NetSDK_Python

Programming Manual

V1.0.0

Foreword

Purpose

Welcome to use NetSDK (hereinafter referred to be "SDK") programming manual (hereinafter referred to be "the manual").

The manual describes the main function modules, interfaces and calling relationships, and provides example codes.

The example codes provided in the manual are only for demonstrating the procedure and not assured to copy for use.

Readers

- SDK software development engineers
- Project managers
- Product managers

Safety Instructions

The following categorized signal words with defined meaning might appear in the manual.

Signal Words	Meaning
 **DANGER**	Indicates a high potential hazard which, if not avoided, will result in death or serious injury.
 **WARNING**	Indicates a medium or low potential hazard which, if not avoided, could result in slight or moderate injury.
 **CAUTION**	Indicates a potential risk which, if not avoided, could result in property damage, data loss, lower performance, or unpredictable result.
 **TIPS**	Provides methods to help you solve a problem or save you time.
 **NOTE**	Provides additional information as the emphasis and supplement to the text.

Revision History

Version	Revision Content	Release Time
V1.0.0	First release.	May 2020

About the Manual

- The manual is for reference only. If there is inconsistency between the manual and the actual product, the actual product shall prevail.
- We are not liable for any loss caused by the operations that do not comply with the manual.
- The manual would be updated according to the latest laws and regulations of related jurisdictions. For detailed information, refer to the paper manual, CD-ROM, QR code or our official website. If there is inconsistency between paper manual and the electronic version, the electronic version shall prevail.
- All the designs and software are subject to change without prior written notice. The product updates might cause some differences between the actual product and the manual. Please contact the customer service for the latest program and supplementary documentation.
- There still might be deviation in technical data, functions and operations description, or errors in print. If there is any doubt or dispute, we reserve the right of final explanation.
- Upgrade the reader software or try other mainstream reader software if the manual (in PDF format) cannot be opened.
- All trademarks, registered trademarks and the company names in the manual are the properties of their respective owners.
- Please visit our website, contact the supplier or customer service if there is any problem occurring when using the device.

Glossary

This chapter provides the definitions to some of the terms that appear in the manual to help you understand the function of each module.

Term	Definition
Main Stream	A type of video stream that usually has better resolution and clarity and provides a better experience if the network resource is not restricted.
Sub Stream	A type of video stream that usually has lower resolution and clarity than the main stream but demands less network resources. The user can choose the stream type according to the particular scenes.
Resolution	Resolution is consisted of display resolution and image resolution. Display resolution refers to the quantity of pixels in unit area, and the image resolution refers to information quantity (the quantity of pixels per inch) stored in the image.
Video Channel	An abstract concept of the communication and video stream transmission between NetSDK and devices. For example, if a number of cameras (SD, IPC) are mounted on a storage device (NVR), the storage device manages the cameras as video channels which are numbered from 0. If NetSDK connects to the camera directly, the video channel is usually numbered as 0.
Motion Detection Alarm	When detecting a moving object on the image, an motion detection alarm will be uploaded.

Table of Contents

1 Overview

1.1 General

The following are the main functions:

Device login, real-time monitoring, record playback, record download, remote snapshot, alarm upload, device search, intelligent event upload and snapshot, device restart, device timing and more.

Table 1-1 Files of NetSDK library

Library Type	Library File Name	Library File Description
Function library	dhnetsdk.dll	Library file
	avnetsdk.dll	Library file
Configuration library	dhconfigsdk.dll	Library file
Auxiliary library of playing (coding and decoding)	dhplay.dll	Dahua playing library
	fisheye.dll	Fisheye correction library
Dependent library of "avnetsdk.dll"	Infra.dll	Base library
	json.dll	Json library
	NetFramework.dll	Network base library
	Stream.dll	Media transmission structure package library
	StreamSvr.dll	Stream service
Auxiliary library of " dhnetsdk "	IvsDrawer.dll	Image display library

Table 1-2 Files of package project

File Name	File Description
NetSDK.py	Call NetSDK library to package the interfaces as Python interfaces which can be used by users.
SDK_Callback.py	Store the callbacks used by the NetSDK library.
SDK_Enum.py	Store the enumerations used by the NetSDK library.
SDK_Struct.py	Store the structures used by the NetSDK library.

- The function library and configuration library are necessary libraries.
- The function library is the main body of SDK, which is used for interaction between client and products, remotely controls device, queries device data, configures device data information, and gets and handles the streams.
- NetSDK library is the base of the Python package project. In project, file NetSDK.py file defines the reference path of the NetSDK library, and you need to put the NetSDK library under the corresponding path when using it. Users can customize the reference path.
- All the externally used interfaces are defined in the NetClient class. Before using, you need to define an object of the NetClient class, and then call the interfaces in the class by the object.

- This manual mainly introduces the Python project which is used to encapsulate the interfaces of the C library. For more details, see the manuals of C NetSDK library: https://www.dahuasecurity.com/support/downloadCenter/softwares?id=2&child=3.

1.2 Applicability

- Recommended memory: No less than 512 M
- Python version: 3.7 version and later
- Operating system: Windows 10, Windows 8.1, Windows Vista, Windows 7 and Windows Server 2008/2003

1.3 Project Configuration

1.3.1 Related Software of Demo

The related software includes python3.7, pyqt5, pyqt5Designer and pycharm. Here takes the installation steps of Win64 as an example.

The storage directory of plug-in is D:\win64, which can be customized.

Step 1 Install python3.7.

Step 2 Open cmd, select the file folder named Scripts in the installation directory of python3.7, and then run the following commands to install plug-in.

```
pip install D:\win64\python_dotenv-0.10.1-py2.py3-none-any.whl
pip install D:\win64\Click-7.0-py2.py3-none-any.whl
pip install D:\win64\PyQt5_sip-4.19.13-cp37-none-win_amd64.whl
pip install D:\win64\PyQt5-5.11.3-5.11.2-cp35.cp36.cp37.cp38-none-win_amd64.whl
pip install D:\win64\pyqt5_tools-5.11.3.1.4-cp37-none-win_amd64.whl
pip install D:\win64\PyQt5Designer-5.10.1-cp37-none-win_amd64.whl
```

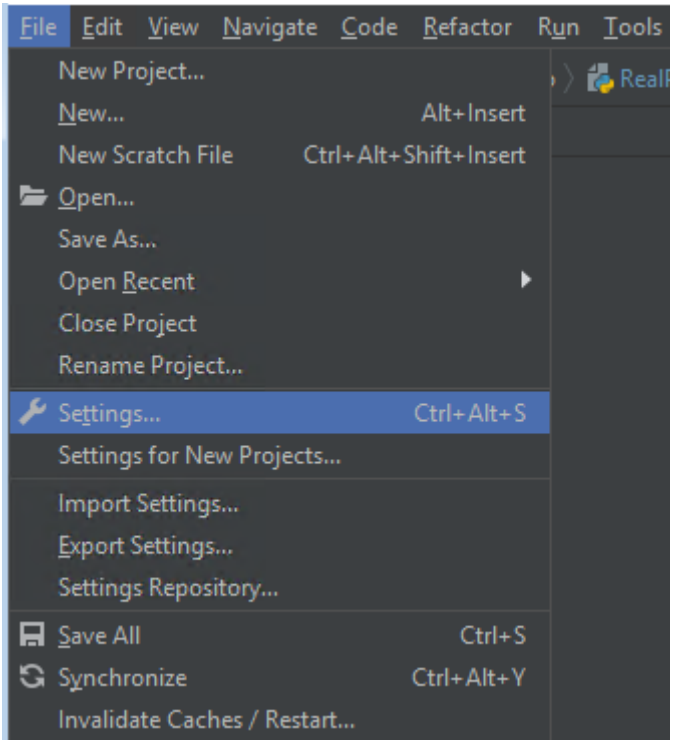
1.3.2 Pycharm Configuration

Configure Interpreter, and then run the Demo project by pycharm.

Step 1 Open pycharm.

Step 2 Select **File > Settings**.

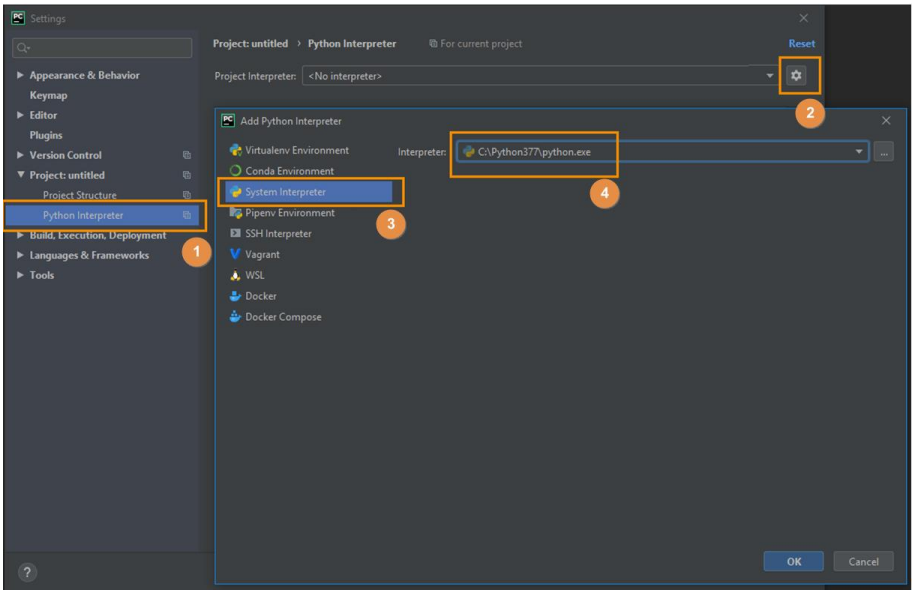
Figure 1-1 Select settings



Step 3 Configure Interpreter.

Information about PyQt5 related software is displayed.

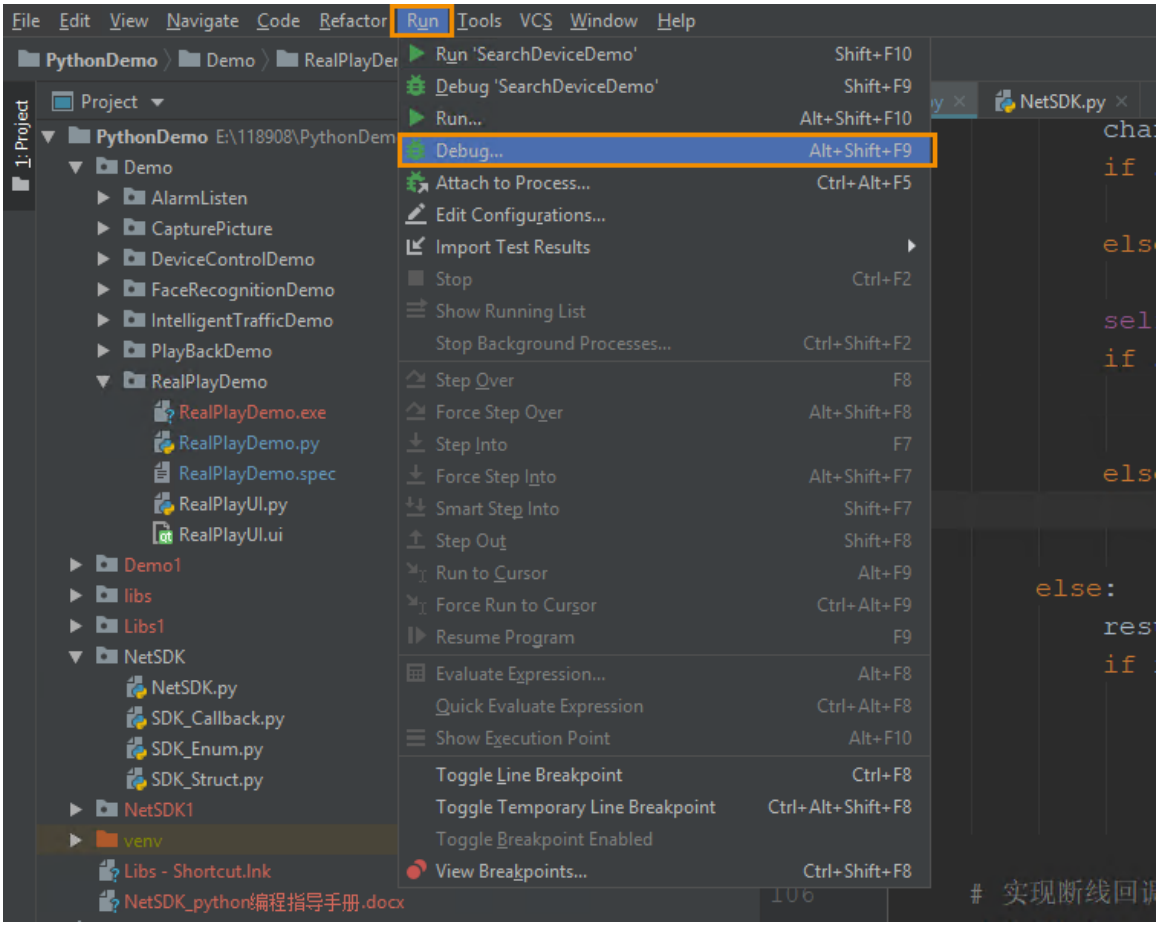
Figure 1-2 Configure interpreter



Step 4 Configure Demo.

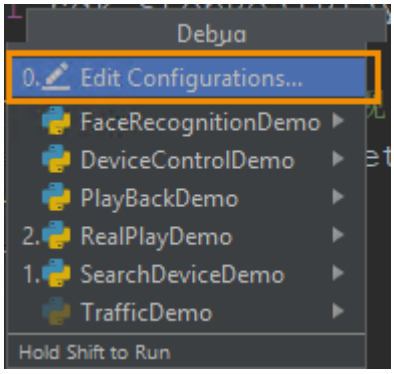
1) Select **Run > Debug**.

Figure 1-3 Configure Demo (1)



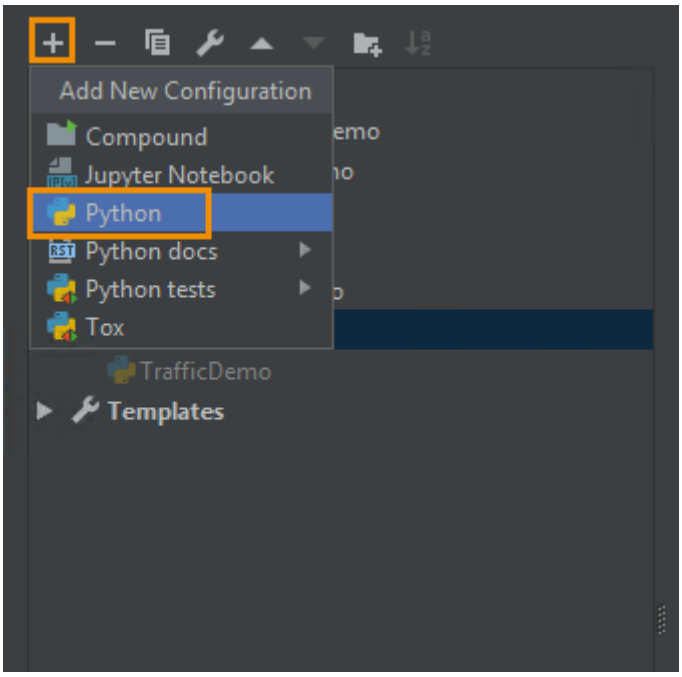
2) Select **Edit Configurations**.

Figure 1-4 Configure Demo (2)



3) Select **+ > Python**.

Figure 1-5 Run Demo configuration (3)

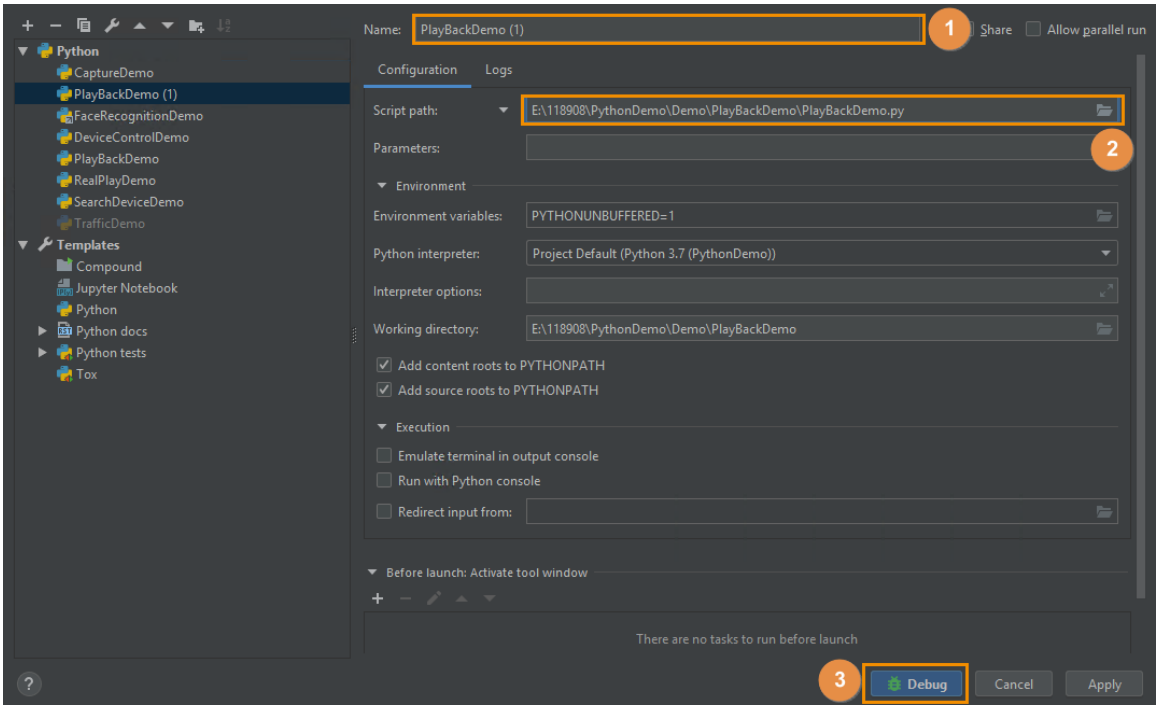


4) Set Demo configuration name and path of Demo.py.

- Name: Set Demo configuration name.
- Script path: Select path of Demo.py. Here takes PlayBackDemo.py as an example.

5) Click **Debug** to run Demo.

Figure 1-6 Run Demo configuration (4)



1.3.3 Adding Tool to Pycharm

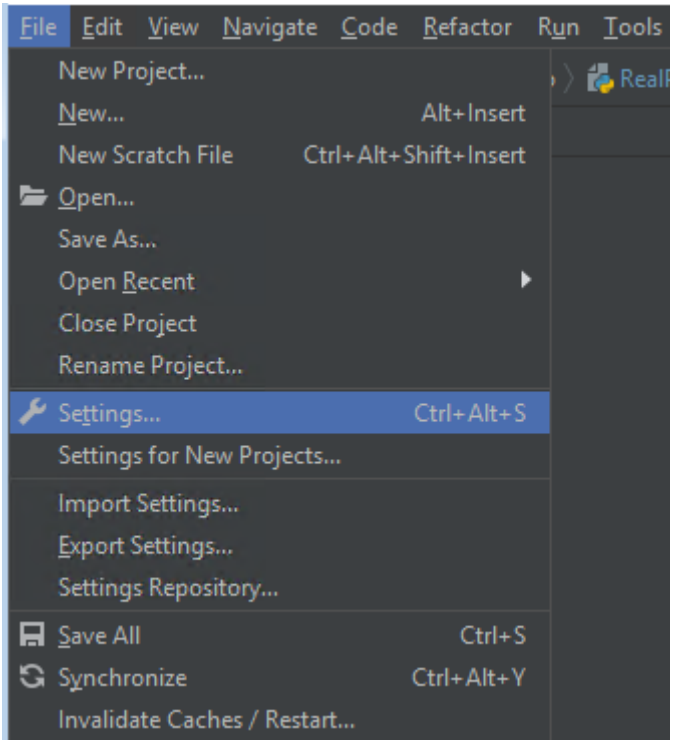
Add pyqt5designer and pyuic5 to pycharm.

- After adding pyqt5designer to pycharm, select the corresponding ui file and open qt designer. Use the tool to design UI.

- After adding pyuic5 to pycharm, select the corresponding .ui file and create .py file. View the defined variables through the py file.

Step 1 Select **File > Settings**.

Figure 1-7 Select external tools



Step 2 Add pyqt5designer. Select **Tool > External Tools**, and click + to configure parameters.

Figure 1-8 Add pyqt5designer

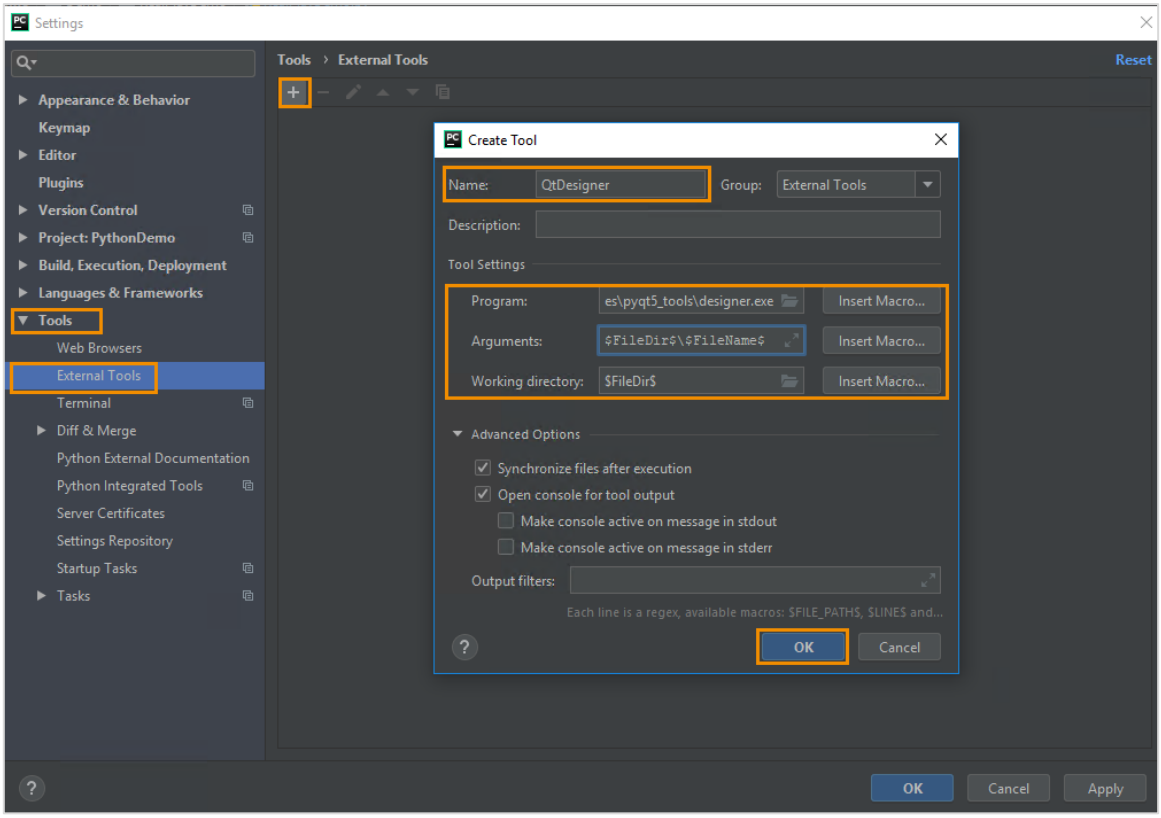


Table 1-3 Parameters of pyqt5designer

Paramater	Description
Name	Tool name which can be customized by users, such as QtDesigner.
Program	Enter the path of pyqt5designer.exe which is in the file folder of Scripts.
Arguments	$FileDir$\$FileName$
Working directory	$FileDir$

Step 3 Add pyuic5. Click + to configure parameters, and then click **OK**.

Table 1-4 Add pyuic5

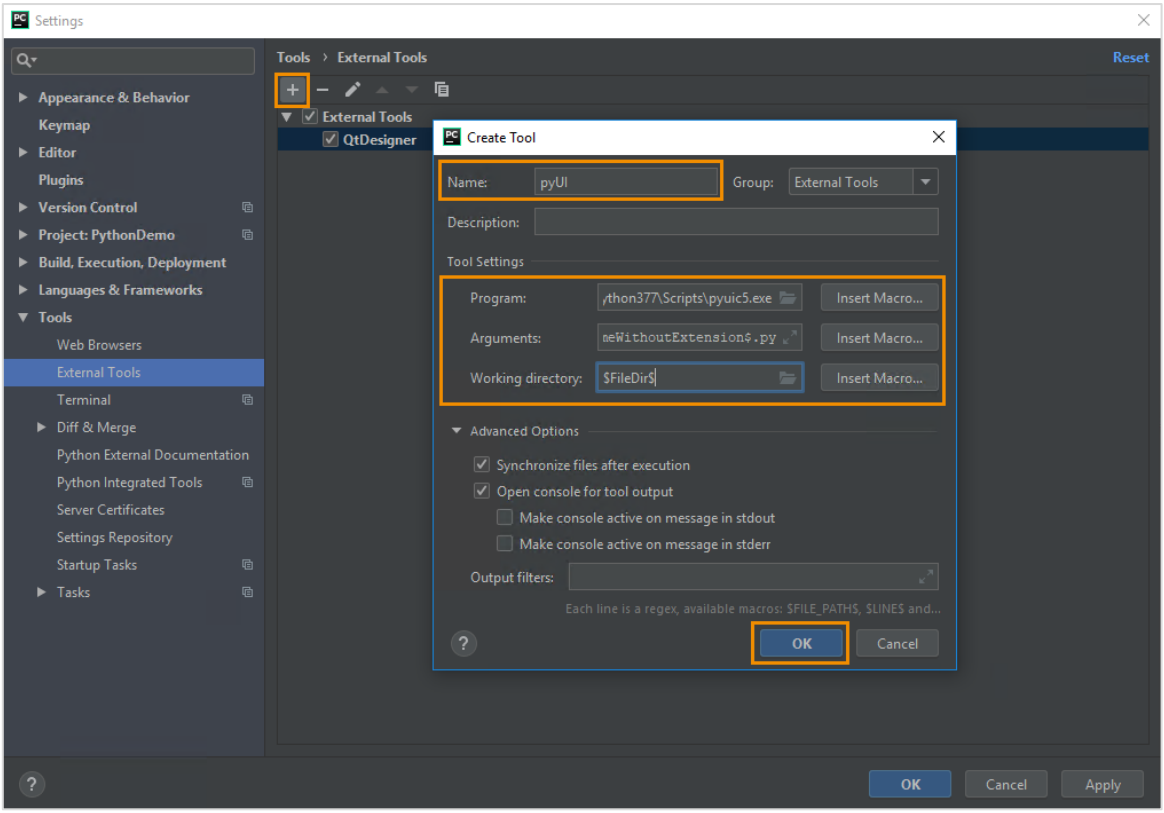


Table 1-5 Parameters of pyuic5

Paramater	Description
Name	Tool name which can be customized by users, such as PyUI.
Program	Enter the path of pyuic5.exe which is in the file folder of Scripts.
Arguments	$FileName$ -o $FileNameWithoutExtension$.py
Working directory	$FileDir$

Step 4 Use design interface of QtDesigner.

Select the corresponding .ui file, and right-click **External Tools** > **QtDesigner** to open QtDesigner.

Figure 1-9 Open QtDesigner

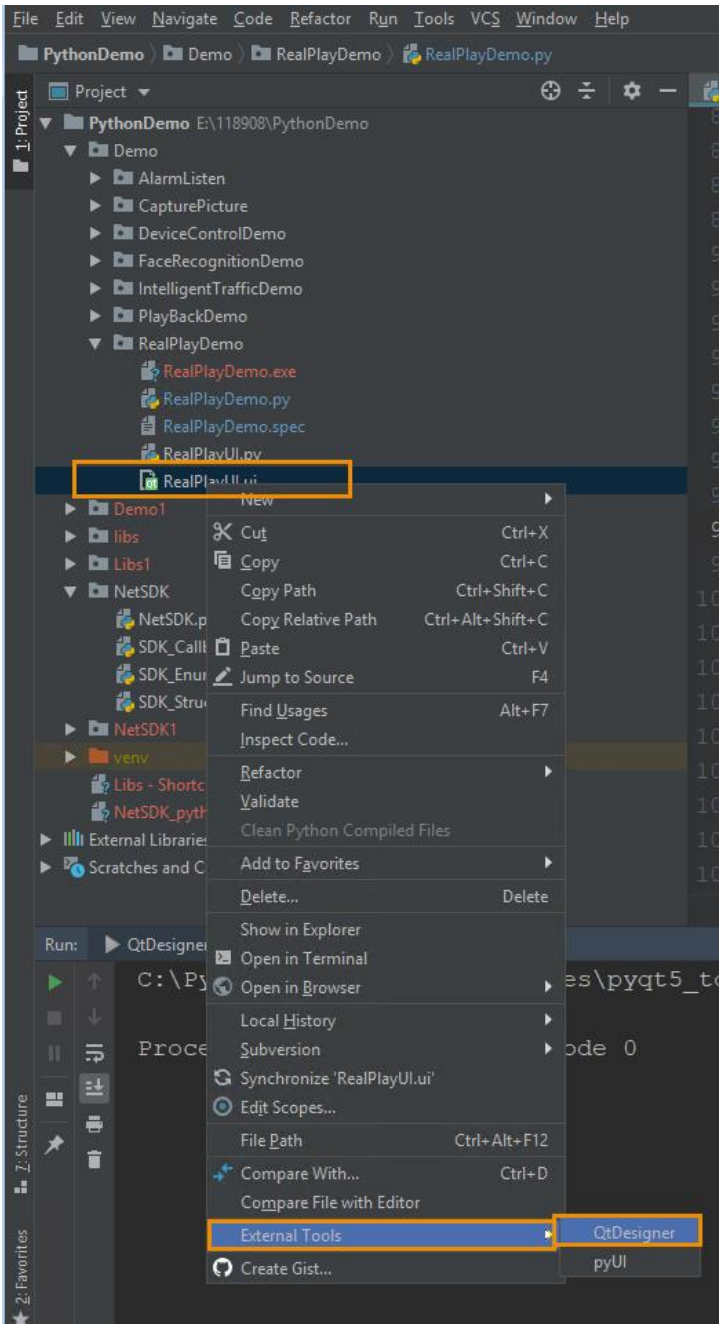
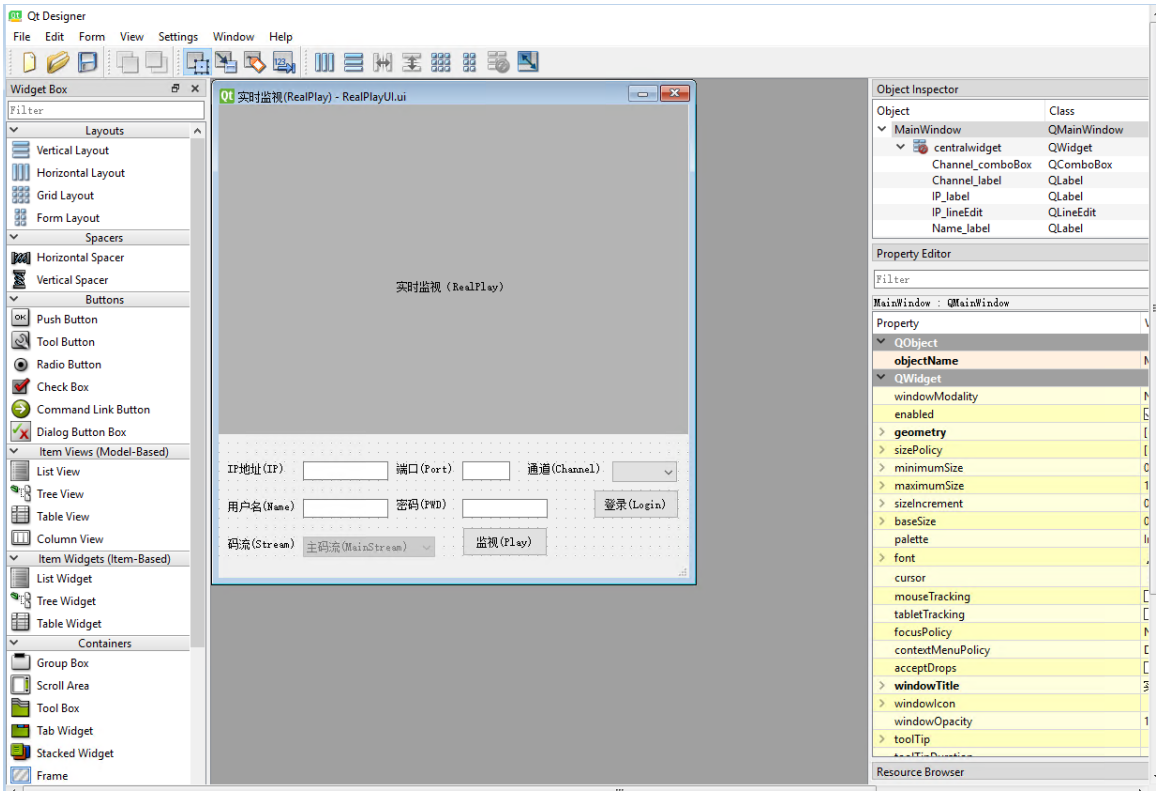


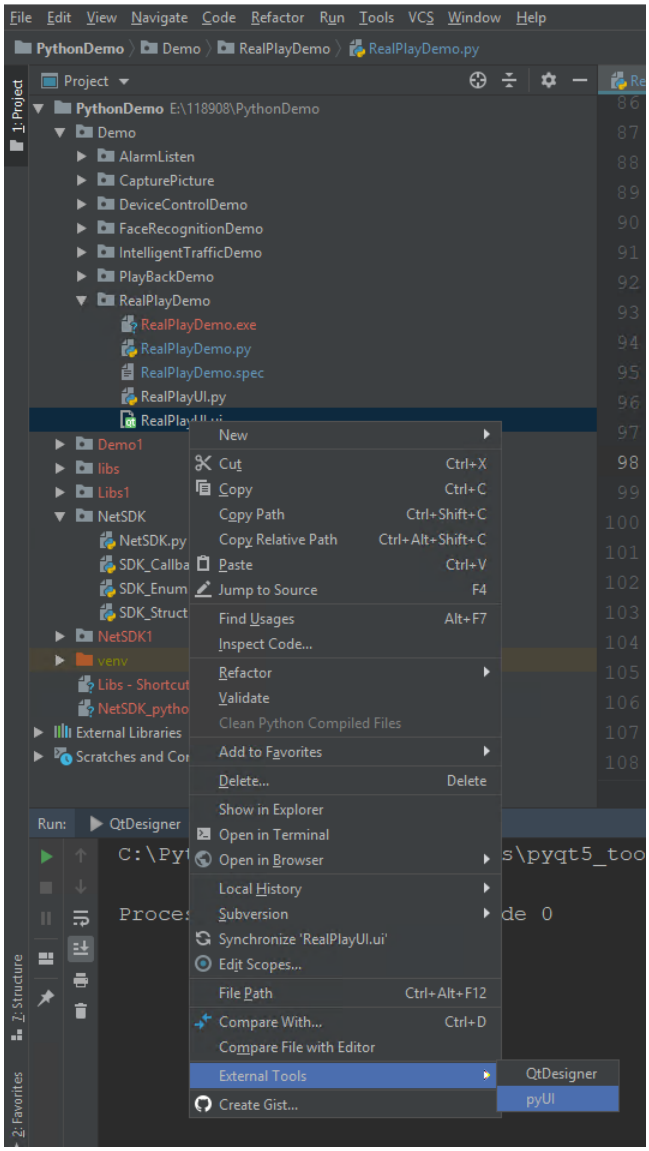
Figure 1-10 Design interface



Step 5 Transform file from .ui format to.py format.

Click the corresponding file in .ui format, right click to open menu, and select **External Tools** > **pyuic5** to transform file format.

Figure 1-11 Transform file format from .ui to .py



2 Function Modules

2.1 SDK Initialization

2.1.1 Introduction

Initialization is the first step of SDK to conduct all the function modules. It does not have the surveillance function but can set some parameters that affect the SDK overall functions.

- Initialization occupies some memory.
- Only the first initialization is valid within one process.
- After using this function, call cleanup interface to release SDK resource.
- The interfaces between **InitEx** and **Cleanup** are one-to-one corresponding. It is recommended to call it only once when writing codes.

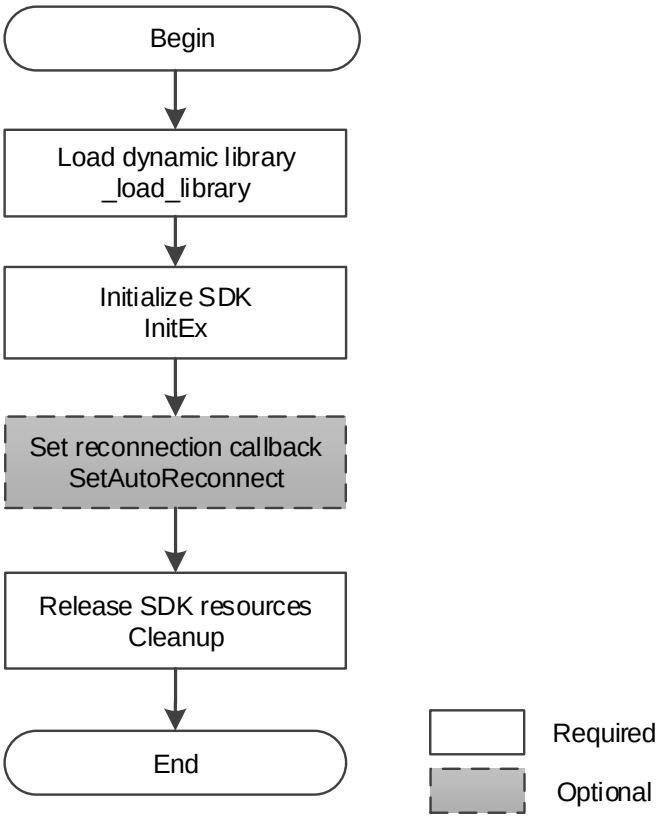
2.1.2 Interface Overview

Table 2-1 Interfaces of initialization

Interface	Implication
_load_library	Load dynamic library.
InitEx	Initialize SDK.
SetAutoReconnect	(Optional) Set reconnection callback.
Cleanup	Release SDK sources.

2.1.3 Process

Figure 2-1 Process of initialization



Process Description

Step 1 Call **_load_library** to load dynamic library.
Step 2 Call **InitEx** to initialize SDK and set disconnection callback.
Step 3 (Optional) Call **SetAutoReconnect** to set reconnection callback.
Step 4 Call **Cleanup** to release SDK resources.

Notes for Process

Call **InitEx** only once before using the SDK during the entire Demo running process. And call **Cleanup** once when all SDK-related functions finish, to release SDK resources. These two interfaces do not need to be called with every function.

2.1.4 Example Code

```
# Delegate and initialize the callback.
self.m_DisConnectCallBack = fDisConnect(self.DisConnectCallBack)
self.m_ReConnectCallBack = fHaveReConnect(self.ReConnectCallBack)
```

```
# Get the NetSDK object and Initialize.
self.sdk = NetClient()
self.sdk.InitEx(self.m_DisConnectCallBack)
self.sdk.SetAutoReconnect(self.m_ReConnectCallBack)

# Implement disconnection callback.
def DisConnectCallBack(self, lLoginID, pchDVRIP, nDVRPort, dwUser):
      self.setWindowTitle("Real-time monitoring (RealPlay)- Disconnection (OffLine)")

# Implement reconnection callback.
def ReConnectCallBack(self, lLoginID, pchDVRIP, nDVRPort, dwUser):
self.setWindowTitle(' Real-time monitoring (RealPlay)- Reconnection (OnLine)')

# Release SDK resources.
self.sdk.Cleanup()
```

2.1.5 Note

- _load_library is an internal callback of the NetClient which will be auto called when the NetClient class object is implemented. Here is just to remind users, if you need to change the location of NetSDK library, or to change the method and timing of calling NetSDK library, modify this function.
- Initialization: Call **InitEx** only once before using the SDK.
- Cleaning up: The interface **Cleanup** clears all the opened processes, such as login, real-time monitoring, and alarm subscription.
- Reconnection: NetSDK can set the reconnection function for the situations such as network disconnection and power off. NetSDK will keep logging until succeeded. Only the real-time monitoring, playback, smart event subscription and alarm subscription modules will be resumed after the connection is back.
- For callback details of example code, see "4 Callback Definition."

2.2 Device Search and Initialization

2.2.1 Introduction

Device search is mainly used to help user to get device info from network. Device search can work with login function. Device search interface can find relevant devices and login interface can login these devices.

Device search is classified into the following two types by whether crossing segment or not:

- Async same-segment device search: Search for device info within current segment.
- Sync cross-segment device search: According to user-set segment info, searching for device in corresponding segment.

2.2.2 Interface Overview

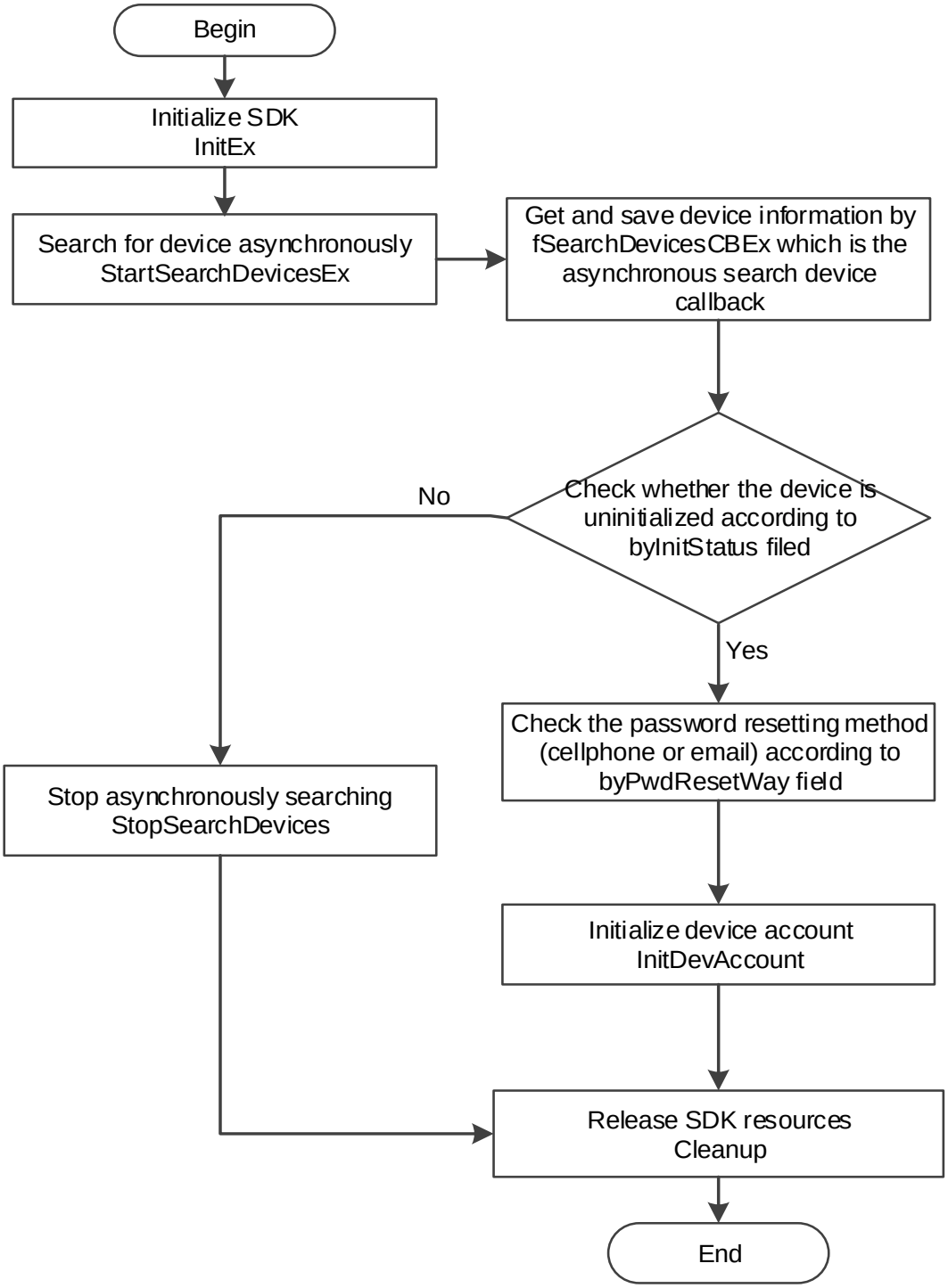
Table 2-2 Interface of device search and initialization

Interface	Implication
InitEx	Initialize SDK.
Cleanup	Clean up SDK.
StartSearchDevicesEx	Asynchronously search for devices within the same networksegment.
StopSearchDevices	Stop asynchronously searching for devices within the same networksegment.
SearchDevicesByIPs	Stop asynchronously searching for devices in cross-segment.
InitDevAccount	Initialize device.
GetLastError	Get error codes of interfaces that fail to be called.

2.2.3 Process

2.2.3.1 Async Searching within Same Segment

Figure 2-2 Process of async device searching and initialization



Process Description

Step 1 Call **InitEx** to initialize SDK.

Step 2 Call **StartSearchDevicesEx** to search for devices.

Step 3 Find the uninitialized devices by search callback **fSearchDevicesCBEx**. Check that the device is uninitialized according to byInitStatus filed. Check that the password can be reset by cellphone or email according to byPwdResetWay field which is also required in interface initialization.

Step 4 Call **InitDevAccount** to initialize device.

Step 5 Call **StopSearchDevices** to stop searching.

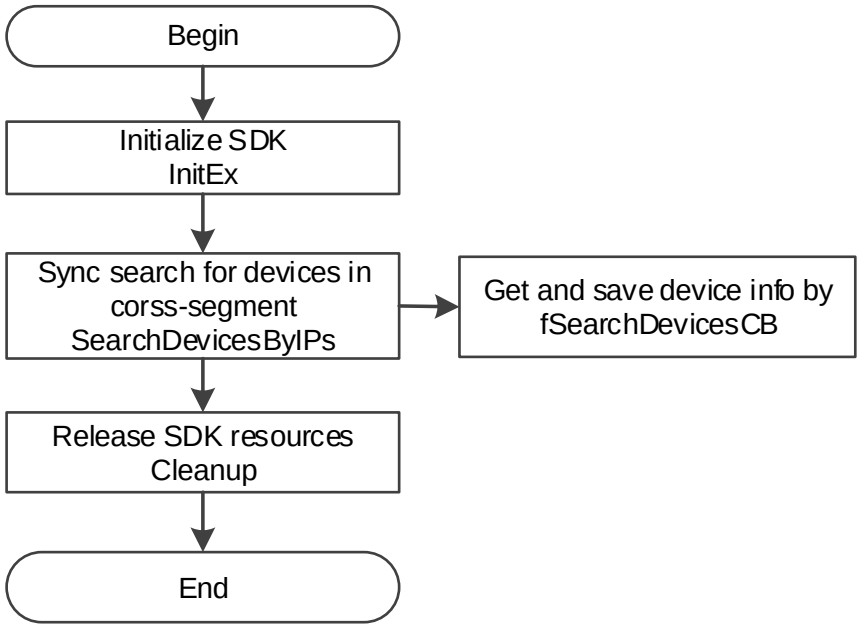
Step 6 Call **INetSDK.Cleanup** to release SDK resource.

Notes for Process

Call **InitEx** only once before using the SDK during the entire Demo running process. And call **Cleanup** once when all SDK-related functions finish, to release SDK resources. These two interfaces do not need to be called with every function.

2.2.3.2 Sync Searching in Cross-segment

Figure 2-3 Process of sync search and initialization



Process Description

Step 1 Call **InitEx** to initialize SDK.

Step 2 Call **SearchDevicesByIPs** to search for devices. Get device info by **fSearchDevicesCB**.

Step 3 Call **INetSDK.Cleanup** to release SDK resource.

Notes for Process

Call **InitEx** only once before using the SDK during the entire Demo running process. And call **Cleanup** once when all SDK-related functions finish, to release SDK resources. These two interfaces do not need to be called with every function.

2.2.4 Example Code

2.2.4.1 Async Searching within Same Segment and Device Initialization

Code path:

Demo\SearchDeviceDemo\ SearchDeviceDemo.py

```
# Multicast and broadcast search
def start_search_device(self):
    # Get local IP. Search in the consideration of the situation of multiple network cards.
    # The number of calls to the search interface is the same as the number of network cards.
    hostname = socket.gethostname()
    localIp = socket.gethostbyname_ex(hostname)
    IPList = self.getIPAddrs()
    for i in range(IPList.__len__()):
        startsearch_in = NET_IN_STARTSERACH_DEVICE()
        startsearch_in.dwSize = sizeof(NET_IN_STARTSERACH_DEVICE)
        startsearch_in.emSendType = EM_SEND_SEARCH_TYPE.MULTICAST_AND_BROADCAST
        startsearch_in.cbSearchDevices = search_device_callback
        startsearch_in.szLocalIp = IPList[i].encode()
        startsearch_out = NET_OUT_STARTSERACH_DEVICE()
        startsearch_out.dwSize = sizeof(NET_OUT_STARTSERACH_DEVICE)
        lSearchHandle = self.sdk.StartSearchDevicesEx(startsearch_in, startsearch_out)
        if lSearchHandle != 0:
            self.localIP_SearchHandle_dict[lSearchHandle] = startsearch_in.szLocalIp
            self.lSearchHandle_list.append(lSearchHandle)
    if(IPList.__len__() > 0):
        del IPList
    if(self.localIP_SearchHandle_dict.__len__() > 0):
        return True
    else:
        return False
# Stop searching. Use with start_search_device in pairs.
def stop_search_device(self):
    for i in range(self.lSearchHandle_list.__len__()):
        result = self.sdk.StopSearchDevices(self.lSearchHandle_list[i])
    self.lSearchHandle_list.clear()
    self.localIP_SearchHandle_dict.clear()
    self.row = 0
    self.column = 0
```

```
        self.device_info_list.clear()
        self.device_mac_list.clear()
        self.tableWidget.clear()
        self.tableWidget.setHorizontalHeaderLabels(['(No.)', '(Status)', '(IP Version)', '(IP Address)', '(Port)',
'(Subnet Mask)', '(Gateway)', '(Mac Address)', '(Device Type)', '(Detail Type)', 'Http(Http)'])
        return
def Init_Btn(self):
        # Get the IP and initialization info of selected rows.
        currentRow = self.tableWidget.currentRow()
        if((len(self.device_info_list) ==0)or((self.device_info_list[currentRow][0]&3) != 1)):
            QMessageBox.about(self, '(prompt)', '' (Please select not initialized device)'')
        else:
            result = self.init_device_accout(self.device_info_list[currentRow])
            if result == True:
                QMessageBox.about(self, '(prompt)', ''(Initialize Success)'')
                item = QTableWidgetItem(''(Initialize)'')
                self.device_info_list[currentRow][0] = 2
                self.tableWidget.setItem(currentRow, 1, item)
                self.tableWidget.update()
                self.tableWidget.viewport().update()
# Initialize device account
def init_device_accout(self, device_info:list):
        child = QDialog()
        child_ui = Ui_InitDevAccount()
        child_ui.setupUi(child)
        if (1 == (device_info[3] & 1)):
            # Phone
            child_ui.way_lineEdit.setText('(Phone)')
        elif (1 == (device_info[3] >> 1 & 1)):
            # Mail
            child_ui.way_lineEdit.setText('(Mail)')
        value = child.exec()
        if (value == 0):
            return False
        init_Account_In = NET_IN_INIT_DEVICE_ACCOUNT()
        init_Account_In.dwSize = sizeof(init_Account_In)
        init_Account_In.szMac = device_info[2]
        username = child_ui.username_lineEdit.text()
        password = child_ui.password_lineEdit.text()
        confirm_password = child_ui.confirm_password_lineEdit.text()
        if(password != confirm_password):
            QMessageBox.about(self, '(prompt)', ''(Confirm password is wrong, please input again)'')
            return
        init_Account_In.szUserName = username.encode()
        init_Account_In.szPwd = password.encode()
        init_Account_In.szCellPhone = child_ui.reset_way_lineEdit.text().encode()
        if (1 == (device_info[3] & 1)):
```

```
        # Phone
            init_Account_In.szCellPhone = child_ui.reset_way_lineEdit.text().encode()
    elif(1 == (device_info[3] >> 1 & 1)):
        # Mail
            init_Account_In.szMail = child_ui.reset_way_lineEdit.text().encode()
    init_Account_In.byPwdResetWay = device_info[3]
    init_Account_Out = NET_OUT_INIT_DEVICE_ACCOUNT()
    init_Account_Out.dwSize = sizeof(init_Account_Out)

    result = self.sdk.InitDevAccount(init_Account_In, init_Account_Out, 5000, self.localIP)
    if result:
        return True
    else:
        QMessageBox.about(self, '(prompt)', 'error:' + str(self.sdk.GetLastError()))
        return False
```

2.2.4.2 Sync Searching in Cross-segment

Code path:

Demo\SearchDeviceDemo\ SearchDeviceDemo.py

```
# Unicast search
def start_search_device_byIP(self, start_IP, end_IP): # Pay attention to the IP validity.
    self.localIP = None
    startsearchbyIp_in = DEVICE_IP_SEARCH_INFO()
    startsearchbyIp_in.dwSize = sizeof(DEVICE_IP_SEARCH_INFO)
    start = struct.unpack("!I", socket.inet_aton(start_IP))[0]    # Network byte order to host byte order.
    end = struct.unpack("!I", socket.inet_aton(end_IP))[0]
    if (end - start > 255):
        QMessageBox.about(self, '(prompt)', "(Number of IP addresses exceeds the upper limit 256.)")
        return False

    startsearchbyIp_in.nIpNum = end - start + 1

    for i in range(startsearchbyIp_in.nIpNum):
        ip = DEVICE_IP_SEARCH_INFO_IP()
        ip.IP = socket.inet_ntoa(struct.pack("!I", start + i)).encode()
        startsearchbyIp_in.szIP[i] = ip

    wait_time = int(wnd.Searchtime_lineEdit.text())
    result = self.sdk.SearchDevicesByIPs(startsearchbyIp_in, search_devie_byIp_callback, 0, None,
wait_time)
    if not result:
        QMessageBox.about(self, '(prompt)', "(Failed to search devices by SearchDevicesByIPs.)")
        return False
    return True
```

2.3 Device Login

2.3.1 Introduction

Device login, also called user authentication, is the precondition of all the other function modules.

You will obtain a unique login ID upon log in to the device and should introduce login ID before using other SDK interfaces. The login ID becomes invalid once logged out.

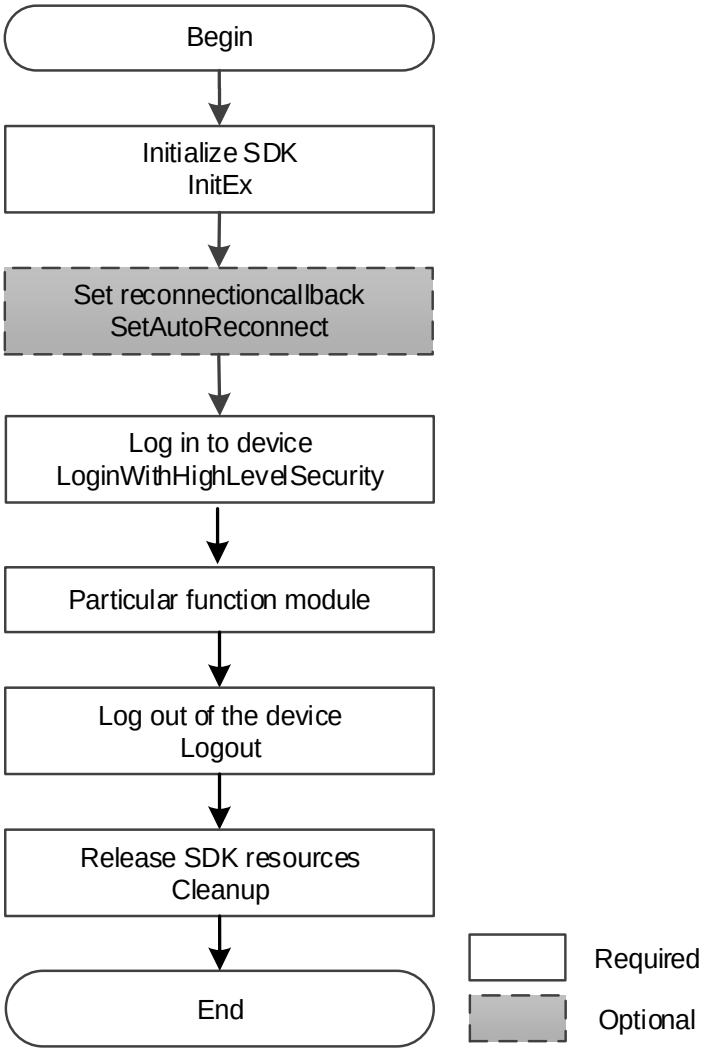
2.3.2 Interface Overview

Table 2-3 Interfaces of device login

Interface	Implication
InitEx	Initialize SDK.
SetAutoReconnect	Set reconnection callback.
Cleanup	Clean up SDK.
LoginWithHighLevelSecurity	Log in with high level security.
Logout	Log out.

2.3.3 Process

Figure 2-4 Process of login



Process Description

Step 1 Call **InitEx** to initialize SDK.
Step 2 Call **SetAutoReconnect** to set reconnection callback.
Step 3 Call **LoginWithHighLevelSecurity** to log in to the device.
Step 4 Implement the required function modules.
Step 5 Call **Logout** to log out of the device.
Step 6 Call **Cleanup** to release SDK resources.

Notes for Process

Call **InitEx** only once before using the SDK during the entire Demo running process. And call **Cleanup** once when all SDK-related functions finish, to release SDK resources. These two interfaces do not need to be called with every function.

2.3.4 Example Code

```
# Log in to device, and get login handle and device info. If login failed, display the error info.
stuInParam = NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY()
stuInParam.dwSize = sizeof(NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY)
stuInParam.szIP = ip.encode()
stuInParam.nPort = port
stuInParam.szUserName = username.encode()
stuInParam.szPassword = password.encode()
stuInParam.emSpecCap = EM_LOGIN_SPAC_CAP_TYPE.TCP
stuInParam.pCapParam = None

stuOutParam = NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY()
stuOutParam.dwSize = sizeof(NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY)

self.loginID, device_info, error_msg = self.sdk.LoginWithHighLevelSecurity(ip, port, username,
password)
stuInParam, stuOutParam)       if self.loginID != 0:
            for i in range(int(device_info.nChanNum)):
                self.Channel_comboBox.addItem(str(i))    # Display the device channels.
      else:
            QMessageBox.critical(self, '(prompt)', error_msg, QMessageBox.Ok, QMessageBox.No)    #
Display error info of login interface.

# Log out of the device.
result = self.sdk.Logout(self.loginID)
      if result:
            self.loginID = 0
```

2.3.5 Note

- Login handle: When the login is successful, the returned value is not 0 (even the handle is smaller than 0, the login is also successful). One device can login multiple times with different handles at each login. If there is no special function module, it is suggested to login only once. The login handle can be repeatedly used on other function modules.
- Duplicate handles: It is normal that the login handle is the same as the existed handle. For example, log in to device A and get handle loginIDA. However, if you log out of loginIDA and then log in, you may get LoginIDA again. But the duplicate handles do not occur throughout the lifetime of the handle.
- Logout: The interface will release the opened functions internally, but it is not suggested to rely on the cleaning up function of lougout. For example, if you opened the monitoring

function, you should call the interface that stops the monitoring function when it is no longer required.

- Use login and logout in pairs: The login consumes some memory and socket information and release sources once logout.
- Login failure: It is suggested to check the failure through return parameter error_msg. for more details, see the error code list in **LoginWithHighLevelSecurity**.
- After reconnection, the original login ID will be invalid. After the device is auto reconnected, the login ID will take effect again.

2.4 Real-time Monitoring

2.4.1 Introduction

Real-time monitoring obtains the real-time stream from the storage device or front-end device, which is an important part of the surveillance system.

SDK can get the main stream and sub stream from the device once it logged.

- Supports calling the window handle for SDK to directly decode and play the stream (Windows system only).
- Supports calling the real-time stream to you to perform independent treatment.
- Supports saving the real-time record to the specific file though saving the callback stream or calling the SDK interface.

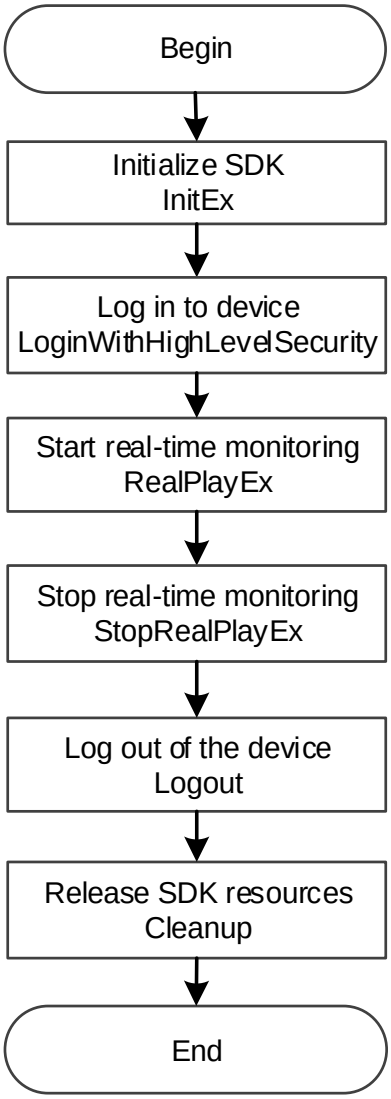
2.4.2 Interface Overview

Table 2-4 Interfaces of real-time monitoring

Interface	Implication
InitEx	Initialize SDK.
Cleanup	Clean up SDK.
LoginWithHighLevelSecurity	Log in with high level security.
Logout	Log out.
RealPlayEx	Start real-time monitoring extension interface.
StopRealPlayEx	Stop real-time monitoring extension interface.
GetLastError	Get error codes of interfaces that fail to be called.
GetLastErrorMessage	Get error info of interfaces that fail to be called.

2.4.3 Process

Figure 2-5 Process of real-time monitoring



Process Description

Step 1 Call **InitEx** to initialize SDK.
Step 2 Call **LoginWithHighLevelSecurity** to log in to the device.
Step 3 Call **RealPlayEx** to start real-time monitoring.
Step 4 Call **StopRealPlayEx** to stop real-time monitoring.
Step 5 Call **Logout** to log out of the device.
Step 6 Call **Cleanup** to release SDK resources.

Notes for Process

Call **InitEx** only once before using the SDK during the entire Demo running process. And call **Cleanup** once when all SDK-related functions finish, to release SDK resources. These two interfaces do not need to be called with every function.

2.4.4 Example Code

```
# Start real-time monitoring.
channel = self.Channel_comboBox.currentIndex()    # Channel No.
if self.StreamTyp_comboBox.currentIndex() == 0:
    stream_type = SDK_RealPlayType.Realplay # Main stream
else:
    stream_type = SDK_RealPlayType.Realplay_1 # Sub stream
self.playID = self.sdk.RealPlayEx(self.loginID, channel, self.PlayWnd.winId(), stream_type)
if self.playID != 0:
    self.play_btn.setText("(Stop)")
    self.StreamTyp_comboBox.setEnabled(False)
else:
QMessageBox.critical(self, '(prompt)', self.sdk.GetLastErrorMessage(), QMessageBox.No)

# Stop real-time monitoring.
result = self.sdk.StopRealPlayEx(self.playID)
if result:
    self.playID = 0
self.PlayWnd.repaint()
```

2.4.5 Notes for Process

- **GetLastError** is the interface used to get the error codes when failed to call NetSDK interfaces. **GetLastErrorMessage** is the interface to get error information.
- It is recommended to call **GetLastErrorMessage** to get error information to identify the cause of the error.

2.5 Record Playback

2.5.1 Introduction

Record playback function plays the videos of a particular period in some channels to find the target videos for check.

The playback includes the following functions: Start playback, pause Playback, resume playback, and stop playback.

2.5.2 Interface Overview

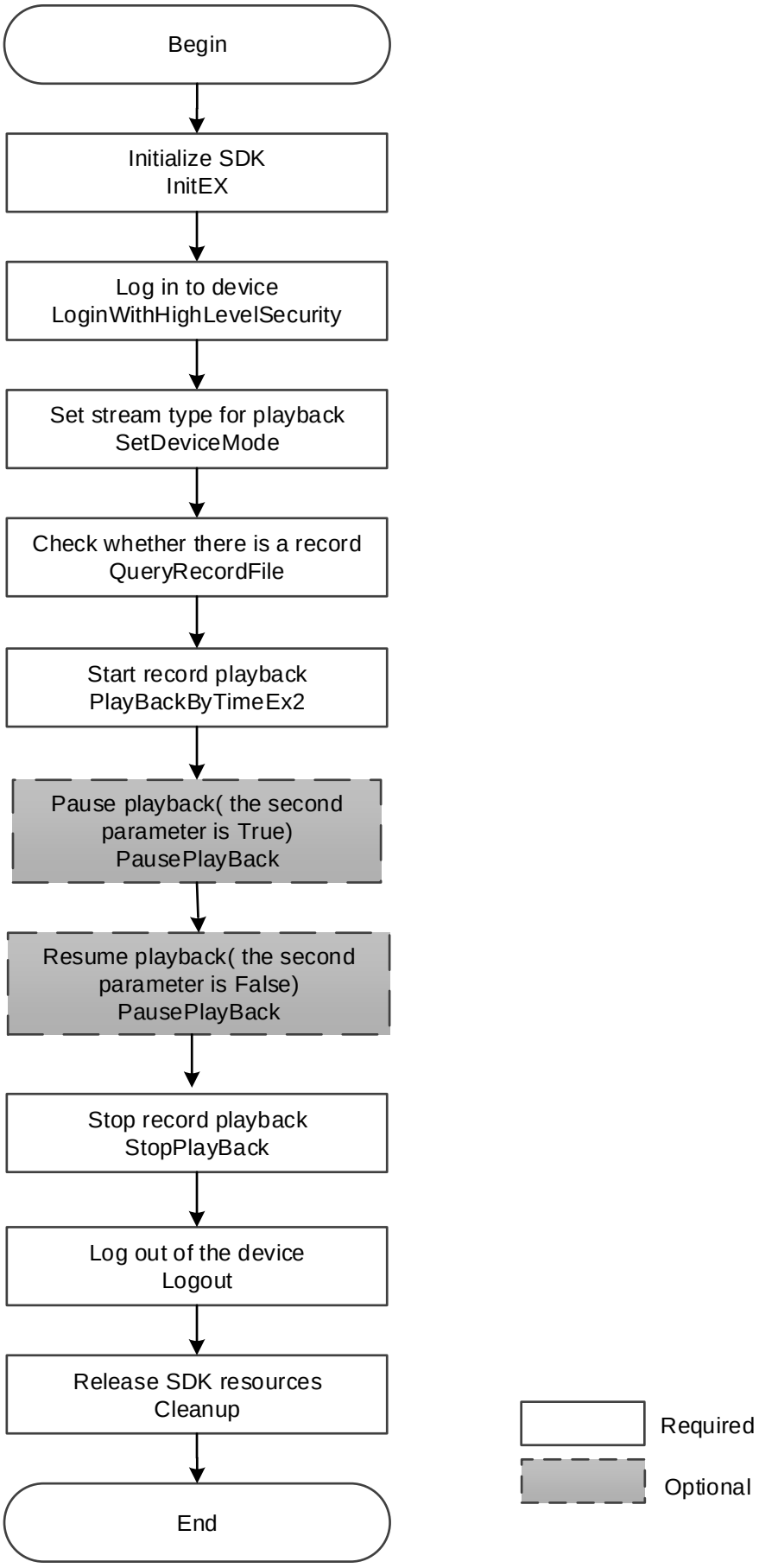
Table 2-5 Interfaces of record playback

Interface	Implication
InitEx	Initialize SDK.
Cleanup	Clean up SDK.
LoginWithHighLevelSecurity	Log in with high level security.
Logout	Log out.
PlayBackByTimeEx2	Extension interface of playback by time.
StopPlayBack	Stop playback.
PausePlayBack	Stop or resume palyback.
SetDeviceMode	Set device mode.
QueryRecordFile	Query for all the record files within a period.

2.5.3 Process

After SDK initialization, you need to input channel number, start time, stop time, and valid window handle to realize the playback of the required record.

Figure 2-6 Process of record playback



Process Description

Step 1 Call **InitEx** to initialize SDK.
Step 2 Call **LoginWithHighLevelSecurity** to log in to the device.
Step 3 Call **SetDeviceMode** to set the stream type.
Step 4 Call **QueryRecordFile** to check whether there is a record in the selected period.
Step 5 Call **PlayBackByTimeEx2** to start playback.
Step 6 (Optional) Call **PausePlayBack**. The playback will pause when the second parameter is True.
Step 7 (Optional) Call **PausePlayBack.** The playback will resume when the second parameter is False.
Step 8 Call **StopPlayBack** to stop playback.
Step 9 Call **Logout** to log out of the device.
Step 10 Call **Cleanup** to release SDK resources.

2.5.4 Example Code

```
# Set stream type of playback. Here set the type as main stream.
stream_type = c_int(0)
result = self.sdk.SetDeviceMode(self.loginID, int(EM_USEDEV_MODE.RECORD_STREAM_TYPE), stream_type)
if not result:
QMessageBox.critical(self, '(prompt)', self.sdk.GetLastErrorMessage(), QMessageBox.No)

# Query record files
result, fileCount, infos = self.sdk.QueryRecordFile(self.loginID, 0, int(EM_QUERY_RECORD_TYPE.ALL), startTime, endTime, None, 5000, False)

# Start playback
inParam = NET_IN_PLAY_BACK_BY_TIME_INFO()
inParam.hWnd = self.PlayBackWnd.winId()
inParam.cbDownLoadPos = DownLoadPosCallBack
inParam.dwPosUser = 0
inParam.fDownLoadDataCallBack = DownLoadDataCallBack
inParam.dwDataUser = 0
inParam.nPlayDirection = 0
inParam.nWaittime = 5000
inParam.stStartTime.dwYear = start_time.dwYear
inParam.stStartTime.dwMonth = start_time.dwMonth
inParam.stStartTime.dwDay = start_time.dwDay
inParam.stStartTime.dwHour = start_time.dwHour
inParam.stStartTime.dwMinute = start_time.dwMinute
```

```
inParam.stStartTime.dwSecond = start_time.dwSecond
inParam.stStopTime.dwYear = end_time.dwYear
inParam.stStopTime.dwMonth = end_time.dwMonth
inParam.stStopTime.dwDay = end_time.dwDay
inParam.stStopTime.dwHour = end_time.dwHour
inParam.stStopTime.dwMinute = end_time.dwMinute
inParam.stStopTime.dwSecond = end_time.dwSecond
outParam = NET_OUT_PLAY_BACK_BY_TIME_INFO()

nchannel = self.Channel_comboBox.currentIndex()
self.playbackID = self.sdk.PlayBackByTimeEx2(self.loginID, nchannel, inParam, outParam)
if self.playbackID != 0:
    self.PlayBack_pushbutton.setText("(Stop)")
    self.Pause_pushbutton.setEnabled(True)
    self.Channel_comboBox.setEnabled(False)
    self.StreamTyp_comboBox.setEnabled(False)
    self.Channel_comboBox.repaint()
    self.StreamTyp_comboBox.repaint()
    self.PlayBackWnd.repaint()
else:
QMessageBox.critical(self, '(prompt)', self.sdk.GetLastErrorMessage(), QMessageBox.No)

# Pause playback
result = self.sdk.PausePlayBack(self.playbackID, True)

# Resume playback
result = self.sdk.PausePlayBack(self.playbackID, False)

# Stop playback
result = self.sdk.StopPlayBack(self.playbackID)
if result:
    self.playbackID = 0
if not result:
    QMessageBox.critical(self, '(prompt)', self.sdk.GetLastErrorMessage(), QMessageBox.No)
    return
```

2.6 Record Download

2.6.1 Introduction

Video surveillance system widely applies to safe city, airport, metro, bank and factory. When any event occurs, you need to download the video records and report to the leaders, public security bureau, or mass media. Therefore, record download is an important function.

The record download function helps you obtain the records saved on the device through SDK and save into the local. It allows you to download from the selected channels and export to the local disk or external USB flash drive. The downloaded files are in the format of Dahua which requires Dahua player or integrated Dahua playsdk to play.

2.6.2 Interface Overview

Table 2-6 Interfaces of record download

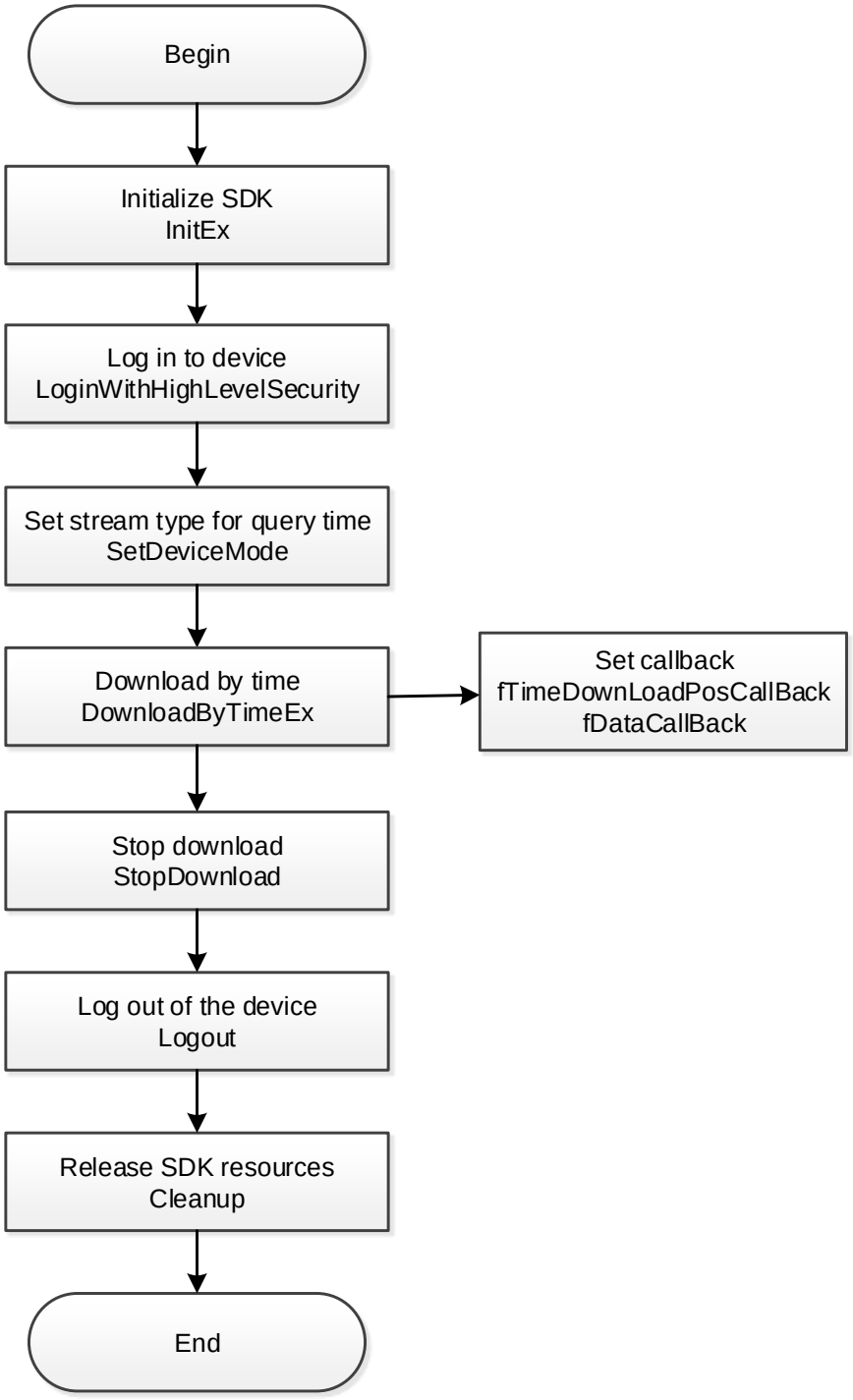
Interface	Implication
InitEx	Initialize SDK.
Cleanup	Clean up SDK.
LoginWithHighLevelSecurity	Log in with high level security.
Logout	Log out.
SetDeviceMode	Set device mode.
DownloadByTimeEx	Download by time.
StopDownload	Stop record download..

2.6.3 Process

You can import the start time and end time of download. SDK can download the specified record file and save it to the required place.

You can also provide a callback pointer to SDK which calls back the specified record file to you。

Figure 2-7 Process of download by time



Process Description

Step 1 Call **InitEx** to initialize SDK.
Step 2 Call **LoginWithHighLevelSecurity** to log in to the device.
Step 3 Call **SetDeviceMode** to set the download stream type.
Step 4 Call **DownloadByTimeEx** to start downloading by time.
Step 5 Call **StopDownload** to stop download.
Step 6 (Optional) Call **fTimeDownLoadPosCallBack** to update the download progress.
Step 7 Call **Logout** to log out of the device.
Step 8 Call **Cleanup** to release SDK resources.

2.6.4 Example Code

```
# Set the download stream type. Here set the type as main stream.
stream_type = c_int(0)
result = self.sdk.SetDeviceMode(self.loginID, int(EM_USEDEV_MODE.RECORD_STREAM_TYPE),
stream_type)
if not result:
QMessageBox.critical(self, '(prompt)', self.sdk.GetLastErrorMessage(), QMessageBox.No)

# Start record download.
start_date = self.Start_dateTimeEdit.date()
start_time = self.Start_dateTimeEdit.time()
startDateTime = NET_TIME()
startDateTime.dwYear = start_date.year()
startDateTime.dwMonth = start_date.month()
startDateTime.dwDay = start_date.day()
startDateTime.dwHour = start_time.hour()
startDateTime.dwMinute = start_time.minute()
startDateTime.dwSecond = start_time.second()

end_date = self.End_dateTimeEdit.date()
end_time = self.End_dateTimeEdit.time()
enddateTime = NET_TIME()
enddateTime.dwYear = end_date.year()
enddateTime.dwMonth = end_date.month()
enddateTime.dwDay = end_date.day()
enddateTime.dwHour = end_time.hour()
enddateTime.dwMinute = end_time.minute()
enddateTime.dwSecond = end_time.second()

save_file_name = 'D:\savedata.dav'# Path and name of saved files.
nchannel = self.Channel_comboBox.currentIndex()
self.downloadID = self.sdk.DownloadByTimeEx(self.loginID, nchannel,
int(EM_QUERY_RECORD_TYPE.ALL), startDateTime, enddateTime, save_file_name,
TimeDownLoadPosCallBack, 0, DownLoadDataCallBack, 0)
if self.downloadID:
    self.Download_pushButton.setText("(Stop)")
else:
    QMessageBox.critical(self, '(prompt)', self.sdk.GetLastErrorMessage(), QMessageBox.No)
```

```
# Stop record download.
result = self.sdk.StopDownload(self.downloadID)
if result:
    self.downloadID = 0

# Callback
@WINFUNCTYPE(None, c_longlong, c_ulong, POINTER(c_ubyte), c_ulong, c_longlong)
def DownLoadDataCallBack(lPlayHandle, dwDataType, pBuffer, dwBufSize, dwUser):
    pass

@WINFUNCTYPE(None, c_longlong, c_ulong, c_ulong, c_int, POINTER(NET_RECORDFILE_INFO),
c_ulong)
def TimeDownLoadPosCallBack(lPlayHandle, total_size, download_size, index, recordfileinfo, dwUser):
try:
# Display download progress.
        if download_size == 0xffffffff:
            self.downloadID = 0
            self.Download_progressBar.setValue(0)
            self.Download_pushButton.setText("(download)")
            self.Message_label.setText("Download End (Stop download)!")
        elif download_size == 0xfffffffe:
            self.downloadID = 0
            self.Download_progressBar.setValue(0)
            self.Download_pushButton.setText("(download)")
            self.Message_label.setText("Download Failed (Failed to download)!")
        else:
            if download_size >= total_size:
                self.Download_progressBar.setValue(100)
            else:
                percentage = int(download_size * 100 / total_size)
                self.Download_progressBar.setValue(percentage)
    except Exception as e:
        print(e)
    except Exception as e:
        print(e)
```

2.7 Device Control

2.7.1 Introduction

Get and set device time, and restart device remotely.

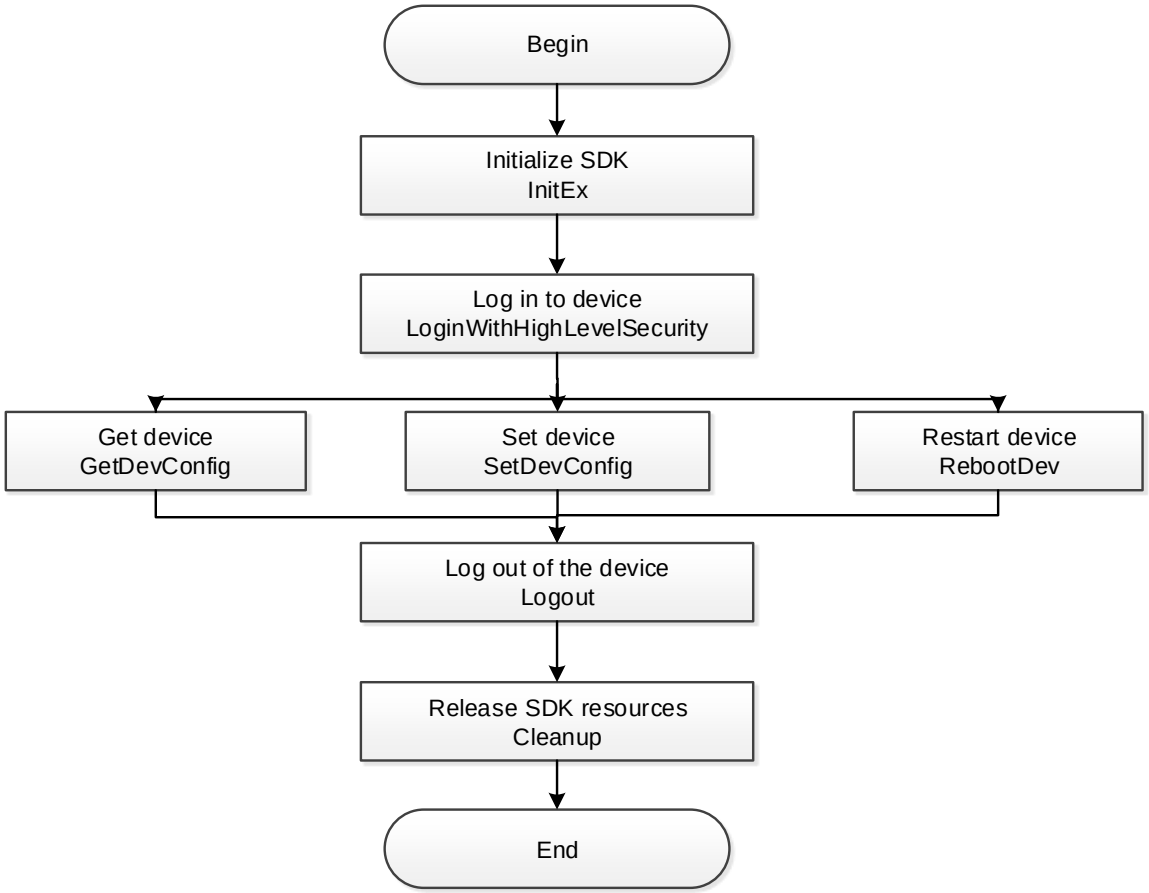
2.7.2 Interface Overview

Table 2-7 Interfaces of device control

Interface	Implication
InitEx	Initialize SDK.
Cleanup	Clean up SDK.
LoginWithHighLevelSecurity	Log in with high level security.
Logout	Log out.
GetDevConfig	Query configuration info.
SetDevConfig	Set configuration info.
RebootDev	Restart device.

2.7.3 Process

Figure 2-8 Process of device control



Process Description

Step 1 Call **InitEx** to initialize SDK.
Step 2 Call **LoginWithHighLevelSecurity** to log in to the device.
Step 3 (Optional) Call **GetDevConfig** to get device time.
Step 4 (Optional) Call **SetDevConfig** to set device time.
Step 5 (Optional) Call **RebootDev** to restart device.
Step 6 Call **Logout** to log out of the device.
Step 7 Call **Cleanup** to release SDK resources.

2.7.4 Example Code

```
# Get device time.
time = NET_TIME()
result = self.sdk.GetDevConfig(self.loginID, int(EM_DEV_CFG_TYPE.TIMECFG), -1, time, sizeof(NET_TIME))
if not result:
    QMessageBox.critical(self, '(prompt)', self.sdk.GetLastErrorMessage(), QMessageBox.Ok, QMessageBox.No)
else:
    get_time = QDateTime(time.dwYear, time.dwMonth, time.dwDay, time.dwHour, time.dwMinute, time.dwSecond)
    self.Time_dateTimeEdit.setDateTime(get_time)

# Set device time.
device_date = self.Time_dateTimeEdit.date()
device_time = self.Time_dateTimeEdit.time()
deviceDateTime = NET_TIME()
deviceDateTime.dwYear = device_date.year()
deviceDateTime.dwMonth = device_date.month()
deviceDateTime.dwDay = device_date.day()
deviceDateTime.dwHour = device_time.hour()
deviceDateTime.dwMinute = device_time.minute()
deviceDateTime.dwSecond = device_time.second()

result = self.sdk.SetDevConfig(self.loginID, int(EM_DEV_CFG_TYPE.TIMECFG), -1, deviceDateTime, sizeof(NET_TIME))
if not result:
    QMessageBox.critical(self, '(prompt)', self.sdk.GetLastErrorMessage(), QMessageBox.Ok, QMessageBox.No)

# Restart device.
```

```
result = self.sdk.RebootDev(self.loginID)
if not result:
    QMessageBox.critical(self, '(prompt)', self.sdk.GetLastErrorMessage(), QMessageBox.Ok, QMessageBox.No)
```

2.8 Remote Snapshot

2.8.1 Introduction

Call NetSDK interface to send snapshot command. Device will capture images from real-time monitoring and send them to NetSDK, and then NetSDK will return the image data to you.

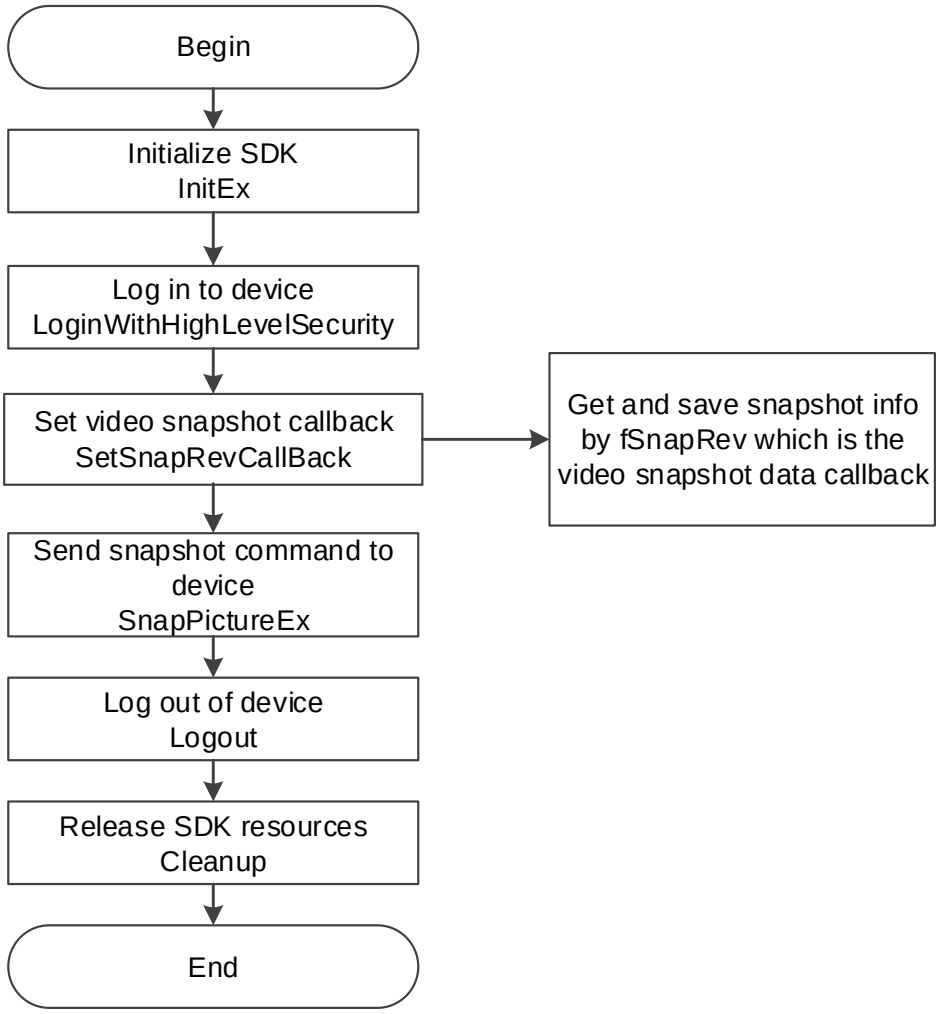
2.8.2 Interface Overview

Table 2-8 Interfaces of remote snapshot

Interface	Implication
InitEx	Initialize SDK.
Cleanup	Clean up SDK.
LoginWithHighLevelSecurity	Log in with high level security.
SetSnapRevCallBack	Set remote snapshot callback.
SnapPictureEx	Snapshot extension interface.
Logout	Log out.
GetLastError	Get error codes of interfaces that failed to be called.

2.8.3 Process

Figure 2-9 Process of remote snapshot



Process Description

Step 1 Call **InitEx** to initialize SDK.

Step 2 Call **LoginWithHighLevelSecurity** to log in to the device.

Step 3 Call **SetSnapRevCallBack** to set snapshot callback. When NetSDK receives image data sent from device, NetSDK will call fSnapRev to send image info and image data to you.

Step 4 Call **SnapPictureEx** to send snapshot command. Wait for the returned image info in fSnapRev.

Step 5 Call **Logout** to log out of the device.

Step 6 Call **Cleanup** to release SDK resources.

Notes for Process

- Call **InitEx** only once before using the SDK during the entire Demo running process. And call **Cleanup** once when all SDK-related functions finish to release SDK resources. These two interfaces do not need to be called with every function.

- The time interval for snapshot should be more than 1 second. 3 seconds are recommended.

2.8.4 Example Code

Code path:

Demo\CapturePicture\CaptureDemo.py

```
def capture_btn_onclick(self):
    # Set snapshot callback.
    dwUser = 0
    self.sdk.SetSnapRevCallBack(CaptureCallBack, dwUser)
    channel = self.Channel_comboBox.currentIndex()
    snap = SNAP_PARAMS()
    snap.Channel = channel
    snap.Quality = 1
    snap.mode = 0
    # Snapshot
    self.sdk.SnapPictureEx(self.loginID, snap)
```

2.9 Alarm Upload

2.9.1 Introduction

Alarm upload, that is, the device sends an alarm to the platform to inform when the events to be set have occurred. The platform can receive information such as external alarms, video signal loss alarms, privacy masking alarms, and motion detection alarms,

Alarm upload can be realized by NetSDK active login device and subscription of the alarm function to the device, which will send the detected alarm event to NetSDK.

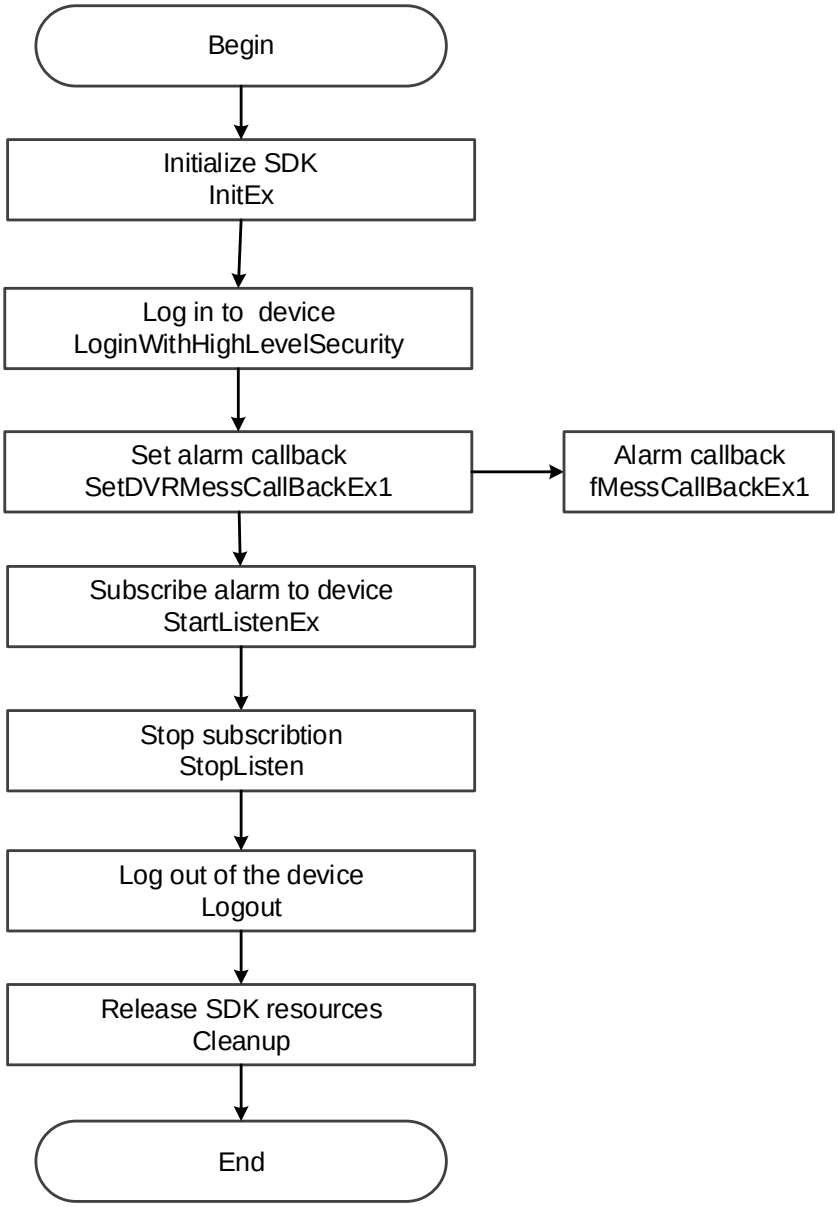
2.9.2 Interface Overview

Table 2-9 Interfaces of alarm upload

Interface	Implication
InitEx	Initialize SDK.
Cleanup	Clean up SDK.
LoginWithHighLevelSecurity	Log in with high level security.
SetDVRMessCallBackEx1	Set alarm callback.
StartListenEx	Alarm susbscribtion extension interface.
StopListen	Stop alarm susbscribtion.
Logout	Log out.
GetLastError	Get error codes of interfaces that fail to be called.

2.9.3 Process

Figure 2-10 Process of alarm upload



Process Description

Step 1 Call **InitEx** to initialize SDK.

Step 2 Call **LoginWithHighLevelSecurity** to log in to the device.

Step 3 Call **SetDVRMessCallBackEx1** to set alarm callback before alarm subscription.

Step 4 Call **StartListenEx** to subscribe to alarm from device. Then the uploaded event will be sent to you by **fMessCallBackEx1**.

Step 5 Call **StopListen** to stop subscribtion.

Step 6 Call **Logout** to log out of the device.

Step 7 Call **Cleanup** to release SDK resources.

Notes for Process

Call **InitEx** only once before using the SDK during the entire Demo running process. And call **Cleanup** once when all SDK-related functions finish, to release SDK resources. These two interfaces do not need to be called with every function.

2.9.4 Example Code

Code path:

Demo\AlarmListen\ AlarmListenDemo.py

```
def __init__(self):
    super(StartListenWnd, self).__init__()
    self.setupUi(self)
    # Initialize
    self.init_ui()

    # Used variables and callbacks by NetSDK
    self.loginID = C_LLONG()
    self.m_DisConnectCallBack = fDisConnect(self.DisConnectCallBack)
    self.m_ReConnectCallBack = fHaveReConnect(self.ReConnectCallBack)

    # Get NetSDK objectand initialize
    self.sdk = NetClient()
    self.sdk.InitEx(self.m_DisConnectCallBack)
    self.sdk.SetAutoReconnect(self.m_ReConnectCallBack)
    # Set alarm callback
    self.sdk.SetDVRMessCallBackEx1(MessCallback,0)

def attach_btn_onclick(self):
    self.row = 0
    self.column = 0
    self.Alarmlisten_tableWidget.clear()
    self.Alarmlisten_tableWidget.setHorizontalHeaderLabels(['(No.)', '(Time)', ' (Channel)', ' (Alarm Type)', ' (Status)'])
    result = self.sdk.StartListenEx(self.loginID)
    if result:
        QMessageBox.about(self, '(prompt)', "(Subscribe alarm success)")
        self.Stopalarmlisten_pushButton.setEnabled(True)
        self.Alarmlisten_pushButton.setEnabled(False)
    else:
        QMessageBox.about(self, '(prompt)', 'error:' + str(self.sdk.GetLastError()))
```

```
def detach_btn_onclick(self):
    if (self.loginID > 0):
        self.sdk.StopListen(self.loginID)
    self.Stopalarmlisten_pushButton.setEnabled(False)
    self.Alarmlisten_pushButton.setEnabled(True)
```

2.10 Intelligent Traffic Event Upload

2.10.1 Introduction

Intelligent traffic event upload is the function to analyze real-time stream from intelligent traffic devices. According to the pre-defined rules, SDK will check whether to upload events and carry images.

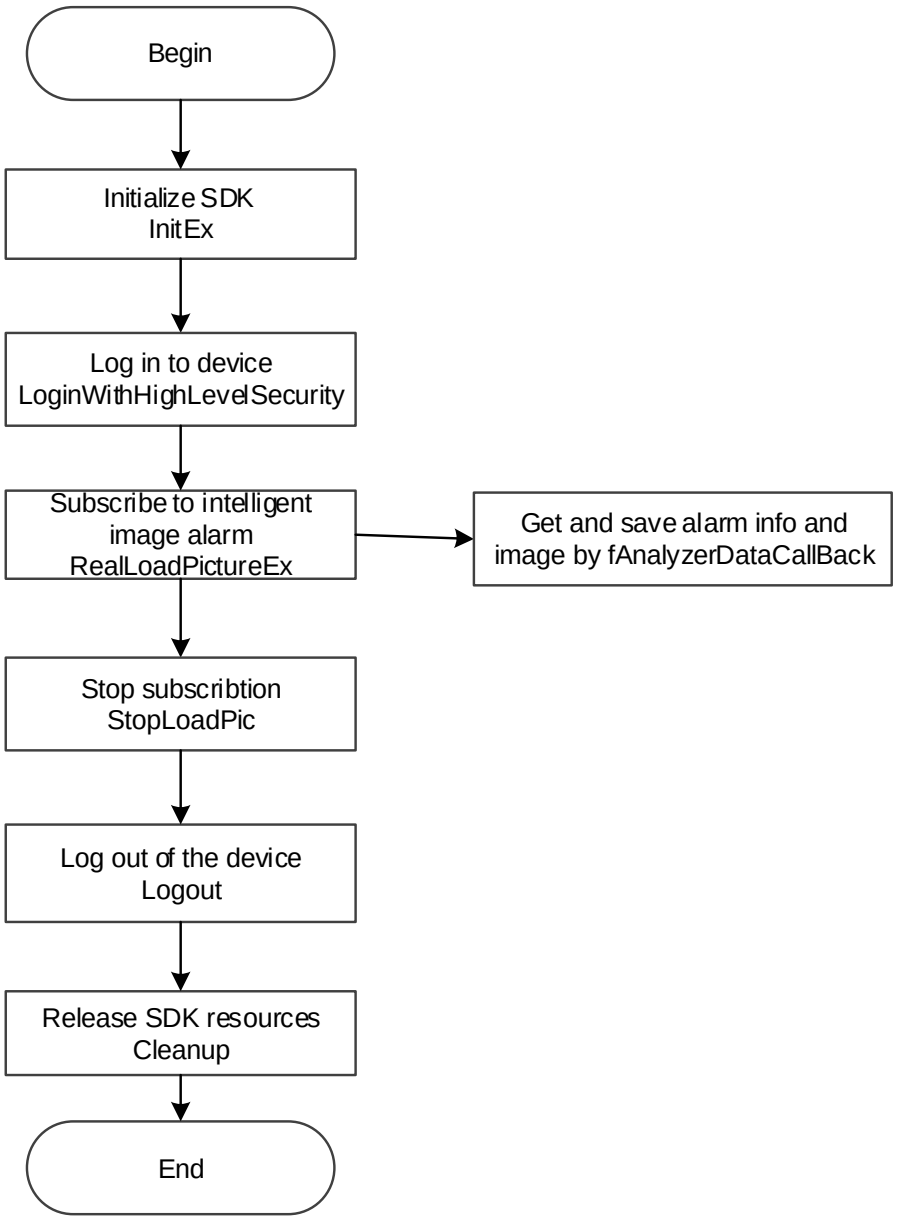
2.10.2 Interface Overview

Table 2-10 Interfaces of intelligent traffic event upload

Interface	Implication
InitEx	Initialize SDK.
Cleanup	Clean up SDK.
LoginWithHighLevelSecurity	Log in with high level security.
RealLoadPictureEx	Intelligent image alarm subscribtion interface.
StopLoadPic	Stop uploading intelligent analysis data-image.
Logout	Log out.
GetLastError	Get error codes of interfaces that fail to be called.

2.10.3 Process

Figure 2-11 Process of intelligent traffic event upload



Process Description

Step 1 Initialize SDK.

Step 2 Call **LoginWithHighLevelSecurity** to log in to the device.

Step 3 Call **RealLoadpictureEx** to subscribe to alarm from device, and the dwAlarmType should correspond to the enumeration values of EM_EVENT_IVS_TYPE. After the subscription, the uploaded event will be sent to you by callback which is be set in fAnalyzerDataCallBack. The main use of callback is to display and save events.

Step 4 Call **StopLoadPic** to stop subscription of intelligent traffic event.

Step 5 Call **Logout** to log out of the device.

Step 6 Call **Cleanup** to release SDK resources.

Notes for Process

Call **InitEx** only once before using the SDK during the entire Demo running process. And call **Cleanup** once when all SDK-related functions finish, to release SDK resources. These two interfaces do not need to be called with every function.

2.10.4 Example Code

2.10.4.1 Intelligent Traffic Junction

Code path:
\Demo\IntelligentTrafficDemo

```
# Subscribe to intelligent traffic junction event
def attach_btn_onclick(self):
    self.Attach_tableWidget.setHorizontalHeaderLabels(['(Time)', '(Event)', '(Plate No.)', '(Plate Color)', '(Vehicle Type)', '(Vehicle Color)'])
    channel = self.Channel_comboBox.currentIndex()
    bNeedPicFile = 1
    dwUser = 0
    self.attachID = self.sdk.RealLoadPictureEx(self.loginID, channel, EM_EVENT_IVS_TYPE.TRAFFICJUNCTION, bNeedPicFile, AnalyzerDataCallBack, dwUser, None)
    if not self.attachID:
        QMessageBox.about(self, '(prompt)', 'error:' + str(self.sdk.GetLastError()))
    else:
        self.Attach_pushButton.setEnabled(False)
        self.Detach_pushButton.setEnabled(True)
        QMessageBox.about(self, '(prompt)', "(Subscribe success)")

# Stop subscription
def detach_btn_onclick(self):
    if (self.attachID == 0):
        return
    self.sdk.StopLoadPic(self.attachID)
    self.attachID = 0
    self.Attach_pushButton.setEnabled(True)
    self.Detach_pushButton.setEnabled(False)
    self.Attach_tableWidget.clear()
    self.row = 0
    self.column = 0
    self.Attach_tableWidget.viewport().update()
    self.Attach_tableWidget.setHorizontalHeaderLabels([' (Time)', ' (Event)', '(Plate No.)', '(Plate Color)', '(Vehicle Type)', '(Vehicle Color)'])
```

2.10.4.2 Face Recognition Event

Code path:
Demo\FaceRecognitionDemo\ FaceRecognitionDemo.py

```
    def listenevent_btn_onclick(self):
    if not self.realloadID:
        channel = self.Channel_comboBox.currentIndex()
        self.realloadID = self.sdk.RealLoadPictureEx(self.loginID, channel,
EM_EVENT_IVS_TYPE.ALL, True, self.m_AnalyzerDataCallBack)
        if self.realloadID != 0:
            self.ListenEvent_pushButton.setText("Stop subscription(Detach Listen)")
        else:
            QMessageBox.critical(self, '(prompt)', self.sdk.GetLastErrorMessage(), QMessageBox.No)
    else:
        result = self.sdk.StopLoadPic(self.realloadID)
        if result:
            self.ListenEvent_pushButton.setText("Subscribe to event (Listen Event)")
            self.realloadID = 0
        else:
            QMessageBox.critical(self, '(prompt)', self.sdk.GetLastErrorMessage(), QMessageBox.No)

def AnalyzerDataCallBack(self, lAnalyzerHandle, dwAlarmType, pAlarmInfo, pBuffer, dwBufSize,
dwUser, nSequence, reserved):
    if lAnalyzerHandle == self.realloadID:
        if dwAlarmType == EM_EVENT_IVS_TYPE.FACERECOGNITION:
            alarm_info = cast(pAlarmInfo,
POINTER(DEV_EVENT_FACERECOGNITION_INFO)).contents
            self.show_recognition_info(alarm_info, pBuffer, dwBufSize)
```

2.10.4.3 Face Detection Event

Code path:
Demo\FaceRecognitionDemo\ FaceRecognitionDemo.py

```
def listenevent_btn_onclick(self):
    if not self.realloadID:
        channel = self.Channel_comboBox.currentIndex()
        self.realloadID = self.sdk.RealLoadPictureEx(self.loginID, channel,
EM_EVENT_IVS_TYPE.ALL, True, self.m_AnalyzerDataCallBack)
        if self.realloadID != 0:
            self.ListenEvent_pushButton.setText("Stop subscription (Detach Listen)")
        else:
            QMessageBox.critical(self, '(prompt)', self.sdk.GetLastErrorMessage(), QMessageBox.No)
    else:
        result = self.sdk.StopLoadPic(self.realloadID)
        if result:
            self.ListenEvent_pushButton.setText("Subscribe to event (Listen Event)")
```

```
                self.realloadID = 0
            else:
                QMessageBox.critical(self, '(prompt)', self.sdk.GetLastErrorMessage(), QMessageBox.No)

def AnalyzerDataCallBack(self, lAnalyzerHandle, dwAlarmType, pAlarmInfo, pBuffer, dwBufSize,
dwUser, nSequence, reserved):
    if lAnalyzerHandle == self.realloadID:
        if dwAlarmType == EM_EVENT_IVS_TYPE.FACEDETECT:
            alarm_info = cast(pAlarmInfo, POINTER(DEV_EVENT_FACEDETECT_INFO)).contents
            self.show_detect_info(alarm_info, pBuffer, dwBufSize)
```

3 Interface Definition

3.1 SDK Initialization

3.1.1 InitEx

Table 3-1 Initialize SDK

Item	Description	
Name	Initialize SDK.	
Function	def InitEx(cls, call_back: fDisConnect = None, user_data: C_LDWORD = 0, init_param: NETSDK_INIT_PARAM = NETSDK_INIT_PARAM()) -> int	
Parameter	[in] call_back	Disconnection callback.
	[in] user_data	User parameter of disconnection callback.
	[in] init_param	Initialzie parameters.
Return value	● Success: 1. ● Failure: 0.	
Note	It is the precondition for calling other function modules. If the callback is set as None, the callback will not be sent to the user after the device is disconnected. The parameter user_data passed in by InitEx will be returned in the same field user_data of fDisConnect. User_data is not processed inside NetSDK, and is only used to carry user data into the callback.	

3.1.2 Cleanup

Table 3-2 Clean up SDK

Item	Description
Name	Clean up SDK.
Function	def Cleanup(cls)
Parameter	None.
Return value	None.
Note	Call the SDK cleanup interface before the process ends.

3.1.3 SetAutoReconnect

Table 3-3 Set reconnection callback

Item	Description
Name	Set auto reconnection callback.

Item	Description	
Function	def SetAutoReconnect(cls, call_back: fHaveReConnect, user_data: C_LDWORD = None)	
Parameter	[in] call_back	Reconnection callback.
	[in] user_data	User parameter of disconnection callback.
Return value	None.	
Note	Set the reconnection callback interface. If the callback is set as None, it will not connect automatically.	

3.2 Device Search and Device Initialization

3.2.1 StartSearchDevicesEx

Table 3-4 Async device search

Item	Description	
Name	Async device search.	
Function	def StartSearchDevicesEx(cls, pInBuf: NET_IN_STARTSERACH_DEVICE, pOutBuf: NET_OUT_STARTSERACH_DEVICE) -> C_LLONG	
Parameter	[in] pInBuf	Async device searching input structure.
	[out] pOutBuf	Async device searching output structure.
Return value	● Success: Search handle. ● Failure: 0. ● Call GetLastError to get error codes.	
Note	Only support searching for devices within the same network segment. The number of calls to the search interface is the same as the number of network cards. After the device searching is successful, bind the search handle to the IP. After the callback search result is returned, find the corresponding local IP through the search handle, and pass in the local IP when initializing the device account.	

3.2.2 SearchDevicesByIPs

Table 3-5 Search for device in cross-segment

Item	Description
Name	Search for device IP in cross-segemt.
Function	def SearchDevicesByIPs(cls, pIpSearchInfo: DEVICE_IP_SEARCH_INFO, cbSearchDevices: fSearchDevicesCB, dwUserData: C_LDWORD, szLocalIp: c_char_p = None,

Item	Description	
	dwWaitTime: C_DWORD = 5000) -> c_int:	
Parameter	[in] pIpSearchInfo	Search device info.
	[in] cbSearchDevices	Search device callback. When a device response packet returns, NetSDK parses the response packet into valid information and notifies users by the callback. For details, see the description of fSearchDevicesCB. Callback cannot be null.
	[in] dwUserData	User data. NetSDK returns the data to users by fSearchDevicesCB whichis the device search callback.
	[in] szLocalIp	Local IP. The default value is None. And no value enrtered is allowed.
	[in] dwWaitTime	Search time expected by users. Se the parameters as nedded. This interfacre is a synchronous interface, so it only returns when the waiting time of search is finished.
Return value	● Success: 1. ● Failure: 0.	
Note	This interfacre is a synchronous interface, so it only returns when the waiting time of search is finished. Enter the search time according to own network situations.	

3.2.3 StopSearchDevices

Table 3-6 Stop async searching

Item	Description	
Name	Stop searching for devices with the same network segment, such as IPC and NVS.	
Function	def StopSearchDevices(cls, lSearchHandle: C_LLONG) -> c_int	
Parameter	[in] lSearchHandle	Async search for device ID. Return value of async search interfaces, such as StartSearchDevicesEx.
Return value	● Success: 1. ● Failure: 0.	
Note	Use with StartSearchDevicesEx in pairs.	

3.2.4 InitDevAccount

Table 3-7 Initialize device

Item	Description
Name	Initialize device account.

Item	Description	
Function	def InitDevAccount(cls, pInitAccountIn: NET_IN_INIT_DEVICE_ACCOUNT, pInitAccountOut: NET_OUT_INIT_DEVICE_ACCOUNT, dwWaitTime: int = 5000, szLocalIp: c_char_p = None) -> c_int	
Parameter	[in] pInitAccountIn	Input structure of decive initialization.
	[out]pInitAccountOut	Output structure of decive initialization.
	[in] dwWaitTime	Waiting time. The unit is ms.
	[in] szLocalIp	Local IP. Should be the same with szLocalIp filed of pInBuf of StartSearchDevicesEx.
Return value	● Success: 1. ● Failure: 0.	
Note	If the PC has several network cards, you need to call StartSearchDevicesEx for several times. After the search is successful, the search handle is bound to the IP. When searching for callback information, find the corresponding local IP by the search handle. During initialization, szLocalIp should be the local IP.	

3.3 Device Login

3.3.1 LoginWithHighLevelSecurity

Table 3-8 Log in

Item	Description	
Name	Log in to the device.	
Function	def LoginWithHighLevelSecurity(cls, stuInParam: NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY, stuOutParam: NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY) -> tuple	
Parameter	[in] stuInParam	Input parameter structure.
	[out] stuOutParam	Output parameter structure.
	[out] device_info	Device info.
	[out] error_message	Error info of login interfece.
Return value	● Success: Non-0. ● Failure: 0.	
Note	None.	

3.3.2 Logout

Table 3-9 Log out

Item	Description
Name	Log out of the device.

Item	Description	
Function	def Logout(cls, login_id: int) -> int	
Parameter	[in]login_id	Return value of LoginWithHighLevelSecurity.
Return value	● Success: 1. ● Failure: 0.	
Note	None.	

3.4 Real-time Monitoring

3.4.1 RealPlayEx

Table 3-10 Start monitoring

Item	Description	
Name	Start real-time monitoring.	
Function	def RealPlayEx(cls, login_id: int, channel: int, hwnd: int, play_type=SDK_RealPlayType.Realplay) -> C_LLONG	
Parameter	[in] login_id	Return value of LoginWithHighLevelSecurity.
	[in] channel	Video channel No. is a round No., starting from 0.
	[in] hwnd	Window handle valid only under Windows system.
	[in] play_type	Live type.
Return value	● Success: Non-0. ● Failure: 0	
Note	Windows system: When hWnd is valid, the corresponding window displays picture. When hWnd is None, get the video data through setting a callback and send to user for handle.	

Table 3-11 Live type and meaning

Live type	Meaning
Realplay	Real-time live
Multiplay	Multi-picture live
Realplay_0	Real-time monitoring—main stream, equivalent to Realplay
Realplay_1	Real-time monitoring—sub stream 1
Realplay_2	Real-time monitoring—sub stream 2
Realplay_3	Real-time monitoring—sub stream 3
Multiplay_1	Multi-picture live—1 picture
Multiplay_4	Multi-picture live—4 pictures
Multiplay_8	Multi-picture live—8 pictures

Live type	Meaning
Multiplay_9	Multi-picture live—9 pictures
Multiplay_16	Multi-picture live—16 pictures
Multiplay_6	Multi-picture live—6 pictures
Multiplay_12	Multi-picture live—12 pictures
Multiplay_25	Multi-picture live—25 pictures
Multiplay_36	Multi-picture live—36 pictures

3.4.2 StopRealPlayEx

Table 3-12 Stop monitoring

Item	Description	
Name	Stop the real-time monitoring.	
Function	def StopRealPlayEx(cls, realplay_id: int) -> int	
Parameter	[in] realplay_id	Return value of RealPlayEx.
Return value	● Success: 1. ● Failure: 0.	
Note	None.	

3.5 Record Playback

3.5.1 SetDeviceMode

Table 3-13 Set working mode

Item	Description	
Name	Set working mode.	
Function	def SetDeviceMode(cls, login_id: int, emType: int, value: c_void_p) -> c_int	
Parameter	[in] login_id	Return value of LoginWithHighLevelSecurity.
	[in] emType	Working mode enumeration.
	[in] value	Structure correspondes to working mode.
Return value	● Success: 1. ● Failure: 0.	
Note	None.	

Table 3-14 Working mode and corresponding structure

emType Enumeration	Meaning	Structure
RECORD_STREAM_TYPE	Set the record stream type to query and playback by time. 0: Main and sub stream	None

emType Enumeration	Meaning	Structure
	1: Main stream 2: Sub stream	
RECORD_TYPE	Set the record file type to playback and download by time.	EM_RECORD_TYPE

3.5.2 QueryRecordFile

Table 3-15 Query for all the record files within a period

Item	Description	
Name	Query for all the record files within a period.	
Function	def QueryRecordFile(cls, login_id: int, channel_id: int, recordfile_type: int, start_time: NET_TIME, end_time: NET_TIME, card_id: str, wait_time:int, is_querybytime:bool) -> tuple	
Parameter	[in] login_id	Return value of LoginWithHighLevelSecurity.
	[in] channel_id	Device channel.
	[in] recordfile_type	Query type. Refer to EM_QUERY_RECORD_TYPE.
	[in] start_time	Start time.
	[in] end_time	End time.
	[in] card_id	Card ID.
	[in] wait_time	Waiting time.
	[in] is_querybytime	Whether to query by time.
	[out] file_count	Returned file No..
	[out] recordfile_infos	File info of returned records. The strcture group of NET_RECORDFILE_INFO.
Return value	● Success: 1. ● Failure: 0.	
Note	Before playback, call this interface to query the video records. When the info of searched record within the entered time is greater than the defined buffer size, SDK only returns the records that can be stored in the buffer. You can continue to query as needed.	

3.5.3 PlayBackByTimeEx2

Table 3-16 Playback by time

Item	Description
Name	Playback by time.

Item	Description	
Function	def PlayBackByTimeEx2(cls, login_id: int, channel_id: int, in_param: NET_IN_PLAY_BACK_BY_TIME_INFO, out_param: NET_OUT_PLAY_BACK_BY_TIME_INFO) -> int:	
Parameter	[in] login_id	Return value of LoginWithHighLevelSecurity.
	[in] channel_id	Device channel No..
	[in] in_param	Query input condition.
	[out] out_param	Query output information.
Return value	● Success: Non-0. ● Failure: 0.	
Note	For the callback declaration cbDownLoadPos and fDownLoadDataCallBack in NET_IN_PLAY_BACK_BY_TIME_INFO, see "4 Callback Definition." The parameters hWnd and fDownLoadDataCallBack in pstNetIn cannot be None at the same time; otherwise, the interface calling will be failed returned.	

3.5.4 StopPlayBack

Table 3-17 Stop playback

Item	Description	
Name	Stop playback.	
Function	def StopPlayBack(cls, playback_id: int) -> int	
Parameter	[in] playback_id	Playback handle. Return value of PlayBackByTimeEx2.
Return value	● Success: 1. ● Failure: 0.	
Note	None.	

3.5.5 PausePlayBack

Table 3-18 Pause or resume playback

Item	Description	
Name	Pause or resume playback.	
Function	def PausePlayBack(cls, playback_id: int, is_pause: bool) -> int:	
Parameter	[in] playback_id	Playback handle. Return value of PlayBackByTimeEx2.
	[in] is_pause	Pause or resume. True: pause; False: resume.
Return value	● Success: 1. ● Failure: 0.	

Item	Description
Note	None.

3.6 Record Download

3.6.1 DownloadByTimeEx

Table 3-19 Download by time

Item	Description	
Name	Download by time.	
Function	def DownloadByTimeEx(cls, login_id: int, channel_id: int, recordfile_type: int, start_time: NET_TIME, end_time: NET_TIME, save_filename: str, callback_timedownloadpos: fTimeDownLoadPosCallBack, time_UserData: C_LDWORD, callback_timedownloaddata: fDataCallBack, data_UserData: C_LDWORD, pReserved: int = 0) -> int	
Parameter	[in] login_id	Return value of LoginWithHighLevelSecurity.
	[in] channel_id	Device channel No., starting from 0.
	[in] recordfile_type	Record file type.
	[in] start_time	Start time.
	[in] end_time	End time.
	[in] save_filename	Record file name to be save. Full path.
	[in] callback_timedownloadpos	Download progress callback.
	[in] time_UserData	Customized dada of download progress callback.
	[in] callback_timedownloaddata	Download data callback.
	[in] data_UserData	Customized dada of download data callback.
	[in] pReserved	Reserved parameter.
Return value	● Success: Non-0. ● Failure: 0.	
Note	For callback declaration of callback_timedownloadpos and callback_timedownloaddata, see "4 Callback Definition." sSavedFileName is not blank, and the record data is input into the file corresponding with the path. fDownLoadDataCallBack is not blank, and the record data is returned through callback.	

3.6.2 StopDownload

Table 3-20 Stop record download

Item	Description	
Name	Stop record download.	
Function	def StopDownload(cls, download_id: int) -> int	
Parameter	[in] download_id	Return value of DownloadByTimeEx.
Return value	● Success: 1. ● Failure: 0.	
Note	Stop downloading after it is completed or partially completed according to particular situation.	

3.7 Device Control

3.7.1 GetDevConfig

Table 3-21 Get device configuration info

Item	Description	
Name	Get device configuration info.	
Function	def GetDevConfig(cls, login_id: C_LLONG, cfg_type: C_DWORD, channel_id: C_LONG, out_buffer: C_LLONG, outbuffer_size: C_DWORD, wait_time: int = 5000) -> int	
Parameter	[in] login_id	Return value of LoginWithHighLevelSecurity.
	[in] cfg_type	Query time. For details, see the EM_DEV_CFG_TYPE enumeration in the SDK_Enum.py file.
	[in] channel_id	Quey channel No..
	[out] out_buffer	Obtained strcture data.
	[in] outbuffer_size	Data length of out_buffer.
	[in] wait_time	Timeout.
Return value	● Success: 1. ● Failure: 0.	
Note	None.	

Table 3-22 Configuration type enumeration

emType Enumeration	Description
TIMECFG	Time configuration. GetDevConfig and SetDevConfig are use together.

3.7.2 SetDevConfig

Table 3-23 Set device configuration info

Item	Description	
Name	Set device configuration info.	
Function	def SetDevConfig(cls, login_id: C_LLONG, cfg_type: C_DWORD, channel_id: C_LONG, in_buffer: C_LLONG, inbuffer_size: C_DWORD, wait_time: int = 5000) -> int	
Parameter	[in] login_id	Return value of LoginWithHighLevelSecurity.
	[in] cfg_type	Query type. For details, see the EM_DEV_CFG_TYPE enumeration in the SDK_Enum.py file.
	[in] channel_id	Quey channel No..
	[in] in_buffer	Imported strcture data.
	[in] inbuffer_size	Data length of in_buffer.
	[in] wait_time	Timeout.
Return value	● Success: 1. ● Failure: 0.	
Note	None.	

3.7.3 RebootDev

Table 3-24 Restart device

Item	Description	
Name	Restart device.	
Function	def RebootDev(cls, login_id: int) -> int:	
Parameter	[in] login_id	Return value of LoginWithHighLevelSecurity.
Return value	● Success: 1. ● Failure: 0.	
Note	None.	

3.8 Remote Snapshot

3.8.1 SetSnapRevCallBack

Table 3-25 Set remote snapshot callback

Item	Description	
Name	Set snapshot callback.	
Function	def SetSnapRevCallBack(cls, OnSnapRevMessage: fSnapRev, dwUser: C_LDWORD) -> None	
Parameter	[in] OnSnapRevMessage	Remote snapshot callback.
	[in] dwUser	User data. SDK will return data to users by fSnapRev.
Return value	None.	
Note	Call SetSnapRevCallBack before calling SnapPictureEx.	

3.8.2 SnapPictureEx

Table 3-26 Snapshot command intension interface

Item	Description	
Name	Snapshot command intension interface.	
Function	def SnapPictureEx(cls, lLoginID:C_LLONG, par:SNAP_PARAMS, reserved=0)->c_int	
Parameter	[in] lLoginID	Return value of LoginWithHighLevelSecurity.
	[in] par	Snapshot parameters. For detalis, see SNAP_PARAMS structure.
	[in] reserved	Picture format.
Return value	● Success: 1. ● Failure: 0.	
Note	None.	

3.9 Alarm Listening

3.9.1 SetDVRMessCallBackEx1

Table 3-27 Set alarm callback

Item	Description	
Name	Set alarm callback.	
Function	def SetDVRMessCallBackEx1(cls, cbMessage:fMessCallBackEx1, dwUser:C_LDWORD)->None	
Parameter	[in] cbMessage	Alarm callback. For details, see fMessCallBackEx1.
	[in] dwUser	User data. SDK will return data to users by fMessCallBackEx1.
Return value	None.	
Note	Call StartListenEx before calling SetDVRMessCallBackEx1.	

3.9.2 StartListenEx

Table 3-28 Start alarm subscription

Item	Description	
Name	Extension interface of device alarm subscribtion.	
Function	def StartListenEx(cls, lLoginID:C_LLONG)->c_int	
Parameter	[in] lLoginID	Return value of LoginWithHighLevelSecurity.
Return value	● Success: 1. ● Failure: 0.	
Note	All alarm evnets of devices are fed back by callback set in SetDVRMessCallBackEx1	

3.9.3 StopListen

Table 3-29 Stop alarm subscription

Item	Description	
Name	Stop alarm subscribtion.	
Function	def StopListen(cls, lLoginID:C_LLONG)->c_int	
Parameter	[in] lLoginID	Return value of LoginWithHighLevelSecurity.
Return value	● Success: 1. ● Failure: 0.	
Note	None.	

3.10 Intelligent Event Upload

3.10.1 RealLoadPictureEx

Table 3-30 Intelligent image alarm subscription

Item	Description	
Name	Intelligent image alarm subscription.	
Function	def RealLoadPictureEx(cls, lLoginID: C_LLONG, nChannelID: c_int, dwAlarmType: c_ulong, bNeedPicFile: c_int, cbAnalyzerData: fAnalyzerDataCallBack, dwUser: C_LDWORD = 0, reserved: c_void_p = None) -> C_LLONG	
Parameter	[in] lLoginID	Return value of LoginWithHighLevelSecurity.
	[in] nChannelID	Channel No. of intelligent image alarm subscribtion, starting from 0.
	[in] dwAlarmType	Alarm event type expected to subscribe. Refer to EM_EVENT_IVS_TYPE.
	[in] bNeedPicFile	Subscribe to image file or not? 1: Subscribe to image. 0: Not subscribe to image.
	[in]cbAnalyzerData	Callback of intelligent event. When there is intelligent image alarm be uploaded, NetSDK will returns data to users.
	[in] dwUser	User data. SDK will return data to users by fAnalyzerDataCallBack.
	[in] reserved	Reserved parameter.
Return value	● Success: ID of Intelligent image alarm subscription. ● Failure: 0.	
Note	If you need to subscribe to several events on one channel, set the evnt type as EM_EVENT_IVS_ALL to subscribe to all event types when calling RealLoadPictureEx, and then process the evnets you need.	

3.10.2 StopLoadPic

Table 3-31 Stop subscription of intelligent event

Item	Description
Name	Stop subscribtion of intelligent event.
Function	def StopLoadPic(cls, lAnalyzerHandle:C_LLONG)->c_int

Item	Description	
Parameter	[in] lAnalyzerHandle	Return value of RealLoadPictureEx.
Return value	● Success: 1. ● Failure: 0.	
Note	None.	

4 Callback Definition

4.1 fDisConnect

Table 4-1 Disconnection callback

Item	Description	
Name	Disconnection callback.	
Precondition	None.	
Function	fDisConnect = WINFUNCTYPE(None, C_LLONG, c_char_p, c_long, C_LDWORD)	
Parameter	lLoginID	Login handle.
	pchDVRIP	IP address.
	nDVRPort	Port.
	dwUser	User data.
Return value	None.	
Note	Be triggered when the device is disconnected. It is not recommended to call any NetSDK interface in this callback. If the callback in the Demo calls, then you can follow and call.	

4.2 fHaveReConnect

Table 4-2 Reconnection callback

Item	Description	
Name	Reconnection callback.	
Precondition	None.	
Function	fHaveReConnect = WINFUNCTYPE(None, C_LLONG, c_char_p, c_long, C_LDWORD)	
Parameter	lLoginID	Login handle.
	pchDVRIP	IP address.
	nDVRPort	Port.
	dwUser	User data.
Return value	None.	
Note	Be triggered when the device is disconnected. It is not recommended to call any NetSDK interface in this callback. If the callback in the Demo calls, then you can follow and call.	

4.3 fSearchDevicesCBEx

Table 4-3 Async device search callback

Item	Description	
Name	Device search callback prototype.	
Precondition	None.	
Function	fSearchDevicesCBEx = WINFUNCTYPE(None, C_LLONG, POINTER(DEVICE_NET_INFO_EX2), c_void_p)	
Parameter	lSearchHandle	Search handle.
	pDevNetInfo	Device info.
	pUserData	User data info.
Return value	None.	
Note	Device search callback. It is not recommended to call any NetSDK interface in this callback. If the callback in the Demo calls, then you can follow and call. Set the callback by StartSearchDeviceEx. When a device is searched, the SDK will call this callback.	

4.4 fSearchDevicesCB

Table 4-4 Device search callback

Item	Description	
Name	Device search callback prototype.	
Precondition	None.	
Function	fSearchDevicesCB = WINFUNCTYPE(None, POINTER(DEVICE_NET_INFO_EX), c_void_p)	
Parameter	pDevNetInfo	Info.
	pUserData	User data info.
Return value	None.	
Note	Device search callback. It is not recommended to call any NetSDK interface in this callback. If the callback in the Demo calls, then you can follow and call. Set the callback by SearchDevicesByIPs. When a device is searched, the SDK will call this callback.	

4.5 fDownLoadPosCallBack

Table 4-5 Playback progress callback

Item	Description
Name	Playback progress callback.
Precondition	None.
Function	fDownLoadPosCallBack = WINFUNCTYPE(None, C_LLONG, C_DWORD, C_DWORD, C_LDWORD)

Item	Description	
Parameter	lPlayHandle	Return handel of PlayBackByTimeEx.
	dwTotalSize	Total size of download.
	dwDownLoadSize	The size that has been downloaded.. ● -1: Current download or playback has been finished. ● -2: The user does not have permission to download or playback.
	dwUser	User data.
Return value	None.	
Note	Playback progress callback. It is not recommended to call any NetSDK interface in this callback. If the callback in the Demo calls, then you can follow and call.	

4.6 fDataCallBack

Table 4-6 Playback data callback

Item	Description	
Name	Playback data callback.	
Precondition	None.	
Function	fDataCallBack = WINFUNCTYPE(c_int, C_LLONG, C_DWORD, POINTER(c_ubyte), C_DWORD, C_LDWORD)	
Parameter	lPlayHandle	Playback data handle.
	dwDataType	Data type.
	pBuffer	Data buffer. Memory is released internally by NetSDK.
	dwBufSize	Data buffer size.
	dwUser	User data.
Return value	0: Failed to call back. The next callback will return the same data. 1: Succeed to call back. The next callback will return the subsequent data.	
Note	Data callback of downloading records.. It is not recommended to call any NetSDK interface in this callback. If the callback in the Demo calls, then you can follow and call.	

4.7 fTimeDownLoadPosCallBack

Table 4-7 Callback of download by time callback

Item	Description	
Name	Callback of download by time.	
Precondition	None.	
Function	fTimeDownLoadPosCallBack = WINFUNCTYPE(None, C_LLONG, C_DWORD, C_DWORD, c_int, NET_RECORDFILE_INFO, C_LDWORD)	
Parameter	lPlayHandle	Return handel of DownloadByTimeEx.
	dwTotalSize	Total size of download.
	dwDownLoadSize	The size that has been downloaded..

Item	Description	
	Index	Index.
	Recordfileinfo	Record file information.
	dwUser	User data.
Return value	None.	
Note	Download progress callback. It is not recommended to call any NetSDK interface in this callback. If the callback in the Demo calls, then you can follow and call.	

4.8 fAnalyzerDataCallBack

Table 4-8 Intelligent image alarm callback

Item	Description	
Name	Intelligent image alarm callback.	
Precondition	None.	
Function	fAnalyzerDataCallBack = WINFUNCTYPE(None, C_LLONG, C_DWORD, c_void_p, POINTER(c_ubyte), C_DWORD, C_LDWORD, c_int, c_void_p)	
Parameter	lAnalyzerHandle	Return handel of RealLoadPictureEx.
	dwAlarmType	Event type of EM_EVENT_IVS_TYPE.
	pAlarmInfo	Event info.
	pBuffer	Image data buffer.
	dwBufSize	Image data buffer size.
	dwUser	User data info entered by RealLoadPictureEx..
	nSequence	Situation of the same uploaded image. ● 0: First time to appear. ● 1: Same image will appear from this time on. ● 2: Last time to appear or only once.
	Reserved	Indicate the status of current called back data when int nState = (int)reserved. ● 0: Current data is real-time data. ● 1: Current data is offline data. ● 2: Offline data transmission ends.
Return value	None.	
Note	Intelligent image alarm callback. It is not recommended to call any NetSDK interface in this callback. If the callback in the Demo calls, then you can follow and call. Set the callback by RealLoadPictureEx. When an intelligent image event is uploaded, the SDK will call this callback. The dwAlarmType value varies according to different data type of pAlarmInfo.	

4.9 fSnapRev

Table 4-9 Snapshot callback

Item	Description	
Name	Snapshot callback prototype.	
Precondition	None.	
Function	fSnapRev = WINFUNCTYPE(None, C_LLONG, POINTER(c_ubyte), c_uint, c_uint, C_DWORD, C_LDWORD)	
	lLoginID	Login handle.
	pBuf	Image buffer.
	RevLen	Image size.
	EncodeType	Encode type: ● 10: Jpeg image. ● 0: I frame of mpeg4.
	CmdSerial	Command serial No..
	dwUser	User data entered by SetSnapRevCallBack.
Return value	None.	
Note	Snapshot callback function. It is not recommended to call any NetSDK interface in this callback. If the callback in the Demo calls, then you can follow and call. Set this callback by SetSnapRevCallBack. When the snapshot data is sent, the SDK will call this callback.	

4.10 fMessCallBackEx1

Table 4-10 Alarm upload callback

Item	Description	
Name	Alarm upload callback prototype.	
Precondition	None.	
Function	fMessCallBackEx1 = WINFUNCTYPE(None, c_long, C_LLONG, POINTER(c_char), C_DWORD, POINTER(c_char), c_long, c_int, c_long, C_LDWORD)	
Parameter	lCommand	Alarm type.
	lLoginID	Login handle.
	pBuf	Alarm info.
	dwBufLen	Alarm info size.
	pchDVRIP	IP address.
	nDVRPort	Port.
	bAlarmAckFlag	● 1: The event can be confirmed. ● 0: The event cannot be confirmed.
	nEventID	Used to assign values to the input parameters of the AlarmAck. hen bAlarmAckFlag is 1, the data is valid.
	dwUser	User data entered by SetDVRMessCallBackEx1.
Return value	None.	

Item	Description
Note	All registered devices use one alarm upload callback. You can identify the uploaded device by parameter lLoginI.D. Data type of pBuf varies according to lCommand value. It is not recommended to call any NetSDK interface in this callback. If the callback in the Demo calls, then you can follow and call.

Appendix 1 Cybersecurity Recommendations

Cybersecurity is more than just a buzzword: it's something that pertains to every device that is connected to the internet. IP video surveillance is not immune to cyber risks, but taking basic steps toward protecting and strengthening networks and networked appliances will make them less susceptible to attacks. Below are some tips and recommendations on how to create a more secured security system.

Mandatory actions to be taken for basic equipment network security:

1. **Use Strong Passwords**

   Please refer to the following suggestions to set passwords:

   - The length should not be less than 8 characters;
   - Include at least two types of characters; character types include upper and lower case letters, numbers and symbols;
   - Do not contain the account name or the account name in reverse order;
   - Do not use continuous characters, such as 123, abc, etc.;
   - Do not use overlapped characters, such as 111, aaa, etc.;

2. **Update Firmware and Client Software in Time**

   - According to the standard procedure in Tech-industry, we recommend to keep your equipment (such as NVR, DVR, IP camera, etc.) firmware up-to-date to ensure the system is equipped with the latest security patches and fixes. When the equipment is connected to the public network, it is recommended to enable the "auto-check for updates" function to obtain timely information of firmware updates released by the manufacturer.
   - We suggest that you download and use the latest version of client software.

"Nice to have" recommendations to improve your equipment network security:

1. **Physical Protection**

   We suggest that you perform physical protection to equipment, especially storage devices. For example, place the equipment in a special computer room and cabinet, and implement well-done access control permission and key management to prevent unauthorized personnel from carrying out physical contacts such as damaging hardware, unauthorized connection of removable equipment (such as USB flash disk, serial port), etc.

2. **Change Passwords Regularly**

   We suggest that you change passwords regularly to reduce the risk of being guessed or cracked.

3. **Set and Update Passwords Reset Information Timely**

   The equipment supports password reset function. Please set up related information for password reset in time, including the end user's mailbox and password protection questions. If the information changes, please modify it in time. When setting password protection questions, it is suggested not to use those that can be easily guessed.

4. **Enable Account Lock**

   The account lock feature is enabled by default, and we recommend you to keep it on to guarantee the account security. If an attacker attempts to log in with the wrong password several times, the corresponding account and the source IP address will be locked.

5. **Change Default HTTP and Other Service Ports**

We suggest you to change default HTTP and other service ports into any set of numbers between 1024~65535, reducing the risk of outsiders being able to guess which ports you are using.

6. **Enable HTTPS**

   We suggest you to enable HTTPS, so that you visit Web service through a secure communication channel.

7. **Enable Whitelist**

   We suggest you to enable whitelist function to prevent everyone, except those with specified IP addresses, from accessing the system. Therefore, please be sure to add your computer's IP address and the accompanying equipment's IP address to the whitelist.

8. **MAC Address Binding**

   We recommend you to bind the IP and MAC address of the gateway to the equipment, thus reducing the risk of ARP spoofing.

9. **Assign Accounts and Privileges Reasonably**

   According to business and management requirements, reasonably add users and assign a minimum set of permissions to them.

10. **Disable Unnecessary Services and Choose Secure Modes**

    If not needed, it is recommended to turn off some services such as SNMP, SMTP, UPnP, etc., to reduce risks.

    If necessary, it is highly recommended that you use safe modes, including but not limited to the following services:

    - SNMP: Choose SNMP v3, and set up strong encryption passwords and authentication passwords.
    - SMTP: Choose TLS to access mailbox server.
    - FTP: Choose SFTP, and set up strong passwords.
    - AP hotspot: Choose WPA2-PSK encryption mode, and set up strong passwords.

11. **Audio and Video Encrypted Transmission**

    If your audio and video data contents are very important or sensitive, we recommend that you use encrypted transmission function, to reduce the risk of audio and video data being stolen during transmission.

    Reminder: encrypted transmission will cause some loss in transmission efficiency.

12. **Secure Auditing**

    - Check online users: we suggest that you check online users regularly to see if the device is logged in without authorization.
    - Check equipment log: By viewing the logs, you can know the IP addresses that were used to log in to your devices and their key operations.

13. **Network Log**

    Due to the limited storage capacity of the equipment, the stored log is limited. If you need to save the log for a long time, it is recommended that you enable the network log function to ensure that the critical logs are synchronized to the network log server for tracing.

14. **Construct a Safe Network Environment**

    In order to better ensure the safety of equipment and reduce potential cyber risks, we recommend:

    - Disable the port mapping function of the router to avoid direct access to the intranet devices from external network.
    - The network should be partitioned and isolated according to the actual network needs. If there are no communication requirements between two sub networks, it is

suggested to use VLAN, network GAP and other technologies to partition the network, so as to achieve the network isolation effect.

- Establish the 802.1x access authentication system to reduce the risk of unauthorized access to private networks.
- It is recommended that you enable your device's firewall or blacklist and whitelist feature to reduce the risk that your device might be attacked.