
```

Parameters	Meaning
errorCode	To query the error code, you can input either positive or negative error codes.
errorMsg	Corresponding Error Message

Recording APIs

Start Recording

Use Instructions: Create a video recorder for local recording of video streams from field devices.

```
/**
* @name TRRO_startRecorder(Experimental)
* @brief Start Recording
* @param[in]   recorderID          Recording ID (To record SDK
collected streams, the ID should be set to the corresponding stream, and
"record_on":1 should be added to stream_config in the config file)
* @param[in]   format              Recording format: 0 : 264
* @param[in]   width               Encoding width
* @param[in]   height              Encoding height
* @param[in]   jump                Recording frame skip count (record
one frame every few frames)
* @param[in]   fps                 Encoding frame rate
* @param[in]   bps                 Bit rate
* @param[in]   filename            File name
* @param[in]   config              Recording options (reserved)
* @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);
```

Parameters	Meaning
recorderID	Recording ID (To record SDK collected video streams, the ID should be set to the corresponding stream ID, and "record_on":1 should be added to the

	stream_config of the corresponding stream. The corresponding stream data will be automatically imported into the recorder with the matching ID.)
format	Recording format: 0 : 264
width	Encoding width
heigh	Encoding height
jump	Recording frame skip count (record one frame every few frames)
fps	Encoding frame rate
bps	Bitrate
filename	File Name
config	Recording options (reserved)

Sending Recording Data

Use instructions: Input the video data to be recorded. When recording external input video streams, use with TRRO_startRecorder API and TRRO_stopRecorder API.

```
/**
* @name TRRO_sendRecordVideoData(Experimental)
* @brief Send recording data
* @param[in]   recorderID          Recording ID
* @param[in]   data                Recording data
* @param[in]   width               Data source width
* @param[in]   height              Data source height
* @param[in]   size                Recording data size
* @param[in]   format              Video source format 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);
```

Parameters	Meaning
recorderID	Recorder ID
data	Recording data

width	Encoding width
heigh	Encoding height
size	Recording data size
format	Encoding frame rate
bps	Video source format 1 yuv420, 4 YUYV

Switching Recording File

Use instructions: Switch recording file, used with recording feature.

```
/**
 * @name    TRRO_switchRecorderFile(Experimental)
 * @brief   Switch Recording File
 * @param[in]    recorderID     Recording ID
 * @param[in]    filename        File name
 * @return   1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_switchRecorderFile(int recorderID, const
char* filename);
```

Parameters	Meaning
recorderID	Recorder ID
filename	Name of the recording file after switching

Stop recording

Use instructions: Stop recording, used with recording feature.

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

Parameters	Meaning
recorderID	Recorder ID

Video Capture APIs

Starting Capture

Use instructions: Used to capture camera data and callback the camera data to the user through a callback function.

```
/**
 * @name TRRO_onVideoCaptureFrame
 * @brief Data callback when TRRO_startVideoCapture is called externally
 * @param[in] context      Context pointer
 * @param[in] capture_id   Current source ID of the captured video
 * @param[in] data         Captured video data
 * @param[in] length       Length of the captured video data
 * @param[in] width        Captured video width
 * @param[in] height       Captured video height
 * @param[in] color_format Captured video format
 * @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 Start camera capture
* @param[in] context   Context
* @param[in] url       Capture path, /dev/video* for camera, rtsp
address for rtsp
* @param[in] protocol  When equal to kV4L2_DMA or kV4L2_MMAP, for camera
capture. When equal to kRTSP, for rtsp address
* @param[in] TrroColor Captured video format
* @param[in] width     Captured video width
* @param[in] height    Captured video height
* @param[in] fps       Video capture frame rate
* @param[in] callback  Video capture callback
```

```
* @param[in, out] capture_id Unique identifier for the capture stream
(returns an error if duplicated), supports input and output (use input
value only when (*capture_id) > 0, otherwise internally generate and
modify this value)
* @return Returns error code if capture start fails, returns 1 for
success
*/
extern "C" TRRO_EXPORT int TRRO_startVideoCapture(void *context, const
char* url, VideoCaptureProtocol protocol, TrroColor color_format, int
width, int height, int fps, TRRO_onVideoCaptureFrame callback, unsigned
long long* capture_id);
```

Parameters	Meaning
context	Context pointer, used for callback positioning
url	Capture path, /dev/video* for camera, rtsp address for rtsp
protocol	kRTSP for network camera RTSP protocol capture, kV4L2_MMAP for USB/GMSL camera V4L2 protocol capture, kV4L2_DMA for NVIDIA Jetson platform V4L2 capture
TrroColor	Captured video format, V4L2 protocol for camera image format, RTSP protocol capture for target callback format
width	Captured video width
height	Captured video height
fps	Captured video frame rate
callback	Video capture callback
capture_id	Capture identifier, used to distinguish different captures, passed as a pointer, supports value specification or auto-generation (use specified input value when (*capture_id) > 0, use SDK auto-generated value when 0)

Stopping the Capture

Use instructions: Used to capture camera data and callback the camera data to the user through a callback function.

```
/**
* @name  TRRO_stopVideoCapture
```

```
* @brief Stop camera capture
* @param[in] capture_id Capturing source ID (cannot be 0)
* @return Returns error code if capture stop fails, returns 1 for
success
*/
extern "C" TRRO_EXPORT int TRRO_stopVideoCapture(unsigned long long
capture_id);
```

Parameters	Meaning
capture_id	Capturing source ID (cannot be 0)

Network Evaluation API

Starting Network Evaluation

Use instructions: Used to evaluate whether the network quality meets the requirements for video transmission, and can be used to test the network before business starts. This API is a blocking function. When using this API, the SDK will actively push the video stream for network testing and collect network metrics for evaluation. If the SDK is already pushing the video stream, it is recommended to directly observe the network metrics through the media status callback API.

```
/**
* @name TRRO_testNetworkQuality
* @brief Evaluate network status                Note: This function is a
blocking function
*                                  If the network connection is not
successful, the timeout period is 10s plus the network detection time
* @param[in]   stream_ids           Array of Stream IDs to be detected
* @param[in]   stream_size          Size of the array of Stream IDs to
be detected
* @param[in]   test_time            Duration of the detection after a
successful connection, minimum 2s, maximum 10s;
*
* @return int Network assessment 0: Unable to assess network, 1: Good
network, 2: Poor network, 3: Unavailable network
*/
```

```
extern "C" TRRO_EXPORT int TRRO_testNetworkQuality(int* stream_ids, int
stream_size, int test_time);
```

Parameters	Meaning
stream_ids	Array of video stream IDs for network assessment
stream_size	Size of the video stream ID array
test_time	Detection duration, minimum 2 seconds, maximum 10 seconds

Getting SDK Information

SDK version API

Use Instructions: Get the sdk version number.

```
/**
* @name  getSdkVersion
* @brief Get the SDK version information
* @param[in] void
* @return version number
*/
extern "C" TRRO_EXPORT const char* TRRO_getSdkVersion();
```

Field Device Configuration Instructions

Last updated：2025-02-11 16:49:23

Sample JSON Configuration File

```
Filename: config.json
File Location: $(workspace)/config.json
File Type: json
Note: Remove comments after "//" when using. The minimum node is as
follows
{
 "device_id":"dev1",  //Change to the device ID created in the console
 "device_name":"vin234",
 "device_streams":1,  //If there are multiple inputs, change the number
of streams here and increase the number of elements in the
streams_config array
 "cloud_mode":"public",
 "certificate":"./device.pem",
 "projectid" : "xxxxx", //Change to the project ID created in the
console
 "password": "xxxxx",  //Change to the password created in the console
 "streams_config": [
 {
     "fps":30,
     "bps":2000,
     "width":1920,
     "height":1080,
     "camera":0,  //Change to the actual connected camera, corresponding
to /dev/videox
     "protocol":"v4l2" //If inputting external yuv data, change to
outside. If inputting external encoded stream data, change to outenc
 }
 ]
}
```

Meaning of JSON Configuration Root Node

Node	Type	Meaning	Defau	Remarks	Parameter Range

Name			lt value		
device_id	**String**	Device ID, a unique code used for registering to the server	–	Required	–
device_name	**String**	Device name, used to identify device information	–	Required	–
force_login	Number	Whether to force login	0	Whether to remove previous device login for the same ID	• 0: When the same ID is already logged in, the current device login is removed • 1: When the same ID is already logged in, the current device will force login, kicking out the previous device
server_ip	**String**	Signaling server IP	–	No need to specify for public cloud account, mandatory for privatization	–
server_port	**Number**	Signaling server port	–	No need to specify for public cloud account, mandatory for privatization	–
rtc_server_ip	**String**	Media server IP	–	No need to specify for	–

				public cloud account, mandatory for privatization	
rtc_server_port	**Number**	Media server port number	–	No need to specify for public cloud account, mandatory for privatization	–
projectid	**String**	Project ID	–	Required for public cloud	–
cloud_mode	**String**	Cloud mode	"private"	Optional	• public: Public cloud • private: Private cloud
password	**String**	device key	–	Required for public cloud	–
certificate	**String**	server public key certificate	–	Required for public cloud, device.pem file path	–
log_enable	**Boolean**	Log switch	0	Optional	• 0: disabled • 1: info log • 2: debug + info log
sdk_mode	**String**	Stream transmission mode	"server"	Optional	–
min_port	**Number**	Minimum local transmission port	50000	Optional	–
max_port	**Number**	Maximum local	50100	Optional	–

		transmission port			
audio_enable	**Number**	Collect audio	0	Optional	• 0: Do not collect • 1: Collect
audio_receive	**Number**	Receive audio	1	Optional	• 0: Do not receive • 1: Receive
device_streams	**Number**	Number of device video streams	–	Required	1–8
streams_config	**JSON Array**	Stream configuration array	–	Required	The number is consistent with device_streams
network_bind	**String Array**	Specified NIC IP	–	Optional	Set when there is a specified NIC or multiple NIC requirements, `"network_bind":["ip1","ip2"]`, configure the local IP of the specified communication NIC
output_path	**String**	Cloud File Storage path	"./"	Optional	Configure the output path for logs and temporary files during SDK runtime
log_file_num	**Number**	Number of stored log files	7	Optional	–
use_local_conf	**Number**	Use local configuration	0	Optional (private deployment only)	• 0: Use remote configuration • 1: Use local configuration
use_local_license	**Number**	Use local license	0	Optional (private deployment only)	• 0: Use remote license • 1: Use local license

fec_enable	Number	Enable FEC	0	Optional	• 0: Disable local FEC • 1: Enable local FEC
up_fec_rate	Number	Specify uplink FEC redundancy	0	Optional	• 0: Automatic redundancy • >0: Specify redundancy percentage, up to 100
down_fec_rate	Number	Specify downlink FEC redundancy	0	Optional	• 0: Automatic redundancy • >0: Specify redundancy percentage, up to 100

Streams Config Meaning

Node Name	Type	Meaning	Default Value	Remarks	Parameter Range
fps	**Number**	Video Frame Rate	–	Required	–
bps	**Number**	Stream Transmission Bitrate	–	Required, configure according to image quality needs and actual network load. The higher the bitrate, the clearer the image quality, and the larger the data transmission.	–
min_bps	**Number**	Expected minimum transmission bitrate	Default value is the	Optional, fill in according to minimum image quality needs.	–

			same as bps		
width	**Number**	Video source width	–	Required, for camera capture use camera resolution, for external input video, fill in according to input resolution	–
height	**Number**	Video source height	–	Required, for camera capture use camera resolution, for external input video, fill in according to input resolution	–
encode_width	**Number**	Encoded video width	Default is the same as width	Optional	–
encode_height	**Number**	Encoded video height	Default is the same as height	Optional	–
min_width	**String**	Minimum adjustable encoding resolution width	Same as encode_width	Optional	Set when dynamic resolution is enabled
min_fps	**Number**	Minimum fps	20	Optional	Set when adjusting dynamic frame rate

forece_min	Number	Enable forced minimum bitrate	0	Optional	• 0: Dynamically control the minimum bitrate • 1: Set the expected minimum bitrate as the minimum bitrate
major	**Number**	Mainstream Configuration	0	Optional	• 0: No • 1: Yes, high priority for mainstream
protocol	**String**	Collection Protocol	“v4l2 ”	Optional	• "v4l2": Collect v4l2 camera data by SDK • "outenc": customers input encoded stream data through the API • "outside": customers input image data through the API
camera	**Number**	v4l2 camera number	0	Fill in the device /dev/videoX corresponding to number X, this field is effective when the protocol field is configured as "v4l2"	–
codec	**Number**	Encoding type	0	Optional	• 0：h264 • 1：h265 • 2：av1

encoder_type	**Number**	Video Codec Methods	1	Optional	• 0: software encoding preferred • 1: hardware encoding preferred
frame_type	**Number**	yuyv–uyvy image reversal	0	Optional	• 0: no reversal • 1: reversal
record_on	**Number**	Enable local storage	0	Optional	• 0: disabled • 1: enabled
capture_acc	**Number**	Enable collection acceleration	0	Optional	• 0: disabled • 1: enabled
need_mirror	**Number**	Enable mirror mode	0	Optional	• 0: disabled • 1: enabled
always_push	**Number**	Enable forced push streaming	0	Optional	• 0: disabled • 1: enabled

streams_config: Stream Configuration, protocol: Extension and Network Camera Collection Configuration

1. For pulling RTSP camera scenes, refer to the streams_config node configuration.

```
Reference configuration: ```json "streams_config": [ { "fps": 25, "bps": 2000, "width": 1920, "height": 1080, "protocol": "normal", "cameras": [ { "width": 1920, "height": 1080, "protocol": 1, "url": "rtsp://xxxxx" } ] } ] ```
```

2. Multi–screen stitching scenario reference streams_config node configuration

```
"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
                }
            ]
        }
        }
    ]
```

Node Name	Type	Meaning	Default Value	Remarks	Parameter Range
protocol	String	Image Process type	"normal"	Default no processing	• normal: normal transmission after decoding • rtsp_enc: pass-through encoded stream • combine: mixed stream stitching
cameras	JSON Array	video source camera	–	When multiple camer	–

				a combinations are involved, multiple camera nodes need to be configured.	
width (in the cameras node)	Number	camera capture width	–	–	–
height (in the cameras node)	Number	camera capture height	–	–	–
protocol (in the cameras node)	Number	collection protocol type	1	–	• 1: rtsp collection • 2: v4l2 mmap collection • 3: v4l2 dma collection
url (in the cameras node)	String	collection data source address	–	camera collection protocol type	• protocol=1: network camera pull stream rtsp URL • protocol=2/3：/dev/videox

Error Codes

Last updated：2025-02-11 16:49:45

The definition of error codes is located in the trro_field.h header file. The function return value is a negative value of the error code. For example, if the error code is 16777215, the return value is -16777215. The specific meaning of error codes are shown in the table below:

Module	Error Code	Hexadecimal	Decimal	Error Reson
General module	TRRO_SUCCED	0x00000001	1	Executed successfully
	TRRO_COMMON_ERROR	0x00FFFFFF	16777215	General failure
Configuration (Troubleshoot file configuration content)	TRRO_CONFIG_ERROR	0x01FFFFFF	33554431	General failure of configuration module
	TRRO_CONFIG_PARSE_FAILED	0x01000002	16777218	Joint node parsing error, check if the minimum startup node in the configuration file (Field Device Configuration Instructions) is complete
	TRRO_CONFIG_ILLEGAL	0x01000003	16777219	Check if width, height, and protocol in the configuration file match the API input parameters
	TRRO_CONFIG_UNEXIST	0x01000004	16777220	Configuration file does not exist or has a length of 0
	TRRO_CONFIG_CER_FAILED	0x01000005	16777221	Start using public cloud mode (cloud_mode node set to public), but the certificate file device.pem is missing or the certificate file verification failed.
	TRRO_CONFIG_LIC_FAILED	0x01000006	16777222	Start using private mode (cloud_mode node set to private), but the license.txt file is missing.
	TRRO_CONFIG_STREAMS_SIZE_ERROR	0x01000007	16777223	Exceeded the maximum supported stream count.

	TRRO_CONIFG_PORT_RANGE_ILLEGAL	0x01000008	16777224	Illegal port range configuration.
	TRRO_CONFIG_LOG_PERMISSON_DENIED	0x01000009	16777225	Insufficient permissions to open the log file.
Initialize the module (Troubleshoot API call order and file configuration content)	TRRO_INIT_ERROR	0x02FFFFFF	50331647	Initialization failed
	TRRO_INIT_INPUT_ILLEGAL	0x02000002	33554434	Initialization failed or configured device ID, port range is illegal
	TRRO_INIT_PARSE_FAILED	0x02000003	33554435	Initialization parsing configuration node abnormal
	TRRO_INIT_CREAT_MEDIAMODE_FAILED	0x02000004	33554436	Media module creation failed
	TRRO_INIT_PRM_ERROR	0x02000005	33554437	Parameter error
	TRRO_INIT_INVALUDE_INPUT	0x02000006	33554438	Initialization parameter illegal
	TRRO_INIT_REPEAT	0x02000007	33554439	Duplicate initialization
Troubleshoot whether the console device authorization is normal (Console)	TRRO_INIT_LICENSE_CHECK_FAILED	0x02000010	33554448	LICENSE verification failed
	TRRO_INIT_LICENSE_FILE_ERROR	0x02000011	33554449	LICENSE file error
	TRRO_INIT_LICENSE_CHECK_TIME_FAILED	0x02000012	33554450	Authorization expired
	TRRO_INIT_LICENSE_CHECK_DEVICE_FAILED	0x02000013	33554451	Hardware verification failed
	TRRO_INIT_LICENSE_CHECK_STREM_FAILED	0x02000014	33554452	The number of authorized streams is less than the

				number of device configuration streams
	TRRO_INIT_LICENSE_CHECK_ID_FAILED	0x02000015	33554453	Device ID verification failed
	TRRO_INIT_PUBLIC_LICENSE_CHECK_TIMEOUT	0x02000100	33554688	Public cloud LICENSE verification timed out
	TRRO_INIT_PUBLIC_LICENSE_CHECK_NOT_BIND	0x02000101	33554689	Public cloud device not bound to LICENSE
	TRRO_INIT_PUBLIC_LICENSE_CHECK_NOT_ENOUGH	0x02000102	33554690	Insufficient public cloud LICENSE quantity
	TRRO_INIT_PUBLIC_LICENSE_CHECK_OVERTIME	0x02000103	33554691	Public cloud LICENSE expired
Signaling Server Module (Troubleshoot network and server status)	TRRO_SIGNAL_ERROR	0x03FFFFFF	67108863	Signaling Module Exception
	TRRO_SIGNAL_REGIST_FAILED	0x03000002	50331650	Failed to register MQTT signaling server
	TRRO_SIGNAL_STATUS_ABNORMAL	0x03000003	50331651	MQTT signaling service connection exception
	TRRO_SIGNAL_MESSAGE_FAILED	0x03000004	50331652	Failed to process signaling message
	TRRO_SIGNAL_CONNECT_OUTTIME	0x03000005	50331653	Server connection timeout, this error code can be ignored if set to asynchronous startup.
	TRRO_SIGNAL_DEVICEID_OR_PASSWORD_INCORRECT	0x03000006	50331654	Incorrect signaling server connection username or password.
Stream collection module (Check whether	TRRO_CAPTURE_ERROR	0x04FFFFFF	83886079	Collection module exception
	TRRO_CAPTURE_OPENDEVICE_FAILED	0x04000002	67108866	Failed to open the device. Check if the camera is being used by another program

the camera access and resolution configuration are correct.)	TRRO_CAPTURE_GETSOURCE_FAILED	0x04000003	67108867	Timeout failure in reading camera data
	TRRO_CAPTURE_UNKNOWN_CAPTURETYPE	0x04000004	67108868	Unknown camera data source type
Stream transmission module	TRRO_CONNECT_ERROR	0x05FFFFFF	100663295	Connection abnormal
	TRRO_CONNECT_OUTTIME	0x05000002	83886082	Timeout disconnection
Message transmission module	TRRO_MESSAGE_ERROR	0x06FFFFFF	117440511	Failed to send the message
	TRRO_MESSAGE_CHANNEL	0x06000002	100663298	No available message sending channel
	TRRO_MESSAGE_BYTE_EXCEED	0x06000003	100663299	Sending bytes exceeded the limit
Storage module	TRRO_STOR_ERROR	0x07FFFFFF	134217727	Storage exception, check API call order
	TRRO_STOR_UNENABLE	0x07000001	117440513	Storage feature not enabled
	TRRO_STOR_IDEXIST	0x07000002	117440514	Repeatedly enable storage
	TRRO_STOR_ID_ILLEGAL	0x07000003	117440515	Storage ID is invalid
	TRRO_STOR_PARAM_ILLEGAL	0x07000004	117440516	Storage data source input parameter is invalid
	TRRO_STOR_UNSET_FILENAME	0x07000005	117440517	Storage file name is empty
External collection module	TRRO_EXTERNAL_RESIZE	0x08000001	134217729	External acquisition mode error

Extension module	TRRO_ERROR_CALLBACK_CAMERA	0x09010000	151060480	Camera asynchronous callback error
	TRRO_ERROR_CALLBACK_MIC	0x09020000	151126016	Microphone asynchronous callback error
	TRRO_ERROR_CALLBACK_BANDWIDTH_LIMIT	0x09030000	151191552	Evaluated network bandwidth cannot meet the minimum bitrate requirement
Reserved	TRRO_UNSUPPORT	0x0F000001	251658241	This call or function feature is not supported currently