Real-time Teleoperation

Remote Device SDK

Product Documentation

Contents

Remote Device SDK

Basic Introduction

Last updated：2025-02-11 16:50:11

Scope of Application

This SDK is used for remote device (e.g. cockpit), suitable for Linux and Windows 64-bit systems, and provides access in the form of so, dll dynamic link libraries.

Main Functions

This SDK provides low-latency audio, video and control/status data transmission for remote real-time operation scenarios. The main functions of SDK are given as follows:

Video Rendering

Render video in a window with a specified handle and automatically respond to window scaling events.

Video Decoding

Supports decoding of video formats such as H264, H265 and AV1 with hardware decoder support.

Video Transmission

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

Binary Data Transmission

Supports the transparent transmission of binary data from remote device to field device.

Stream Management

The SDK can dynamically manage the stream push status of the field device according to viewing requirements.

Video Latency and Link Status Reporting

The SDK can monitor video latency and link status, and report the corresponding status data through a callback interface.

Basic Concepts

Device ID

Divided into field device (e.g. vehicle) ID and remote device (e.g. cockpit) ID. The device ID is unique, and used to identify the device identity, . If there are two devices with the same device ID connected to the signal server, the server will kick out one device according to the login sequence and the force-login option.

Video Stream ID

Used to identify the video stream index of the field device, and the range of values is [0, size of the streams_config array). When the remote device pulls the video stream, it specifies the video stream to be pulled through the field device ID + video stream ID.

Receiver Handle ID (conn_fd)

Used to identify the video stream receiver of the remote device, and the range of values is [0, max_streams). The video stream receiver is linked with the field device video stream and the video rendering window. When pulling the video stream, you need to specify the corresponding video stream receiver to receive the stream; The following is the process of receiving the video stream.
field device video stream >> remote device video stream receiver >> video rendering window

Video Window Handle

Used to identify the video rendering window, which is a pointer to the video window, such as HWND for windows platform. Through the video window handle, the SDK can render the video image on the specified video window.

Operation Permisson

The permission is used to determinde whether the remote device can control the field device, e.g. send the control data and audio data to the field device. The operation permission is managed by the operation permission SDK API.

Session Permisson

The permission is used to determinde whether the remote device can watch the field device, and get the info/video/data from the field device. The session permission is managed by the cloud API or the cloud console.

Configuration File

It is a JSON format file (i.e. config.json), which contains the initialization configuration of the SDK.

SDK Call Procedure

Last updated：2025-02-11 16:50:28

Recommended Steps

> **Note**
> The registration of onSignalState callback should be called before SDK initialization, and other callback functions can be registered after initialization.

Step 1: Initialization

Register the onSignalState callback, call the Init interface to initialize SDK by loading a json config file or inputting a json config string.

Step 2: Register callback functions as needed

According to needs, register permission callback, report data callback, log callback, video connection status callback, video media and link information callback, latency information callback, video image data callback and other functions to handle corresponding status information and events.

Step 3: Set video rendering window handle

If rendering video by the application program based on the SDK callback video image data, this step can be skipped. If rendering video by SDK (only support for windows), call TRRO_setWindows interface to link the video receiver handle ID and the video rendering window handle, so that the video stream received by the receiver handle can be rendered on the linked video window.

Step 4: Connect video streams

Call TRRO_connect interface to pull the specified video streams of the specified field devices through the specified video receiver handle. This interface can be called multiple times to connect video streams from different devices. If the video receiver handle is switched to recevie a new video stream, the old video stream will be automatically disconnected. TRRO_disconnect interface can also be called to disconnect the video stream.

Step 5: Send control data as needed

call TRRO_sendControlData Interface to send the input binary data to the specified field device. Before calling the interface, the operation permission for the field device should be the master permissoin. TRRO_requestPermission and TRRO_OnOperationPermissionState interfaces can be used to request and check the operation permission.

Step 6: Destory the SDK

When the program exits, call the destroy interface to destroy the SDK, especially for C# since C# program may not automatically destroy dll resource when it exits.

Sample Code

```
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "trro_remote.h"

//This sample does not include render code
int main() {
    //register signal state callback
    TRRO_registerSignalStateCallback(nullptr, [](void *context,
SignalState state) {
        if(state == kTrroReady) {
            //connect to server successfully
            printf("init success \n");
        }
        if (state == kTrroAuthFailed) {
            //auth failed
            printf("device_id or password is incorrect\n");
        }
        if (state == kTrroKickout) {
            //kicked out caused by the same device ID login
            printf("the device is kicked by server, may be there is
another device using the same device id\n");
        }
    });

    // init function can be replaced by other init function, e.g.  json
string as input
    int ret = TRRO_initJsonPath("./config.json");
    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);
```

```
        }
    }

    //register the callback functions as needed
    //register video connection state callback
    TRRO_registerOnState(this, [](void* context, const char* gwid, int
stream_id, int conn_fd, int state) {
      printf("###### onstate gwid %s, streamid %d  conn_fd %d  state
%d\n", gwid, stream_id, conn_fd, state);
    });
    //register video transmission state callback
    TRRO_registerOnMediaState(this, [](void* context, int windows_id,
int fps, int bps, int rtt, long long decd, int jitter,
        long long packets_lost, long long packets_received, int stun) {
           printf("onMediaState window id %d fps %d bps %d rtt %d decd
%lld rb %d lost %lld rec %lld stun %d\n",windows_id, fps, bps, rtt,
decd, jitter, packets_lost, packets_received, stun);
    });
    //register video latency info callback
    TRRO_registerAllLatencyCallback(this, [](void* context, const char*
gwid, int windows_id, long long latency1, long long latency2, long long
gateway_time, int vcct) {
        printf("allLatency %s, %d, latency1 %lld latency2 %lld video
time %lld, vcct %d\n", gwid, windows_id, latency1, latency2,
gateway_time, vcct);
    });

    //register video image data callback, can be used for application
program rendering
    TRRO_registerRemoteFrameCallback(this, [](void * context, const char
* gwid, int stream_id, int conn_id, const char* data, int width, int
height, long long videotime) {
       //here can realize the rendering functions, but please do not
block the callback thread.
       //you can try the async way to render video otherwise the video
will be lagging
       //showMyVideo(gwid, conn_id, (unsigned char*)data, width,
height);
    });
```



```
    //receiver handle ID array
    int confds[8] = {0,1,2,3,4,5,6,7};
    //rendering windows handle
    WindowIdType window_hwnds[8] = { nullptr};
    //for SDK internal rendering, set the target win32 HWND, only for
windows;
    //for application program rendering, just set nullptr,and realize
showMyVideo function
    TRRO_setWindows(confds, window_hwnds, 8);
    //connect the video streams, assume two devices, each with 4 video
streams.
    int stream_num = 4;
    int stream_indexs[4] = {0,1,2,3};
    //connect field device1, 4 video streams (0-3), using the receiver
handle (0-3)
    TRRO_connect("field_deviceId1", "", stream_num , &stream_indexs[0],
&confds[0]);
    //connect field device1, 4 video streams (0-3), using the receiver
handle (4-7)
    TRRO_connect("field_deviceId2", "", stream_num , &stream_indexs[0],
&confds[4]);
    //wait for exiting
    while(true){
        sleep(3000);
        // Disconnect the video stream
        TRRO_disconnectAll();
        // Exit the SDK and release resources
        TRRO_destroy();
        break;
    }
    return 0;
}
```

C Remote Device SDK API

Last updated：2025-02-11 16:50:48

Introduction

This document provides a detailed description of the remote device C SDK API, which are suitable for Windows/Linux 64-bit systems. The API is defined and briefly described by the header file ***trro_remote.h*** in the SDK package. The overview and detailed usage descriptions of the SDK API are given as follows.

API Overview

Initialization and Destruction Interfaces

Interfaces	Description
TRRO_initJsonPath	Initialize the SDK using a JSON configuration file path
TRRO_initJson	Initialize the SDK using a JSON string
TRRO_destroy	Destroy the SDK.

Field Device Query Interfaces

Interfaces	Description
TRRO_getGwList	Retrieves a list of online field devices with session permissions.
TRRO_getGwInfo	Retrieves detailed information about a specific field device, requiring session permissions for that device.

Video Operation Interfaces

Interfaces	Description
TRRO_setWindows	associate the video receiver handle with the video rendering window handle
TRRO_connectField	pull the video streams from field devices using specified video receiveres
TRRO_disconnect	disconnect specified video receivers
TRRO_disconnectAll	disconnect all the video receivers
TRRO_SetRenderConfig	updates the video rendering options, e.g. super-resolution, low light enhancement, rotation, and mirroring.

TRRO_fieldDeviceEncodeConfig	updates the encoding parameters for the specifie video stream of the field device.
TRRO_startMediaPush	bypass push the received video to the target rtmp url
TRRO_stopMediaPush	stop the bypass video push

Audio and Video Input/Output Interfaces

Interfaces	Description
TRRO_registerRemoteFrameCallback	register the decoded raw video image callback, default YUVI420 format.
TRRO_registerRemoteEncodedFrameCallback	register the received encoded video frame callback
TRRO_useExternalDecoder	indicate the external video decoder to SDK，which will disable internal decoder and do not callback the raw video image
TRRO_externAudioData	externally input the audio data (PCM data) for audio transmission
TRRO_audioMute	mute / unmute the specified field device audio

Message Interfaces

Interfaces	Description
TRRO_sendControlData	send the binary message to the specified field device
TRRO_registerReportDataCallback	register the callback function for receiving the binary message from field devices

Operation Permission Interfaces

Interfaces	Description
TRRO_requestPermission	send the operation permission request to the specified field device， e.g. master permission or guest permission （watch only）
TRRO_registerOnOperationPermissionState	register the callback function for receiving the permission state update notification from field device

Status Callback Interfaces

Interfaces	Description
TRRO_registerAllLatencyCallback	register the callback function for various latency monitoring, e.g. video tranmission or control latency
TRRO_registerSignalStateCallback	register the callback function for monitoring the server access state of remote device
TRRO_registerOnState	register the callback function for monitoring the video link state of remote device
TRRO_registerOnMediaStateInfo	register the callback function for monitoring video transmission and network status of remote device
TRRO_registerOnAudioMediaStateInfo	register the callback function for monitoring video transmission and network status of remote device
TRRO_registerFieldSideNetworkState	register the callback function for retrieving the network status of field device

Diagnostic Interfaces

Interfaces	Description
TRRO_registerLogCallback	register the callback function for retrieving the sdk log data
TRRO_diagRequest	start the end-to-end diagnosis for checking the video transmission problem of the connected field device
TRRO_registerOnDiagReport	register the callback function for retrieving the diagnosis report, which is helpful for problem positioning

Image Transform Interfaces

Interfaces	Description
TRRO_I4202ARGB	transform image data from the I420 color format to ARGB color format

SDK Info Interfaces

Interfaces	Description
TRRO_getVersion	get the SDK version

API Usage Description

Initialization and Destruction Interfaces

SDK is initialized based on the configurations. either a json file path or a json string can be inputted as the configurations.

Init SDK by json file path

```
/*
SDK Init will be blocked until the server is connected
*/
int TRRO_initJsonPath(const char* jsonPath);
```

parameters	description
jsonPath	the file path of the config json file
return	1 success, <=0 failed

Init SDK by json string

```
/*
SDK Init will be blocked until the server is connected
*/
int TRRO_initJson(const char* json);
```

parameters	description
json	the json string of configuration
return	1 success, <=0 failed

Destroy SDK

It will free all resources of SDK. After destroy, it should init the SDK again before calling SDK API.

```
void TRRO_destroy();
```

Field Device Query Interfaces

Retrive online field devices list

Obtain a list of online field devices that the remote device has their session permissions. If the number of devices is too large, the cloud server API is recommended instead of the SDK API.

```
//block until the list is retrived from cloud,  it should copy the
returned strings as soon as possible.
const char* TRRO_getGwList();
```

parameters	description

return	online field devices list in format of JSON string
	{
	"ret":0,
	"msg":"suc",
	"gateways":[
	{"deviceID":"xx","name":"xx","type":"gateway","status":"connected","streams":4,"timestamp":1682231838673,"version":"xx"}
	],
	"count":1
	}

Retrive online field device info

Obtain the specified online field device info. The remote device should have the session permission of the field device. If the field device is offline or the session permission is forbidden, the null value will be returned.

```
//block until the info is retrived from cloud,  it should copy the
returned strings as soon as possible.
const char* TRRO_getGwInfo(const char* gwid);
```

parameters	description
gwid	field device Id, with projectId prefix for public cloud, e.g. – public cloud: in format of projectId/deviceId – private cloud: in format of deviceId
return	field device info in format of JSON string {"ret":0, "msg": "suc", "deviceID":"xx", "name":"xx", "type":"gateway", "status":"connected","streams":4,"timestamp":1682231838673,"version":"xx","sdk_mode":"server2","ability":0,"licenseExpire":1745666980}

Video Operation Interfaces

Connect the video stream

The interface is used to pull the video stream of the field device. It can be called with multiple times to connect different video streams of different field devices. If connecting the field device with different SDK mode, the compatible option should be enabled.

> **Note:**
> In the default configuration (i.e. "auto_permission": 1), the auto permission mode will be enabled, so that the master permission request will be sent to the field device automatically when calling the connect interface,

```
// async mode to connect the video, the video connection status is
notified by onState callback
// can be called with multiple times to connect different field devices
int TRRO_connectField(const char* gwid, int streams_num, int*
streams_id, int* conn_fds, int compatible = 0, const char* record_config
= NULL);
```

parameters	description
gwid	field device Id, with projectId prefix for public cloud, e.g. projectId/deviceId
streams_num	size of streams_id array
streams_id	the array of the target stream id of video streams which is coresponding to the element index of streams_config array of field device
conn_fds	the array of the video receiver handle id for receiving video streams, same size as streams_id, starts from 0. one video receiver can only receive one video stream at same time, and the old video stream will be disconnected if using the same video receiver to connect new video stream.
compatible	1 for compatible mode, which can connect video stream with different sdk mode, but costing more time 0 for fast connection with configured sdk mode, the sdk mode should be same as the field device
record_config	Specifiy the record config for connected video strreams in format of json string

	{"file_names": [{"file":"test1_file", "duration":15}, {"file":"test2_file", "duration":15}]} where file is the path and name of record file, duration is the slice duration in unit of minutes. If not need recording, set it as NULL
return	1 success, <=0 failed

Set the video rendering windows

The interface is only used for SDK internal rendering. It associates the video receiver handle with the video rendering windows. So that the video stream received by the video receiver handle can be rendered in the target video window.

```
void TRRO_setWindows(int* conn_fds, WindowIdType * windows, int num);
```

parameters	description
conn_fds	the array of the video receiver handle Id
WindowIdType	the array of the video windows handle, for windows platform, HWND is the WindowIdType
num	size of the array

Close the video stream

The interface is used to close the video stream received by the video receiver.

```
int TRRO_disconnect(int* conn_fds, int fd_num);
```

parameters	description
conn_fds	the array of the video receiver handle Id
fd_num	size of the array
return	1 success, <=0 failed

Close all the video stream

The interface is used to disconnect all the video receiver, so that all the video stream will be closed.

```
int TRRO_disconnectAll();
```

parameters	description
return	1 success, <=0 failed

Update rendering config

The interface is used to update the rendering config of the video receiver, it will affect the rendering effect of the video stream received by the video receiver.
The interface is only worked in the SDK internal rendering mode.

```
int TRRO_SetRenderConfig(int *conn_fds, int *rotation, int *scale, int
*config, int num);
```

parameters	description
conn_fds	the array of the video receiver handle Id for updating render config
rotation	the array of the rotation degree, (-360, +360), default 0
scale	the array of the draw mode, default 0 0 full window stretch 1 keep origin ratio
config	the array of render effect config, the render effect config element is defined in format of 0x00000XYZ X low light enhance level, 0 for disable, 1-3 for different enhance level, 2 is recommend, default 0 Y super resolution option, 0 for disable, 1 for enable, default 0 Z mirror option, 0 for disable, 1 for enable, default 0
num	size of the array
return	1 success, <=0 failed

Update field device encoder config

The interface is used to update the encoder config of target video stream of fied device. The parameter not configured will keep the same on the field device.

```
int TRRO_fieldDeviceEncodeConfig(const char* gwid, int streams_id, const
char* encode_config);
```

parameters	description
gwid	the field device id，with projectId prefix for public cloud, e.g. projectId/deviceId
streams_id	the stream id that needs to update encoder config

encode_config	encode config to be updated，in the format of the json string. The encode parameter not involved in the json string will keep the same.
	{
	"fps": 30,
	"encode_width": 1920,
	"encode_height": 1080,
	"bps": 3000,
	"min_fps": 30,
	"min_bps": 1800,
	"force_min": 0,
	"min_width": 1920
	}

return	1 success，<=0 failed

Start video stream bypass push

The interface is used to bypass push the video stream received by the video receiver handle to the target RTMP URL. The push will be automatically stoped if the video receiver is disconnected.

```
int TRRO_startMediaPush(int conn_fd, const char* path);
```

parameters	description
conn_fd	the video receiver handle ID
path	the RTMP URL for bypass push，e.g. rtmp://xxxx
return	1 success，<=0 failed

Stop video stream bypass push

The interface is used to stop the bypass video push.

```
int TRRO_stopMediaPush(int conn_fd);
```

parameters	description
conn_fd	the video receiver handle ID
return	1 success, <=0 failed

Audio/Video Input/Output Interfaces

Video image callback

The interface is used to get the decoded video image of the received video stream. Default is in YUVI420 format.

Note
please do not block in the callback function for long time, otherwise the video will be lagging.

```
void TRRO_registerRemoteFrameCallback(void* context,
TRRO_onRemoteFrameData * callback, int frame_type = 0);

typedef void STD_CALL TRRO_onRemoteFrameData(void* context, const char*
gwid, int stream_id, int conn_fd, const char* data, int width, int
height, long long videotime);
```

parameters	description
context	context pointer for callback, can set as NULL if not used
gwid	the field device id, with projectId prefix for public cloud, e.g. projectId/deviceIds
stream_id	the video stream id
conn_fd	id of the video receiver handle receiving the video stream
data	decoded video image
width	image width
height	image height
videotime	timestamp for the video image received by the field device SDK, with the steady_clock of the field device

Encoded Video frame callback

The interface is used to get the encoded video frame of the received video stream.

> **Note**
> please do not block in the callback function for long time, otherwise the video will be lagging.

```
void TRRO_registerRemoteEncodedFrameCallback(void* context,
TRRO_onRemoteEncodedFrameData* callback);


typedef void STD_CALL TRRO_onRemoteEncodedFrameData(void* context, const
char* gwid, int stream_id, int conn_fd,
                                                    const char* data,
int len, TrroCodec trro_codec, bool is_key_frame,
                                                    long long
videotime);
```

parameters	description
context	context pointer for callback, can set as NULL if not used
gwid	the field device id, with projectId prefix for public cloud, e.g. projectId/deviceIds
stream_id	the video stream id
conn_fd	id of the video receiver handle receiving the video stream
data	encoded frame data buffer pointer
len	length of data buffer
trro_codec	codec of the encoded frame, Trro_H264, Trro_H265, Trro_AV1
is_key_frame	whether the video frame is IDR frame, true for IDR frame
videotime	timestamp for the video image received by the field device SDK, with the steady_clock of the field device

Use external video decoder

The interface is used to disable the SDK internal video decoder. If external decoder is enabled, the internal decoder and the video image callback will be disabled.


```
int TRRO_useExternalDecoder(bool external);
```

parameters	description
external	true disable internal decoder and video image callback, default false
return	1 success, <=0 failed

Input external audio data (pcm)

The interface is used to input the audio for transmission instead of the SDK audio capturer. The audio data should be PCM format with 16bit sampling.

> **Note**
> - To use the interface, please add the json key-value pair "audio_external":1 in the configuration, so that the SDK audio capturer will be disabled.
> - To let the field device receivie the audio data, the operation permission should be master.

```
int TRRO_externAudioData(const char* gwid, const char* data, int
data_size, int channel, int sample_rate);
```

parameters	description
gwid	ID of the target field device receiving the audio, with projectId prefix for public cloud, e.g. projectId/deviceIds
data	pointer to audio data buffer, pcm format with 16bit sampling
data_size	size of audio data buffer
channel	number of audio channel, 1 or 2
sample_rate	the audio sample rate, e.g. 16000 for 16khz sampling, support 8k/16k/24k/32k/44.1k/48kHz

Mute field device audio

The interface is used to mute the field device audio.

```
int TRRO_audioMute(const char* gwid, bool mute);
```

parameters	description
gwid	ID of the target field device, with projectId prefix for public cloud, e.g. projectId/deviceIds
mute	true for mute, false for unmute

Message Interfaces

Send message to field device

The interface is used to send binary data to the specified field device. Currently the data size is limited to 700 bytes per call, frequency is limited to 100Hz/s.

> **Note**
> When sending the data to a field device, it must have the master operation permission for the field device.

```
int TRRO_sendControlData(const char* gwid, const char* msg, int len, int qos = 0);
```

parameters	description
gwid	ID of the target field device, with projectId prefix for public cloud, e.g. projectId/deviceIds
msg	pointer to the data buffer of the msg
len	size of the data buffer
qos	qos mode for delivery 0 utrla low latency mode, with message loss possibility if network is poor 1 reliable mode, with larger latency if network is poor

Receive message from field device

The interface is used to receive messages from field device.

```
void TRRO_registerReportDataCallback(void* context, TRRO_onReportData * callback);

/*
```

```
* @name :  TRRO_onReportData
* @brief : Callback for receiving information from field devices
* @input : context     The context pointer for the callback, which
returns the context passed when the callback function is registered
*          gwid        The ID of the field device from which the message
originates
*          msg         Message, binary passthrough
*          len         Message length
*          qos         Message transmission QoS source, 0: unreliable
transmission, 1: reliable transmission
* @return : void
*/
typedef void STD_CALL TRRO_onReportData(void* context, const char* gwid,
const char* msg, int len, int qos);
```

parameters	description
context	callback context pointer
gwid	the ID of the field device sending the message
msg	size of the data buffe
msg	pointer to the data buffer of the msg
len	size of the data buffer
qos	message qos mode 0 utrla low latency mode 1 reliable mode

Operation permission Interfaces

Request master permission

The interface is used to request the master permission for the specified field device. The requested result will be notified by the callback function.

```
int TRRO_requestPermission(const char* gwid, int permisson);
```

parameters	description

gwid	the target field device id
permission	0 guest permission, only can watch 1 master permission, can watch, control and shout

Update the permission status

The interface is used to receive the permission updating notification, so that it can get the newest permission status.

```
void TRRO_registerOnOperationPermissionState(void* context,
TRRO_OnOperationPermissionState* callback);

typedef void STD_CALL TRRO_OnOperationPermissionState(void* context,
const char* field_devid, int self_permission, const char* master_devid);
```

parameters	description
context	callback context pointer
field_devid	the ID of the field device notifying the permission update
self_permission	current permission of the remote device itself 0 guest permission, only can watch 1 master permission, can watch, control and shout
master_devid	the remote device id which has the master permission of the field device. It will be empty if no device having the master permission

Status Callback Interfaces

Latency status callback

The interface is used to notify the latency of the connecting session, both monitoring video latency and control latency.

> **Note**
> When the video stream is connecting, the callback will be triggered ceaselessly . If the callback is interrupted (e.g. not triggered for 300–500ms), it indicates that the video stream may be disconnected caused by some reasons, e.g. the network outage, some fallback measures will be taken.

```
void TRRO_registerAllLatencyCallback(void* context,
TRRO_onAllLatencyReport * callback);

typedef void STD_CALL TRRO_onAllLatencyReport(void* context, const char*
gwid, int conn_fd, long long latency1, long long latency2, long long
videotime, int rcct);
```

parameters	description
context	callback context pointer
gwid	the field device id
conn_fd	the video receiver handle id
latency1	video transmission latency, in unit of ms, the precision depends on the difference of network up and down direction latency
latency2	video transmission latency (from field to remote) + control message latency (from remote to field), in unit of ms
videotime	the timestamp of the latest received video frame, in the system clock of field device
rcct	the round-trip control channel time (from remote to field then back to remote), in unit of ms

Siganl state callback

The interface is used to monitor the status of the signal connection between the remote device and the server.

```
int TRRO_registerSignalStateCallback(void *context, TRRO_onSignalState
*callback);

enum SignalState {
    kTrroReady = 0, /**< first connection is setup successfully */
    kTrroLost = 1,  /**< connection is lost, but the sdk will try to
reconnect */
    kTrroReup = 2,  /**< reconnect successfully */
    kTrroKickout = 3,  /**< the connection is kicked out due to same id
login */
```

```
    kTrroAuthFailed = 4,  /**< deviceid or password error */
};

typedef void TRRO_onSignalState(void *context, SignalState state);
```

parameters	description
context	callback context pointer
state	status of signal connection between remote device and server, refer to SignalState

Video connection state callback

The interface is used to monitor the status of the video stream connection between the remote device and the server.

```
void TRRO_registerOnState(void* context, TRRO_OnState * callback);

enum TrroState {
    kDisconnect = 0,   /**< video connection is disconnected */
    kConnecting = 1,    /**< begin to setup the video connection */
    kConnected = 2,    /**< video connection is connected successfully
*/
    kDisconnecting = 3,    /**< begin to disconnect the video connection
*/
};

typedef void STD_CALL TRRO_OnState(void* context, const char* gwid, int
stream_id, int conn_fd, int state);
```

parameters	description
context	callback context pointer
gwid	id of the field device of the video stream
stream_id	id of the video stream for the video connection
conn_fd	id of the video receiver handle for the video stream connection
state	status of video connection, refer to TrroState

Remote side video receiving status callback

The interface is used to monitor the video connection status between the remote device and the server.

```
void TRRO_registerOnMediaStateInfo(void* context, TRRO_OnMediaStateInfo
* callback);

struct TrroMediaState {
    int fps;     // video fps
    int bps;     // video bitrate, in unit of bps
    int rtt;     //  network rtt between remote device and server for
video transmission
    long long decd;    //video decode time, in unit of ms
    int jitter;    // network rtt jitter, in unit of ms
    long long packets_lost;    // packets loss rate (0-255), loss percent
= packets_lost / 255 * 100%
    long long packets_received; // num of received packets
    int stun;  // stun mode of the video connection, 0: host, 1: srflx,
2: prflx, 3: relay
    double lag_k100;  // 100ms video lag rate, equals
sum(max(video_frame_interval - 100, 0)) / video_duration
    double lag_k150;  // 150ms video lag rate, equals
sum(max(video_frame_interval - 150, 0)) / video_duration
    long long duration;  // duration of video stream, in unit of ms
};

typedef void STD_CALL TRRO_OnMediaStateInfo(void* context, int conn_fd,
TrroMediaState mediastate);
```

parameters	description
context	callback context pointer
conn_fd	id of the video receiver handle for the video stream connection
mediastate	media status of video connection, refer to TrroMediaState

Field side video sending status callback

The interface is used to monitor the video transmission status between the remote device and the server.

```
void TRRO_registerFieldSideNetworkState(void* context,
TRRO_onFieldSideMediaState * callback);

typedef void TRRO_onFieldSideMediaState(void* context, const char* gwid,
int stream_id, int fps, int bps, int rtt, int jitter, int lost);
```

parameters	description
context	callback context pointer
gwid	the field device id
stream_id	the video stream id
fps	number of frames per second of the field device video stream
bps	bitrate of the field device video stream, in unit of bps
rtt	network rtt between field device and server, in unit of ms
jitter	network rtt jitter, in unit of ms
lost	packets loss rate (0–255) of the field device video stream, loss percent = lost/255*100%

Remote Device Configuration Description

Last updated：2025-02-11 16:51:10

Sample JSON Configuration File

```
Configuring File Overview
File Name: config.json
File Location: ${workspace}/config.json
File Type: json
Note: Remove comments after "//". Minimum node as follows
{
    "device_id":"dev1",  //change to the remote device ID created in the console
    "device_name":"vin99",
    "max_streams":8,
    "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
}
```

Meaning of JSON Configuration Node

Node Name	Type	Meaning	Default Value	Remarks	Parameter Range
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	-
max_streams	**Number**	maximum	-	Required	None,

ms		number of streams connected to remote devices			recommended demo support >=8
projectid	**String**	Project ID	-	Required for public cloud	-
cloud_mode	**String**	Cloud mode	-	Required	• 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	-
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, removing the previous device
server_ip	**String**	TRRO server-side IP	-	Not required for public cloud Required for privatization	-
server_port	**Number**	TRRO server port number	-	Not required for public cloud	-

				Required for privatization	
log_enable	**Boolean**	Log switch	1	Optional	• 0: disabled • 1: info log • 2: info + debug log
sdk_mode	**String**	Stream transmission mode	"server2"	Optional	Public cloud requires server2
rtc_server_ip	**String**	Media server IP	–	No need to specify for public cloud, required for privatization	–
rtc_server_port	**Number**	Media server port number	–	No need to specify for public cloud, required for privatization	–
audio_enable	Number	Audio capture enabled	0	Optional	• 0: disabled • 1: Enabled
audio_receive	Number	Audio reception enabled	1	Optional	• 0: disabled • 1: Enabled
auto_permission	Number	Automatic request for permissions	1	Optional	• 0: Application request for permissions • 1: Automatic request for permissions
play_mode	Number	Viewing mode	0	Optional	• 0: Ultra low-latency viewing • 1: Smooth viewing
decode_h	Number	Whether	0	Optional	• 0: Software

w_acc		hardware decoding is prioritized			decoding • 1: Hardware decoding prioritized
file_path	**String**	Save file path	–	Optional, enable recording during configuration, ensure the configuration path exists	–
min_port	**Number**	local port range minimum value for video transmission	50000	Optional	–
max_port	**Number**	local port range maximum value for video transmission	50100	Optional	–

Error Codes

Last updated：2025-02-11 16:51:29

The definition of error codes is located in the trro_remote.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
Common	TRRO_SUCCED	0x00000001	1	Successful Return
	TRRO_COMMON_ERROR	0x00FFFFFF	16777215	Parameter Input or Initialization State Abnormal
	TRRO_GWID_NOT_FOUND	0x00000010	15728640	Non-existent Gateway ID
	TRRO_PARAM_INVALID	0x00F00000	16777215	Invalid Parameter
Configuration	TRRO_CONFIG_ERROR	0x01FFFFFF	33554431	General Configuration Error
	TRRO_CONFIG_PARSE_FAILED	0x01000002	16777218	Format parsing exception in configuration file. It is recommended to use an online JSON validation tool for inspection.
	TRRO_CONFIG_ILLEGAL	0x01000003	16777219	The parameters in the configuration file are invalid. It is recommended to check if the minimum nodes are covered.
	TRRO_CONFIG_UNEXIST	0x01000004	16777220	Configuration file does not exist
	TRRO_CONFIG_CER_FAILED	0x01000005	16777221	Certificate error in public cloud model
Initialization	TRRO_INIT_ERROR	0x02FFFFFF	50331647	General initialization failed for unknown reason

	TRRO_INIT_INPUT_ILLEGAL	0x02000002	33554434	The input parameter is illegal, please check if the input is normal.
	TRRO_INIT_PARSE_fAILED	0x02000003	33554435	Failed to parse initialization parameters; check if the minimum node in the configuration file is normal
	TRRO_INIT_CREAT_MEDIAMODE_FAILED	0x02000004	33554436	Media module creation failed
	TRRO_INIT_LICENSE_CHECK_FAILED	0x02000005	33554437	LICENSE verification failed
	TRRO_INIT_INVALUDE_INPUT	0x02000006	33554438	API call input parameters are invalid
	TRRO_INIT_MQTT_ABNORMAL	0x02000007	33554439	MQTT initialization exception
	TRRO_INIT_REPEAT	0x02000008	33554440	Duplicate initialization
Signaling	TRRO_SIGNAL_ERROR	0x03FFFFFF	67108863	Signaling Module Exception
	TRRO_SIGNAL_REGIST_FAILED	0x03000002	50331650	Signaling Registration Failed
	TRRO_SIGNAL_STATUS_ABNORMAL	0x03000003	50331651	Connection to MQTT Signaling Service Exception
	TRRO_SIGNAL_MESSAGE_FAILED	0x03000004	50331652	Failed to send the message
	TRRO_SIGNAL_CONNECT_OUTTIME	0x03000005	50331653	Connection to MQTT service timed out
Connection	TRRO_CONNECT_ERROR	0x05FFFFFF	100663295	Connection abnormal
	TRRO_CONNECT_OUTTIME	0x05000002	83886082	Timed out and disconnected
	TRRO_CONNECT_RESET	0x05000003	83886083	When connecting, if there are previous connections that

				have not been disconnected, a reset is required
Diagnostics	TRRO_DIAG_STARTED	0x06000000	100663296	Diagnostics Tool is starting
	TRRO_DIAG_TIMEOUT	0x06000001	100663297	Diagnostics Tool timed out in obtaining streaming information
	TRRO_DIAG_FAILED	0x06000002	100663298	Diagnostics Tool failed in processing
Permission	TRRO_UNAUTHORIZED	0x07000000	117440512	Gateway unauthorized, unable to send messages
Message	TRRO_MESSAGE_BYTE_EXCEED	0x08000002	134217730	Sending bytes exceeded the limit