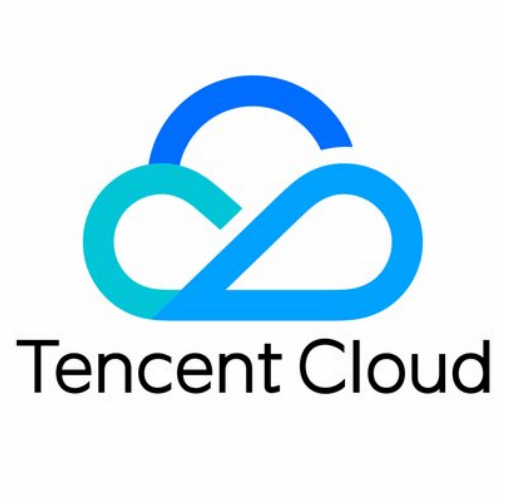
Real-time Teleoperation

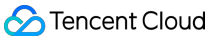
Development Guide

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

Development Guide
Multi-Path Network Transmission

Last updated：2025-02-11 16:46:31

> **Note:**
> Multi-path network transmission can support the transmission of audio and video streaming and control data flow using multiple network paths. When using it, ensure that the device's network environment meets the requirements, and configure the SDK to bind multiple network cards.

SDK Multi-network Configuration

During SDK initialization, use multiple network binding configuration. By binding multiple network card IPs, it can enable the multi-path network transmission feature

- Network Card IP Binding Configuration

```
"network_bind":["IP1","IP2"]
```

If using the web tool to generate the device configuration, in advanced settings, check the specified IP binding network card configuration, enter multiple network card IPs separated by semicolons.

- Video Multi-network Transmission Mode (Optional)

```
"media_trans_mode":1
```

1 suitable for multiple networks with acceptable bandwidth; 2 for Balanced Mode; 3 for multiple networks with very low bandwidth.

> **Note:**
> When running the application with the SDK in multi-network mode, sudo permissions are required to ensure successful network card binding. If granting sudo permissions is inconvenient, you can grant network card binding permissions to the application separately by executing the following in the application directory:
> `sudo setcap cap_net_raw,cap_net_bind_service+ep ./application_name`

Multi-network Environment Verification

Before using the SDK for multi-network transmission, it is necessary to verify if the device's multi-network environment meets the SDK requirements. For verification, you can use the curl command on the device to

access a website URL that returns the client's public IP address (using http://cip.cc as an example below), to verify whether the public IP addresses of the multiple network card are different or not.

```
curl http://43.130.151.72:5100/ip --interface local_network_card_IP1
curl http://43.130.151.72:5100/ip --interface local_network_card_IP2
```

If **curl requests from different network card IPs can stably return different public IPs** and match the corresponding network ISP circuits, the verification is successful.

Device Routing and IP Configuration (for reference)

The following is an example of the network deployment scheme that meets the requirements of multi-network environment. Note that it is not a mandatory recommendation. Any deployment scheme that can pass the verification is OK.

CPE Configuration Requirements

Single Multi-Network CPE

The same CPE accesses different ISP WAN networks. It creates multiple different IP range LAN gateways pointing to different WAN networks through VLAN+route configuration. For example, the gateway for the 192.168.67.1 IP range points to the WAN1 network (A ISP), and the gateway for the 192.168.8.1 IP range points to the WAN2 network (B ISP).

Multiple Single-Network CPE

Different CPEs access different ISP WAN networks. Set the LAN IP ranges of multiple CPEs to distinct IP ranges. For example, CPE1 uses the 192.168.8.1/24 IP range, and CPE2 uses the 192.168.67.1/24 IP range.

Device IP and Route Configuration

Single Network Card Connecting to CPE

The device uses 1 network port and network cable to connect to multiple LAN ports of the CPE through the ethernet switch.

- IP Settings

  Set multiple IP ranges and corresponding gateways for the network card. Below is an example of the Ubuntu system `/etc/network/interfaces.d` settings. For the eth0 network card, create eth0:0, assign IP 192.168.8.96 and gateway 192.168.8.1, and create eth0:1 assign IP 192.168.67.96 and gateway 192.168.67.1.

  ```
  auto eth0
  iface eth0 inet static
  ```

```
auto eth0:0
iface eth0:0 inet static
address 192.168.8.96
netmask 255.255.255.0
gateway 192.168.8.1

auto eth0:1
iface eth0:1 inet static
address 192.168.67.96
netmask 255.255.255.0
gateway 192.168.67.1
```

- Route Table Settings

  Create separate route tables for each network IP and specify the gateway for the route. Below is an example of different IP route settings for the eth0 network card:

```
ip rule add from 192.168.8.96 table 10
ip rule add from 192.168.67.96 table 20
ip route add default via 192.168.8.1 dev eth0 proto static metric 0
table 10
ip route add default via 192.168.67.1 dev eth0 proto static metric 0
table 20
```

Multiple Network Cards Connecting to CPE

The equipment uses multiple network ports and network cables to connect to multiple LAN IP range gateways of the CPE (can be through a switch).

- IP Settings

  Set the corresponding IP and gateway for each network card. The following is an example of settings for Ubuntu system /etc/network/interfaces.d, where IP 192.168.8.96 and gateway 192.168.8.1 are assigned to network card eth0, and IP 192.168.67.96 and gateway 192.168.67.1 are assigned to network card eth1.

```
auto eth0
iface eth0 inet static
address 192.168.8.96
netmask 255.255.255.0
gateway 192.168.8.1
```

```
auto eth1
iface eth1 inet static
address 192.168.67.96
netmask 255.255.255.0
gateway 192.168.67.1
```

- Route Table Settings

  Create an independent route table for each network IP and specify the gateway for the routes. The following is an example of different IP route settings for network cards eth0 and eth1:

```
ip rule add from 192.168.8.96 table 10
ip rule add from 192.168.67.96 table 20
ip route add default via 192.168.8.1 dev eth0 proto static metric 0
table 10
ip route add default via 192.168.67.1 dev eth1 proto static metric 0
table 20
```

Control data transfer

Last updated：2025-02-11 16:46:49

Note:
Control data transfer, used for binary data communication between the field device and the remote device. The SDK provides a data sending interface and a data receiving callback interface for both types of devices.

Note:
- Field device SDK, the data sending interface will send data in a broadcast manner to all remote devices that are in a session connection with the field device.
- Remote Device SDK, the data sending interface can send data to the specified field device within the session connection. Only when the remote device has master permission of the field device, the message can be successfully sent.

Field Device

Send data to remote devices

Note:
This interface will send data messages to all remote devices that are in session connection with the field device. The single message length limit is 700 bytes, and the sending frequency is limited to 100Hz.

```
extern "C" TRRO_EXPORT int TRRO_sendControlData(const char* msg, int
len, int qos = 0);
```

Parameter	Meaning
msg	Message content
len	Message length
qos	Send qos type. • 0: ultra-low latency transmission, data lost is allowed • 1: Reliable Transmission
Returned values	• Success: 1

	• Failed: <= 0

Register control message callback

```
extern "C" TRRO_EXPORT int TRRO_registerReportDataCallback(void*
context, TRRO_onReportData * callback);
```

Parameter	Meaning
context	Callback Context
TRRO_onControlData	Callback Function

```
typedef void TRRO_onReportData(void* context, const char* gwid, const
char* msg, int len, int qos)
```

Parameter	Meaning
context	Callback Context
gwid	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

Remote Device

Send data to field device

This interface will send data messages to the specified field device in the session connection. Sending will succeed only if the remote device has the master permission of the field device. The single message length limit is 700 bytes, and the sending frequency is limited to 100Hz.

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

Parameter	Meaning
gwid	On-site Equipment ID
msg	Send binary data
len	Message length
qos	Send qos: • 0: ultra-low latency transmission, data lost is allowed • 1: Reliable Transmission
Returned values	• Success: 1 • Failed: <= 0

Register control message callback

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

```
extern "C" TRRO_EXPORT int TRRO_registerReportDataCallback(void*
context, TRRO_onReportData * callback);
```

Parameter	Meaning
context	Context Pointer
callback	Callback Function

```
typedef void TRRO_onReportData(void* context, const char* gwid, const
char* msg, int len, int qos)
```

Parameter	Meaning
context	Callback Context
gwid	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

Operation Permission Management

Last updated：2025-02-11 16:47:06

> **Note:**
> Operation permission management is used to manage the permissions for remote devices to send control messages to field device. When multiple remote devices have permissions to control a certain field device, operation permission can be used to ensure that only one remote device can send control messages to the field device at a time, ensuring the security of operations.

Concepts

Permission Type

1. Guest Permission

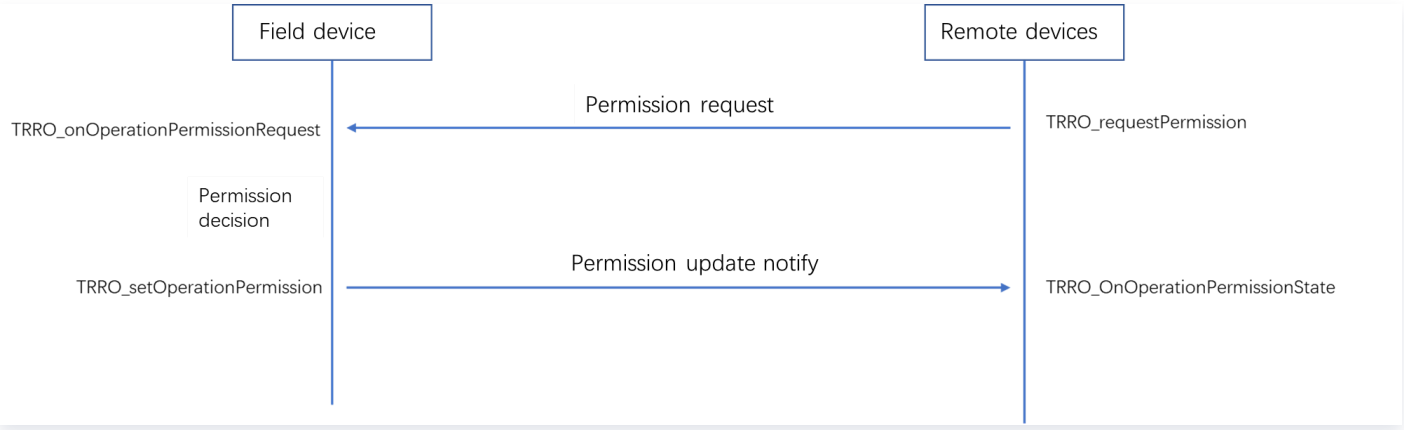
   The remote device acts as a spectator, it can only view audio and video content from the field device and receive reported data from it, but cannot send control messages to the field device or speak to the field device.
2. Master Permission

   The remote device acts as a controller, it can both view audio and video content from the field device, receive reported data, and send control messages to the field device.

> **Note:**
> - Currently, the field device SDK restricts that only one remote device can be authorized as master permission at a time.
> - When the field device SDK authorizes a remote device as master permission, other connected remote devices will automatically convert to guest permission.

Basic Process

1. The remote device sends a permission request to the field device for permission. 0 is for guest permission, and 1 is for master permission.
2. The field device authorizes the remote device based on the callback of the remote device's permission request and sends an permission notification to the relevant remote devices.
3. The remote device can determine the current permission status based on the permission notification from the field device.



Authorization Mode

The remote device SDK and the field device SDK can configure whether the authorization mode is automatic mode respectively. The specific configuration JSON field is `auto_permission`. When set to 1, it is automatic authorization mode. When set to 0, it is application authorization mode. The default configuration is automatic authorization mode.

Automatic Authorization Mode

The application does not need to handle authorization. The SDK will automatically handle the authorization process based on the following default logic:

- When the remote device sets the authorization mode to automatic, it will automatically send a master permission request to the connected field device when initiating a connect request.
- When the field device sets the authorization mode to automatic, it will automatically handle the authorization for the remote device according to the requested permissions from the remote device upon receiving the permission request.

Application Authorization Mode

The SDK provides relevant authorization interfaces. The application needs to call these interfaces to handle the authorization process:

- The remote device SDK will not automatically send master permission requests. The application needs to explicitly call the permission request interface to request master permission from the field device.
- The field device SDK will not automatically handle permission requests. The application needs to explicitly call the set permission interface to authorize the remote device based on the permission request and the application-defined authorization rules.

Authorization Interface

Remote Device

Registers the permissions change notification

```
/*
*   Callback Registration Function
*   context Context pointer, this pointer will be returned during the
callback
*   callback Registered TRRO_OnOperationPermissionState Callback
Function
*/

extern "C" TRRO_EXPORT void
TRRO_registerOnOperationPermissionState(void* context,
TRRO_OnOperationPermissionState* callback);


/*
* #name :   TRRO_OnOperationPermissionState
* @brief :  Callback for permissions state notification
* @input :  context             Callback context pointer, returned when
the registration callback function is invoked
*           field_devid         the field device ID
*           self_permission     The current permissions type of this
remote device, refer to TrroPermission, 0 is guest, only view
permission, 1 is master, has full operation permission
*           master_devid        Remote device ID with master permissions
* @return : void
*/
typedef void STD_CALL TRRO_OnOperationPermissionState(void* context,
const char* field_devid, int self_permission, const char* master_devid);
```

Request for permissions of target field device

```
/*
* @name : TRRO_requestPermission
* @brief : Send permission request to the field device, the field device
will respond with TRRO_OnOperationPermissionState to update permissions
```

```
* @input : gwid          Target field device ID
*          permisson     Refer to the structure TrroPermission
* @return : successful 1; failed <= 0
*/
extern "C" TRRO_EXPORT int TRRO_requestPermission(const char* gwid, int
permisson);
```

Field Device

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              //Application context
* @param[in]   callback             Callback function
* @return 1 for success, other failed
*/
extern "C" TRRO_EXPORT int TRRO_registerOperationPermissionRequest(void
*context, TRRO_onOperationPermissionRequest *callback);
```

Parameter	Meaning
remote_devid	target remote device id
permission	TRRO_onOperationPermissionRequest Callback function

```
/**
* @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);
```

Parameter	Meaning
remote_devid	target remote device id
remote_devid	Remote device ID requesting permission
permission	Requested permission, refer to TrroPermission: • 0:guest (view only) • 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 Sets the control permission for remote devices. Currently,
only one remote device can have master permission at a time. If a remote
device already has master permission, calling this interface to set
master permission will automatically revoke the master permission from
the previous device and set it for 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);
```

Parameter	Meaning
remote_devid	target remote device id
permission	the permission to be set

Voice Interaction

Last updated：2025-02-11 16:47:21

> **Note:**
> Voice interaction, used for voice communication between remote and field devices. Audio link will be automatically connected when a control session is established. The SDK defaults to only enable the upstream audio (from field devices to remote devices). If you need to use bidirectional audio, please prepare the relevant audio hardware, add the necessary configuration, and restart the SDK.

Hardware Preparation

Device Side	Upstream audio: field device -> remote device	Downstream audio: remote device -> field device
Field Device	Microphone	Speaker
Remote Device	Speaker	Microphone

Modifying Software Configuration

Modify the configurations of field devices and remote devices, enable the audio switch, and set the audio device.

Enable Audio Switch

Enable Audio Reception by setting audio_receive to 1; Enable Audio Capture by setting audio_enable to 1.

Device Side	Upstream audio: field device -> remote device	Downstream audio: remote device -> field device
Field Device	"audio_enable":1	"audio_receive":1
Remote Device	"audio_receive":1	"audio_enable":1

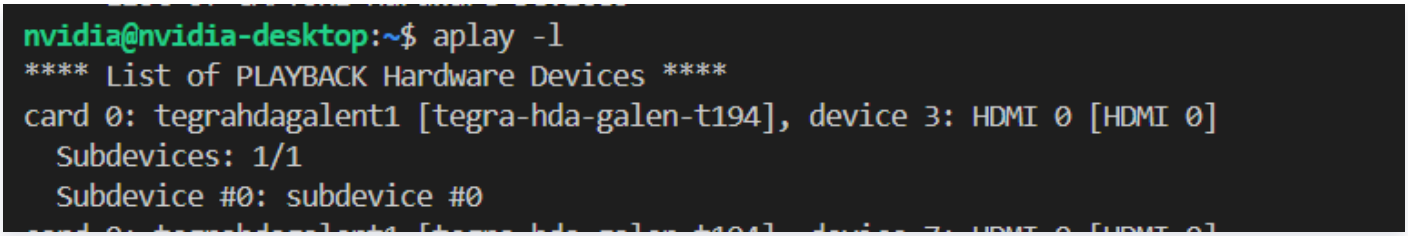
Use Configurations to Set Audio Devices

The SDK uses the system default audio playback and capture devices by default. If the target audio device is not the system default device on Linux, the corresponding audio device can be specified through configuration. After modifying the configuration, you need to restart the SDK.

Item	Configuration Instructions	Configuration Example

Playback	• `"audio_play":"hw:#Card,#Device,#SubDevice"` • Here, #Card, #Device, #SubDevice are the sound card device numbers. On Linux platforms, you can query the card, device, and subdevice numbers of the target device using `aplay -l`.	For example, the target device is Card 0, Device 0, SubDevice #0: `audio_play":"hw:0,0,0"`
Capture	• `"audio_record":"hw:#Card,#Device,#SubDevice"` • #Card, #Device, #SubDevice represent the sound card device numbers. On the Linux platform, you can use `arecord -l` to query the card, device, subdevice numbers of the target device.	For example, the target device is Card 0, Device 0, SubDevice #0: `audio_record":"hw:0,0,0"`

Example: Execute the command "aplay –l" to view audio playback devices.



If the playback device used is "card 0, device 3, subdevice 0" as shown, add the configuration node "audio_play":"hw:0,3,0" to the config.json file.

Use API to Set Audio Devices

You can use the sdk API to set the current audio device.
1) TRRO_getDeviceCount retrieves the number of audio devices of target type.
2) TRRO_getDeviceName retrieves the device name, and filters by name to find the audio device number of the target name
3) TRRO_setCurrentDevice sets the audio device with the corresponding device number as the current audio device

Interfaces	Description
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

Video Viewing and Stream Switching

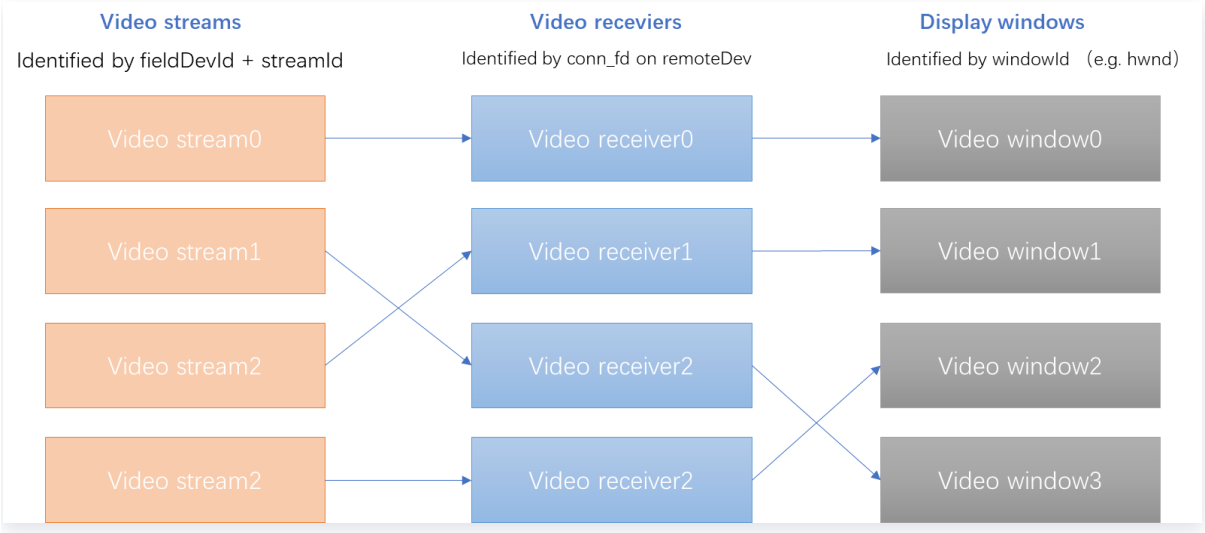
Last updated：2025-02-11 16:47:37

> **Note:**
> Video viewing and stream switching are used for the remote device to control the display of video streams of field device. Currently, it supports viewing and switching multiple video streams from multiple field devices. The remote device SDK initiates this without requiring any operations on the field device SDK.

Usage Introduction

Receiving Video Stream

When the remote device receives video streams, calling the TRRO_connect interface will receive the video stream through the video stream receiver and associate it with the video display window set through the TRRO_setWindows interface. The following diagram illustrates the association relationship between a set of video streams and display windows:



- **field device video stream:**Uniquely identified by fieldDeviceID + Stream ID. Stream ID numbers range from 0 to N, corresponding to the element numbers in the video stream array in the streams_config configuration.
- **Video Display Window:**Uniquely identified by the video window handle, such as HWND in win32. In internal rendering mode, the SDK can render and display the video on the specified window according to the video window handle. In external rendering mode, the developer maintains the video display window and video rendering display.
- **Remote Device Video Stream Receiver:**Uniquely identified by the receive handle (conn_fd). The receive handle numbers increment from 0 and can be customized by the developer, usually set to the window

number.

> **Note:**
> At the same time, a video stream receiver can only receive one video stream. When the video stream receiver connects to a new video stream, the previously connected video stream will automatically disconnect.

It can be seen that for internal rendering, the remote device video stream receiver bridges the field device video stream and the video window, allowing the SDK to automatically complete video stream display. For external rendering, the remote device video stream receiver handle assists the developer in identifying which receiver the video stream data is from and which windows should be rendered on.

Video Stream Switching

When the remote device switches video streams, it can be done through the TRRO_connect interface. Depending on whether the newly switched video stream needs to be displayed in a new window, there are two scenarios:

1. Switching video streams to display in an existing window:

Use the receive handle corresponding to the existing window, and receive the video stream to be switched to the window through the TRRO_connect interface. In this case, the new video stream will switch to the corresponding window, and the previous video stream on the window will be disconnected.

2. Switching video streams to display in a new window:

Use the receive handle corresponding to the new window, receive the switched video stream through the TRRO_connect interface, and disconnect the pre-switch video stream through the TRRO_disconnect interface.

Video Window Switching

In the internal rendering mode, when the remote device switches video windows, it can be done through the TRRO_setWindows interface. Generally, during the SDK startup phase, the receive handle of the video stream receiver will be associated with the video window. When window layout adjustments are needed (e.g., swapping two video streams' windows without switching the video streams themselves), the TRRO_setWindows interface can be used to update the corresponding relationship between the receive handle of the video stream receiver and the video window, completing the video stream window switch without disconnecting the video stream.

Relevant APIs

Connecting Video Stream

Instructions: This interface can be used to connect and switch the video streams of field device.

```
/*
* @name : TRRO_connect
* @brief : Initiates a video connection. Multiple calls can be made to
connect different streams. Asynchronous mode, confirms successful video
connection based on the onState status callback
* @input : gwid         ID of the target on-site equipment to connect
*          record_config:
*                       Default is to use global recording
configuration. When the default naming rules cannot meet the needs, use
a JSON string to configure each stream. e.g.: "{\"file_names\":
[{\"file\":\"test\", \"duration\":15}, {\"file\":\"test01\",
\"duration\":15}]}" ps: file - filename, duration - chunk duration in
seconds
* streams_num is the number of live device video streams to be pulled,
matching the length of the conn_fds and streams_num arrays, with values
ranging from 1 to the number of device_streams supported by the live
device
*          streams_id  Array of field device video stream IDs. The field
device video stream ID starts from 0, with a maximum value of the number
of device_streams supported by the on-site equipment - 1
* conn_fds is the array of conn_fd, where conn_fd represents the handle
identifier for receiving video streams. It is numbered by the user,
starting from 0, with the maximum value being max_streams - 1.
max_streams is configured in the remote device configuration file
* @return : 1 for success; <= 0 for failure
*/
int TRRO_connect(const char* gwid, const char* record_config, int
streams_num, int* streams_id, int* conn_fds);
```

Parameter	Meaning
gwid	field device ID
record_config	video recording configuration.json string, each video stream to be connected needs to be configured. It is recommended to use the global recording configuration by default, and use this method when it cannot be satisfied. `{ "file_names": [{`

	<pre>"file": "test",
	"duration": 15
	},
	{
	"file": "test01",
	"duration": 15
	}]
	}</pre>
	The parameters are described as follows:
	• file: File Name
	• duration: slice duration (unit: seconds)

streams_num	number of video streams to be connected
streams_id	Array of video stream IDs of field device to be connected
conn_fds	Array of handles used to receive video streams
Returned values	• Success 1 • Failed <= 0

Set video stream rendering window

Instructions: You can set the rendering window through this interface. It is used for internal rendering. If rendering externally, set it to a null pointer.

```
/*
* @name : TRRO_setWindows
* @brief : Set the display window handle corresponding to the receiving
stream handle
* @input : conn_fds Array of handles used to receive video streams
*         windows Array of display window handles. The display window
handle is the handle/pointer of the display form, such as HWND on the
Windows platform;
*                  When rendering externally, configure the display
window handles in the array as nullptr
*         num      Set the number of display windows to match the length
of conn_fds and Windows arrays
```

```
* @return :void
*/
void TRRO_setWindows(int* conn_fds, WindowIdType * windows, int num);
```

Parameter	Meaning
conn_fds	Array of handles used to receive video streams
windows	Array of display window handles, can be set to a null pointer array for external rendering
num	Consistent length with conn_fds and windows arrays

Close video connections

Instructions: This interface can be used to disconnect video connections.

Close specified video connection

```
/*
* @name : TRRO_disconnect
* @brief : Close the video connection corresponding to conn_fds
* @input : conn_fds Array of handles for the video connections to be
closed
*          fd_num Length of the conn_fd array
* @return : 1 for success; <= 0 for failure
*/
int TRRO_disconnect(int* conn_fds, int fd_num);
```

Parameter	Meaning
conn_fd	Array of video receiving handles
fd_num	Array length
Returned values	• Success 1 • Failed <= 0

Close all video connections

```
/*
* @name : TRRO_disconnectAll
```

```
* @brief : Close all video connections
* @input : void
* @return : 1 for success; <= 0 for failure
*/
int TRRO_disconnectAll();
```