

# **NetSDK\_JAVA Programming Manual (Field Surveillance Unit)**






## **User's Manual**



# Foreword

## Safety Instructions

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

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

## Revision History

Version	Revision Content	Release Time
V1.0.0	First release.	April 2021

# Glossary

This chapter provides the definitions to some of the terms appearing in the Manual to help you understand the function of each module.

Term	Description
Device ID	This ID uniquely identifies an external device for the monitoring and collection of various data. It is described as DeviceID in the interface structure.
Monitoring Point	External device monitoring point setting, identified by a unique ID. It is described as ID in the interface structure.
Remote Signaling	Monitoring point alarm information.
Telemetry	Monitoring data upload of monitoring point.

# Table of Contents

<b>Foreword</b> .....	<b>I</b>
<b>Glossary</b> .....	<b>I</b>
<b>1 Introduction</b> .....	<b>1</b>
1.1 Overview .....	1
1.2 Applicability .....	1
1.3 Field Surveillance Unit.....	2
1.4 Scenes .....	2
<b>2 Main Functions</b> .....	<b>3</b>
2.1 Initializing SDK.....	3
2.1.1 Introduction.....	3
2.1.2 Interface Overview .....	3
2.1.3 Process Description.....	3
2.1.4 Example Code .....	4
2.2 Device Login and Logout.....	7
2.2.1 Introduction.....	7
2.2.2 Interface Overview .....	7
2.2.3 Process Description.....	7
2.2.4 Example Code .....	9
2.3 Getting Device List .....	10
2.3.1 Introduction.....	10
2.3.2 Interface Overview .....	10
2.3.3 Process Description.....	11
2.3.4 Example Code .....	11
2.4 Get Device Point Information.....	13
2.4.1 Introduction.....	13
2.4.2 Interface Overview .....	13
2.4.3 Process Description.....	14
2.4.4 Example Code .....	14
2.5 Subscribing to Monitoring Point Alarm.....	15
2.5.1 Introduction.....	15
2.5.2 Interface Overview .....	15
2.5.3 Process Description.....	16
2.5.4 Example Code .....	17
2.6 Subscribing to Monitoring Point Information.....	17
2.6.1 Introduction.....	17
2.6.2 Interface Overview .....	17
2.6.3 Process Description.....	18
2.6.4 Example Code .....	18
2.7 Subscribing to General Alarm .....	19
2.7.1 Introduction.....	19
2.7.2 Interface Overview .....	20
2.7.3 Process Description.....	20
2.7.4 Example Code .....	21
<b>3 Interface Function</b> .....	<b>22</b>

3.1 Initializing SDK .....	22
3.1.1 CLIENT_Init .....	22
3.1.2 CLIENT_Cleanup .....	22
3.1.3 CLIENT_SetAutoReconnect .....	22
3.1.4 CLIENT_SetNetworkParam .....	23
3.2 Logging in and out .....	23
3.2.1 CLIENT_LoginWithHighLevelSecurity .....	23
3.2.2 CLIENT_Logout .....	23
3.3 Searching for Devices CLIENT_QueryDevState .....	24
3.4 Get Device Point Information CLIENT_SCADAGetAttributeInfo .....	24
3.5 Subscribing to Monitoring Point Alarm .....	25
3.5.1 CLIENT_SCADAAlarmAttachInfo .....	25
3.5.2 Stopping Subscription CLIENT_SCADAAlarmDetachInfo .....	25
3.6 Subscription to Real-time Information of Monitoring Point .....	25
3.6.1 CLIENT_SCADAAttachInfo .....	25
3.6.2 CLIENT_SCADADetachInfo .....	26
3.7 Reporting Alarm .....	26
3.7.1 CLIENT_SetDVRMessCallBack .....	26
3.7.2 CLIENT_StartListenEx .....	27
3.7.3 CLIENT_StopListen .....	27
<b>4 Callback .....</b>	<b>28</b>
4.1 fDisconnect .....	28
4.2 fHaveReConnect .....	28
4.3 fSCADAAlarmAttachInfoCallBack .....	28
4.4 fMessCallBack .....	29
<b>Appendix 1 Cybersecurity Recommendations .....</b>	<b>30</b>

# 1 Introduction

## 1.1 Overview

The Manual introduces SDK interfaces that include main functions, interface functions, and callbacks. The main functions include: SDK initialization, device login, device list acquisition, subscription to point information, subscription to real-time information of monitoring point, and subscription to device status alarm.

The development kit might include different files according to the environment.

- For the files included in Windows development kit, see Table 1-1.

Table 1-1 Files included in the development kit

Library Type	Library File Name	Description
Function Library	dhnetSDK.h	Header file
	dhnetSDK.lib	Lib file
	dhnetSDK.dll	Library file
	avnetSDK.dll	Library file
Configuration Library	dhconfigSDK.dll	Library file
Transcoding Library	StreamConvertor.dll	Library file

- For the files included in Linux development kit, see Table 1-2.

Table 1-2 Files included in the development kit

Library Type	Library File Name	Description
Function Library	dhnetSDK.h	Header file
	libdhnetSDK.so	Library file
	libavnetSDK.so	Library file
Configuration Library	libdhconfigSDK.so	Library file
	dhconfigSDK.h	Header file
Transcoding Library	StreamConvertor.so	Library file



- The function library and configuration library are necessary libraries.
- The function library is the main body of SDK, which is used for communication interaction between client and products, remote control, search, configuration, acquisition and processing of stream data.
- The configuration library packs and parses the structures of configuration functions.

## 1.2 Applicability

- Recommended memory: No less than 512 M.
- Jdk version: jdk1.6 and jdk1.8.
- Systems supported by SDK:
  - ◇ Windows  
Windows 10/Windows 8.1/Windows 7/vista/XP/2000 and Windows Server 2008/2003

- ◇ Linux  
Common Linux systems such as Red Hat/SUSE

## 1.3 Field Surveillance Unit

Field surveillance unit is an excellent digital surveillance product designed for power and environment surveillance. The unit uses the embedded stable LINUX operating system.

- It supports protection zone alarm input and output, and the access of 4 mA–20 mA current sensor and RS-485 bus-based sensor.
- Based on video surveillance and the general H.264 video compression technology and G.711 audio compression technology, an all-around surveillance solution with advanced control technologies and powerful network data transmission capabilities is provided to integrate alarm management, power and environment acquisition and control, video surveillance, voice talk and audio advertisement, network switching, and optical fiber ring network.
- With an embedded design, the unit has high security and reliability.
- The unit can work independently and locally or be connected to the network to form a powerful security monitoring network.
- Along with professional network video monitoring platform (network) software, the unit has powerful networking and remote monitoring capabilities.

## 1.4 Scenes

The unit can be applied to the security in various fields and departments such as energy, natural gas, mining, telecommunication, power, agriculture, transportation, intelligent communities, factories, warehouses, resources, and water conservancy facilities.

## 2 Main Functions

### 2.1 Initializing SDK

#### 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 initialization, call the SDK cleaning up interface to release resource.

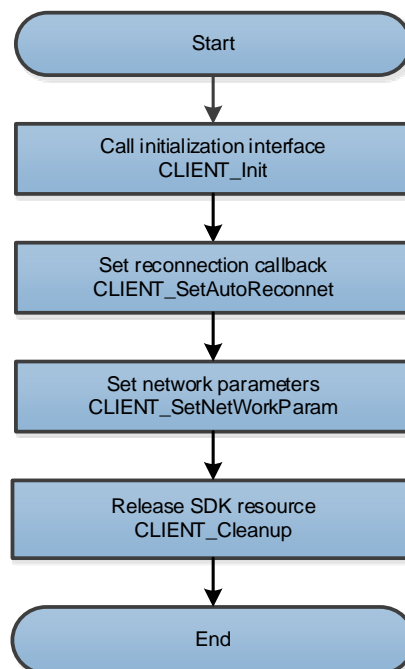
#### 2.1.2 Interface Overview

Table 2-1 Description of SDK initialization interfaces

Interface	Description
CLIENT_Init	SDK initialization interface
CLIENT_Cleanup	SDK cleaning up interface
CLIENT_SetAutoReconnect	Setting of reconnection callback interface
CLIENT_SetNetworkParam	Setting of login network environment interface

#### 2.1.3 Process Description

Figure 2-1 Process of SDK initialization





## Process Description

- Step 1 Call **CLIENT\_Init** to initialize SDK.
- Step 2 (Optional) Call **CLIENT\_SetAutoReconnect** to set reconnection callback to allow the auto reconnecting after disconnection within SDK.
- Step 3 (Optional) Call **CLIENT\_SetNetworkParam** to set network login parameters, which include the timeout period for device login and the number of attempts.
- Step 4 After using all SDK functions, call **CLIENT\_Cleanup** to release SDK resource.

## Notes

- You need to call the interfaces CLIENT\_Init and CLIENT\_Cleanup in pairs. It supports single-thread multiple calling in pairs, but it is suggested to call the pair for only one time overall.
- Initialization: Internally calling the interface CLIENT\_Init multiple times is only for internal count without repeating applying resources.
- Cleaning up: The interface CLIENT\_Cleanup clears all the opened processes, such as login, real-time monitoring, and alarm subscription.
- Reconnection: SDK can set the reconnection function for the situations such as network disconnection and power off. SDK will keep logging until succeeded. Only the real-time monitoring and playback function modules can be restored after reconnection.
- Load dynamic library: If an error such as "Unable to load library 'C:/wrongpath/libs/win64/dhnetSDK': Specified module not found" occurred when loading the dynamic library, the cause is usually path mismatch. You need to adjust the location of the dynamic library or modify the code according to the error message. This problem often occurs when packaging the engineering as a jar package to provide to other projects. Because this problem is related to the platform and engineering usage, it should be analyzed case by case. For example, when directly using the engineering on the Linux platform, you can load the dynamic library path to the dynamic library search path through the following method.
  1. Enter `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/XXX` at the terminal, and the current terminal takes effect.
  2. Modify `~/.bashrc` or `~/.bash_profile`, add `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/XXX` in the last row, and then save it. Use `source .bashrc` to execute the file, and the current user takes effect.
  3. Modify `/etc/profile`, add the same content as item 2, and then save it. Use `source` to execute the file, and all users take effect.

### 2.1.4 Example Code

```
import java.io.File;

import main.java.com.netsdk.lib.NetSDKLib;
import main.java.com.netsdk.lib.NetSDKLib.LLong;
import main.java.com.netsdk.lib.ToolKits;
```

```

import com.sun.jna.ptr.IntByReference;

/**
 * Login interface implementation
 * Main functions: Initialization, login, and logout
 */
public class LoginModule {

    public static NetSDKLib netsdk      = NetSDKLib.NETSDK_INSTANCE;
    public static NetSDKLib configsdk   = NetSDKLib.CONFIG_INSTANCE;

    // Login handle
    public static LLong m_hLoginHandle = new LLong(0);

    private static boolean blnit      = false;
    private static boolean bLogopen = false;

    // Initialization
    public static boolean init(NetSDKLib.fDisconnect disconnect, NetSDKLib.fHaveReConnect
haveReConnect) {
        blnit = netsdk.CLIENT_Init(disconnect, null);
        if(!blnit) {
            System.out.println("Initialize SDK failed");
            return false;
        }

        // Open log (optional)
        NetSDKLib.LOG_SET_PRINT_INFO setLog = new NetSDKLib.LOG_SET_PRINT_INFO();
        File path = new File("./sdklog/");
        if (!path.exists()) {
            path.mkdir();
        }
        String logPath = path.getAbsolutePath().getParent() + "\\sdklog\\" + ToolKits.getDate() +
".log";
        setLog.nPrintStrategy = 0;
        setLog.bSetFilePath = 1;
        System.arraycopy(logPath.getBytes(), 0, setLog.szLogFilePath, 0, logPath.getBytes().length);
        System.out.println(logPath);
        setLog.bSetPrintStrategy = 1;
        bLogopen = netsdk.CLIENT_LogOpen(setLog);
    }
}

```

```

        if(!bLogopen ) {
            System.err.println("Failed to open NetSDK log");
        }

        // Set the reconnection callback interface. After the reconnection callback is successfully set,
        when the device is disconnected, the SDK will automatically reconnect it.

        // This operation is optional, but it is recommended to set it.
        netsdk.CLIENT_SetAutoReconnect(haveReConnect, null);

        // Set login timeout period and number of attempts (optional).
        int waitTime = 5000; //Set the timeout period of response to login request to 5 S.
        int tryTimes = 1;    //Try to establish a link once upon login.
        netsdk.CLIENT_SetConnectTime(waitTime, tryTimes);

        // Set more network parameters, nWaittime of NET_PARAM, nConnectTryNum member,
        and CLIENT_SetConnectTime.

        // The timeout period for device login and number of attempts set by the interface have
        the same purpose, both are optional.
        NetSDKLib.NET_PARAM netParam = new NetSDKLib.NET_PARAM();
        netParam.nConnectTime = 10000;    // The timeout period for trying to establish a link
        upon login
        netParam.nGetConnInfoTime = 3000; // Set the timeout period of sub-connection.
        netsdk.CLIENT_SetNetworkParam(netParam);

        return true;
    }

    // Clear environment.
    public static void cleanup() {
        if(bLogopen) {
            netsdk.CLIENT_LogClose();
        }

        if(bInit) {
            netsdk.CLIENT_Cleanup();
        }
    }
}

```

## 2.2 Device Login and Logout

### 2.2.1 Introduction

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

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

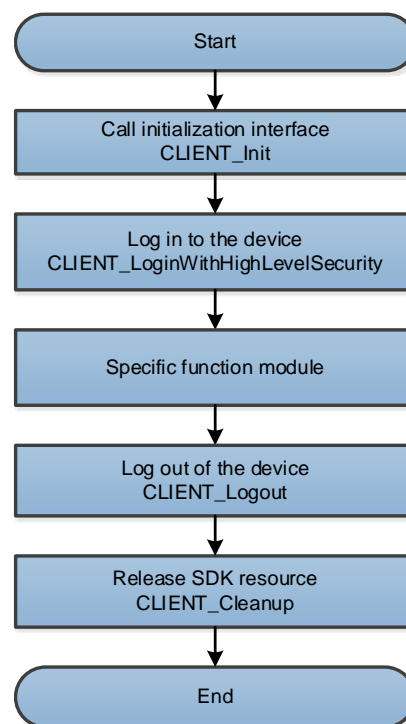
### 2.2.2 Interface Overview

Table 2-2 Description of device login interfaces

Interface	Description
CLIENT_LoginWithHighLevelSecurity	High security level login interface
CLIENT_Logout	Logout interface

### 2.2.3 Process Description

Figure 2-2 Process of login



#### Process Description

- Step 1 Call **CLIENT\_Init** to initialize SDK.
- Step 2 Call **CLIENT\_LoginWithHighLevelSecurity** to log in to the device.
- Step 3 After login, you can realize the required function module.
- Step 4 After using the function module, call **CLIENT\_Logout** to log out of the device.
- Step 5 After using SDK functions, call **CLIENT\_Cleanup** to release SDK resource.

## Notes

- Login handle: When the login is successful, the returned value of the interface is not 0 (even the handle is smaller than 0, the login is also successful). One device can log in multiple times with different handle at each login. If there is no special function module, it is suggested to log in only one time. The login handle can be repeatedly used on other function modules.
- Handle repetition: The login handle might be the same as an existing handle, which is normal. For example, if you log in to Device A and get **loginID**, then cancel **loginID**, you might get **loginID** again when you log in again. However, throughout the life cycle of a handle, the same handle will not appear.
- Logout: The interface will release the opened functions in the login session internally, but it is not suggested to rely on the cleaning up function of the logout interface. For example, if you enabled 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 releases sources once logged out.
- Login failure: It is suggested to check the failure through the error parameter (login error code) of the login interface. For common error codes, see Table 2-3.
- Login to multiple devices: After SDK initialization, you can log in to multiple devices, but the corresponding login handle and information need to be adjusted.

Table 2-3 Common error codes and meanings

Error Code	Meaning
1	Incorrect password
2	User name does not exist
3	Login timeout. The example code to avoid this error is as follows: <pre>NET_PARAM stuNetParam = new NET_PARAM(); stuNetParam.nWaittime = 8000; // unit ms CLIENT_SetNetworkParam (stuNetParam);</pre>
4	The account has been logged in
5	The account has been locked
6	The account is blacklisted
7	Out of resources, the system is busy
8	Sub-connection failed.
9	Primary connection failed.
10	Exceeded the maximum number of user connections.
11	Lack of avnetsdk or avnetsdk dependent library
12	USB flash drive is not inserted into device, or the USB flash drive information error
13	The client IP address is not authorized with login

## 2.2.4 Example Code

```
import java.io.File;

import main.java.com.netsdk.lib.NetSDKLib;
import main.java.com.netsdk.lib.NetSDKLib.LLong;
import main.java.com.netsdk.lib.ToolKits;

import com.sun.jna.ptr.IntByReference;

public class LoginModule {

    public static NetSDKLib netsdk      = NetSDKLib.NETSDK_INSTANCE;
    public static NetSDKLib configsdk   = NetSDKLib.CONFIG_INSTANCE;
    // Omit SDK initialization and SDK cleaning up
    // Device information
    public static NetSDKLib.NET_DEVICEINFO_Ex m_stDeviceInfo = new
NetSDKLib.NET_DEVICEINFO_Ex();
    // Login handle
    public static LLong m_hLoginHandle = new LLong(0);

    // Log in to the device.
    public static boolean login(String m_strIp, int m_nPort, String m_strUser, String m_strPassword)
{
    // Input parameter
    NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY pstInParam=
new NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY();
    pstInParam.szIp= m_strIp;
    pstInParam.nport= m_nPort;
    pstInParam.szUserName= m_strUser;
    pstInParam.szPassword= m_strPassword;
    // Output parameter
    NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY pstOutParam=
new NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY();
    m_hLoginHandle =
netsdk.CLIENT_LoginWithHighLevelSecurity(NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY
pstInParam, NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY pstOutParam);

    if(m_hLoginHandle.longValue() == 0) {
        System.err.printf("Login Device[%s] Port[%d]Failed. %s\n", m_strIp, m_nPort,
```

```

ToolKits.getErrorCodePrint());
    } else {
        System.out.println("Login Success ");
    }

    return m_hLoginHandle.longValue() == 0? false:true;
}

// Log out of the device
public static boolean logout() {
    if(m_hLoginHandle.longValue() == 0) {
        return false;
    }

    boolean bRet = netsdk.CLIENT_Logout(m_hLoginHandle);
    if(bRet){
        m_hLoginHandle.setValue(0);
    }

    return bRet;
}
}

```

## 2.3 Getting Device List

### 2.3.1 Introduction

Get the ID of devices that are connected to the unit.

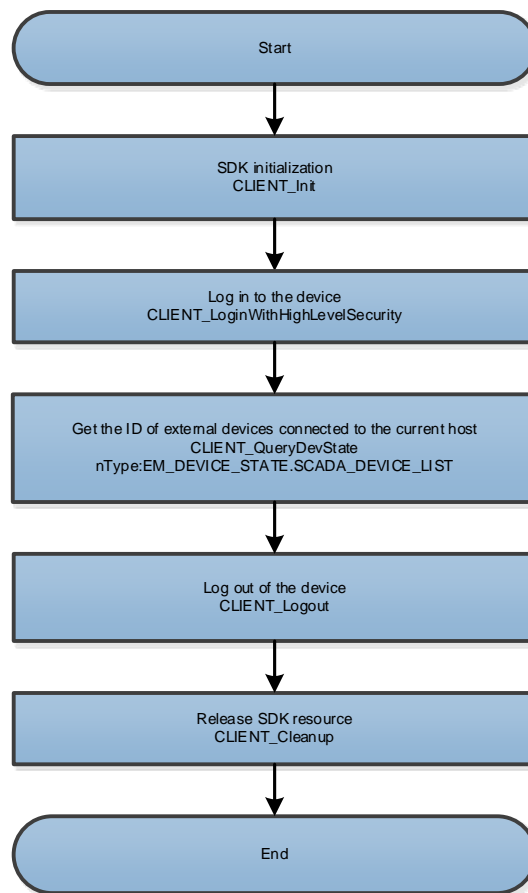
### 2.3.2 Interface Overview

Table 2-4 Description of the interface for getting device list

Interface	Description
CLIENT_QueryDevState	Get the ID of external devices that are connected to the current unit.

## 2.3.3 Process Description

Figure 2-3 Process of getting device list



### Process Description

- Step 1 Call **CLIENT\_Init** to initialize NetSDK.
- Step 2 Call **CLIENT\_LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **CLIENT\_QueryDevState** to get the device list. The parameter nType value is EM\_DEVICE\_STATE\_SCADA\_DEVICE\_LIST, and the corresponding structure is NET\_SCADA\_DEVICE\_LIST.
- Step 4 After using the function module, call **CLIENT\_Logout** to log out of the device.
- Step 5 After using NetSDK, call **CLIENT\_Cleanup** to release NetSDK resource.

## 2.3.4 Example Code

```
/**  
 * Get the ID of external devices that are connected to the current unit.  
 */  
public boolean queryDeviceIdList() {  
    m_IstDeviceID.clear(); // Clear the list of external device IDs.  
  
    int nCount = deviceinfo.byChanNum; // Total number of channels of the device
```



```

        NET_SCADA_DEVICE_ID_INFO[] stuDeviceIDList = new
NET_SCADA_DEVICE_ID_INFO[nCount];
        for (int i = 0; i < stuDeviceIDList.length; ++i) {
            stuDeviceIDList[i] = new NET_SCADA_DEVICE_ID_INFO();
        }

        NET_SCADA_DEVICE_LIST stuSCADADeviceInfo = new NET_SCADA_DEVICE_LIST();
        stuSCADADeviceInfo.nMax = nCount;
        int nSize = stuDeviceIDList[0].size() * nCount;
        stuSCADADeviceInfo.pstuDeviceIDInfo = new Memory(nSize);    // Monitoring device
information
        stuSCADADeviceInfo.pstuDeviceIDInfo.clear(nSize);
        ToolKits.SetStructArrToPointerData(stuDeviceIDList,
stuSCADADeviceInfo.pstuDeviceIDInfo);

        if (queryDevState(NetSDKLib.NET_DEVSTATE_SCADA_DEVICE_LIST, stuSCADADeviceInfo)) {
            System.err.println("Failed to get the IDs of external devices that are connected to the
current unit" + ToolKits.getErrorCode());    // Failed to call the interface
            return false;
        }

        if (stuSCADADeviceInfo.nRet == 0) {
            System.out.println("The number of valid IDs of external devices that are connected to
the current unit is 0.");    // External devices do not have valid IDs.
            return false;
        }

        // Extract data from Pointer.
        ToolKits.GetPointerDataToStructArr(stuSCADADeviceInfo.pstuDeviceIDInfo,
stuDeviceIDList);
        // Print data and update device ID.
        System.out.println("Get the number of valid IDs of external devices that are connected to
the current unit" + stuSCADADeviceInfo.nRet);
        for (int i = 0; i < stuSCADADeviceInfo.nRet; ++i) {
            String deviceId = "";
            try {
                System.out.printf("External Device [%d] Device ID [%s] Device Name [%s]\n", i,
                    new String(stuDeviceIDList[i].szDeviceID, encode).trim(),
                    new String(stuDeviceIDList[i].szDevName, encode).trim());
                deviceId = new String(stuDeviceIDList[i].szDeviceID, encode).trim();
            }

```

```

        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        m_lstDeviceID.add(deviceID); // Update the list of external device IDs.
    }
    return true;
}

```

## 2.4 Get Device Point Information.

### 2.4.1 Introduction

Get the corresponding monitoring point information according to the ID of each device on the field surveillance unit.

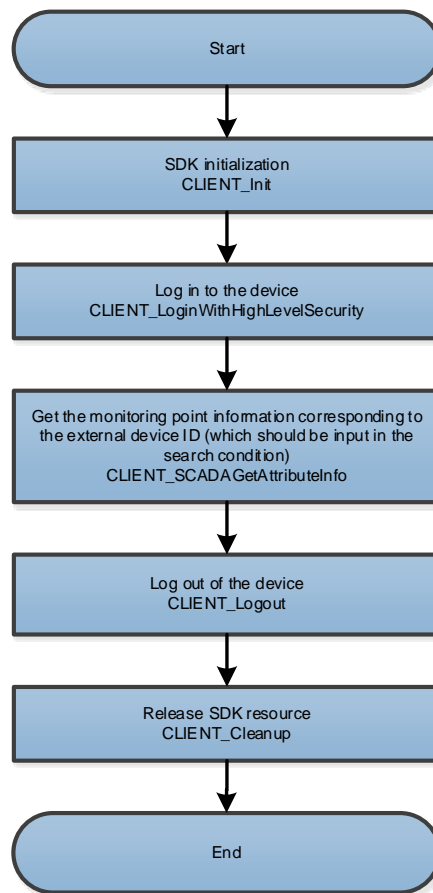
### 2.4.2 Interface Overview

Table 2-5 Description of the interface for getting monitoring point information

Interface	Description
CLIENT_SCADAGetAttributeInfo	Get device point information.

## 2.4.3 Process Description

Figure 2-4 Process of getting device point information



### Process Description

- Step 1 Call **Client\_Init** to initialize NetSDK.
- Step 2 After successful initialization, call **Client\_LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **CLIENT\_CADAGetAttributeInfo** to get the monitoring point information corresponding to the external device ID, which is obtained by inputting CLIENT\_QueryDevState in szDeviceID of the search condition stuCondition.
- Step 4 After using the function module, call **CLIENT\_Logout** to log out of the device. After using NetSDK, call **CLIENT\_Cleanup** to release NetSDK resource.

## 2.4.4 Example Code

```
public void getDeviceAttribute(){  
    NET_IN_SCADA_GET_ATTRIBUTE_INFO inParam = new  
NET_IN_SCADA_GET_ATTRIBUTE_INFO();  
    NET_OUT_SCADA_GET_ATTRIBUTE_INFO outParam = new  
NET_OUT_SCADA_GET_ATTRIBUTE_INFO();  
    NET_GET_CONDITION_INFO conditionInfo = new NET_GET_CONDITION_INFO();
```

```

        conditionInfo.szDeviceID = "A15DFG56".getBytes();
        outParam.nMaxAttributeInfoNum = 20;
        NET_ATTRIBUTE_INFO attributeInfo = new NET_ATTRIBUTE_INFO();
        attributeInfo.write();
        outParam.pstuAttributeInfo = attributeInfo.getPointer();
        inParam.write();
        outParam.write();

        boolean ret = netsdkApi.CLIENT_SCADAGetAttributeInfo(m_hLoginHandle,
inParam.getPointer(), outParam.getPointer(), 3000);
        if (ret){
            System.out.println("SCADAGetAttributeInfo succeed!");
            outParam.read();
            int retAttributeInfoNum = outParam.nRetAttributeInfoNum;
            for (int i = 0; i < retAttributeInfoNum; i++) {
                NET_ATTRIBUTE_INFO out = new NET_ATTRIBUTE_INFO();
                ToolKits.GetPointerDataToStruct(outParam.pstuAttributeInfo, 0, out);
                System.out.println(out.emStatus);
            }
        }
    }
}

```

## 2.5 Subscribing to Monitoring Point Alarm

### 2.5.1 Introduction

Monitor the alarm information of each monitoring point.

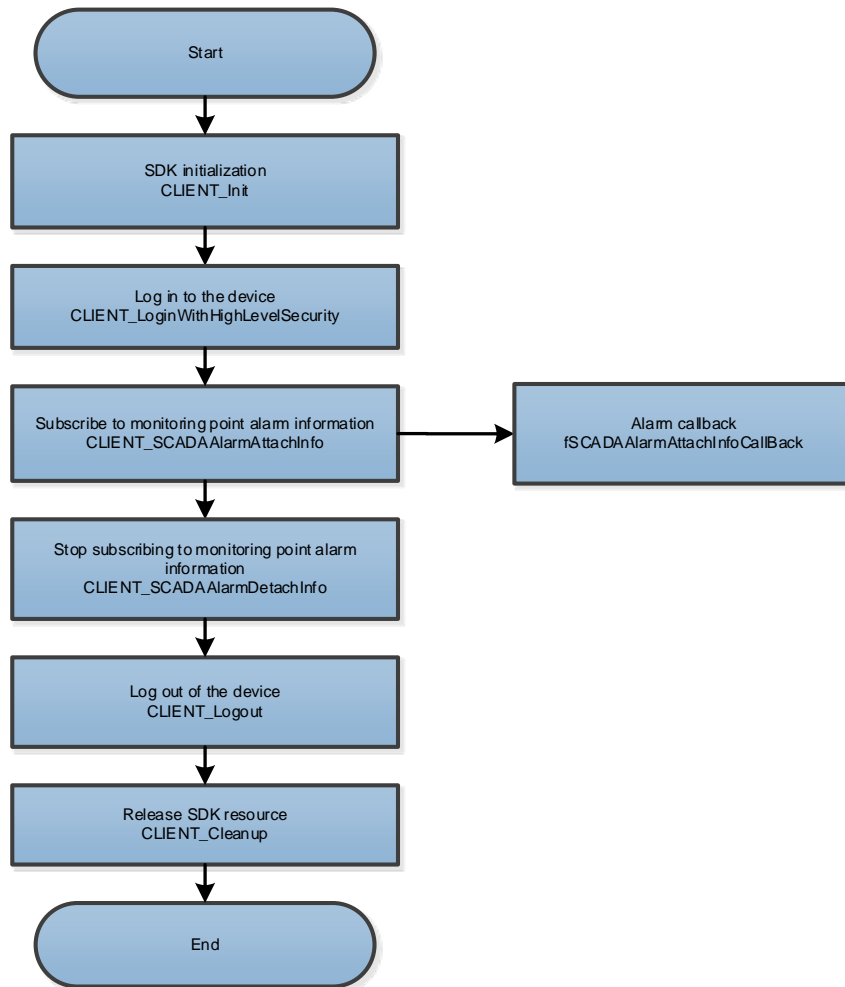
### 2.5.2 Interface Overview

Table 2-6 Description of the monitoring point alarm interface

Interface	Description
CLIENT_SCADAAAlarmAttachInfo	Subscribe to monitoring point alarm information.
CLIENT_SCADAAAlarmDetachInfo	Unsubscribe from monitoring point alarm information.

## 2.5.3 Process Description

Figure 2-5 Process of monitoring point alarm information



### Process Description

- Step 1 Call **CLIENT\_Init** to initialize NetSDK.
- Step 2 After successful initialization, call **CLIENT\_LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **CLIENT\_SCADAAlarmAttachInfo** to subscribe to alarm from the device. The corresponding input and output structures are NET\_IN\_SCADA\_ALARM\_ATTACH\_INFO and NET\_OUT\_SCADA\_ALARM\_ATTACH\_INFO. After successful subscription, alarm events reported by the device are notified to you through the callback cbCallback set in the NET\_IN\_SCADA\_ALARM\_ATTACH\_INFO structure.
- Step 4 After using the alarm reporting function, call **CLIENT\_SCADAAlarmDetachInfo** to stop subscribing to alarm from the device.
- Step 5 After using the function module, call **CLIENT\_Logout** to log out of the device.
- Step 6 After using NetSDK, call **CLIENT\_Cleanup** to release NetSDK resource.

## 2.5.4 Example Code

```
/**
 * Subscribe to monitoring point alarm information.
 */
public void scadaAlarmAttach() {
    // Input parameter
    NET_IN_SCADA_ALARM_ATTACH_INFO stIn = new
NET_IN_SCADA_ALARM_ATTACH_INFO();
    stIn.cbCallBack = SCADAAlarmAttachInfoCallBack.getInstance();

    // Output parameter
    NET_OUT_SCADA_ALARM_ATTACH_INFO stOut = new
NET_OUT_SCADA_ALARM_ATTACH_INFO();

    m_hAlarmAttachHandle = netsdkApi.CLIENT_SCADAAlarmAttachInfo(m_hLoginHandle,
stIn, stOut, 3000);
    if (m_hAlarmAttachHandle.longValue() != 0) {
        System.out.println("Successfully subscribed to monitoring point alarm information.");
    } else {
        System.err.println("Failed to subscribe to monitoring point alarm information." +
ToolKits.getErrorCode());
    }
}
```

## 2.6 Subscribing to Monitoring Point Information

### 2.6.1 Introduction

Monitor the information upload of each monitoring point.

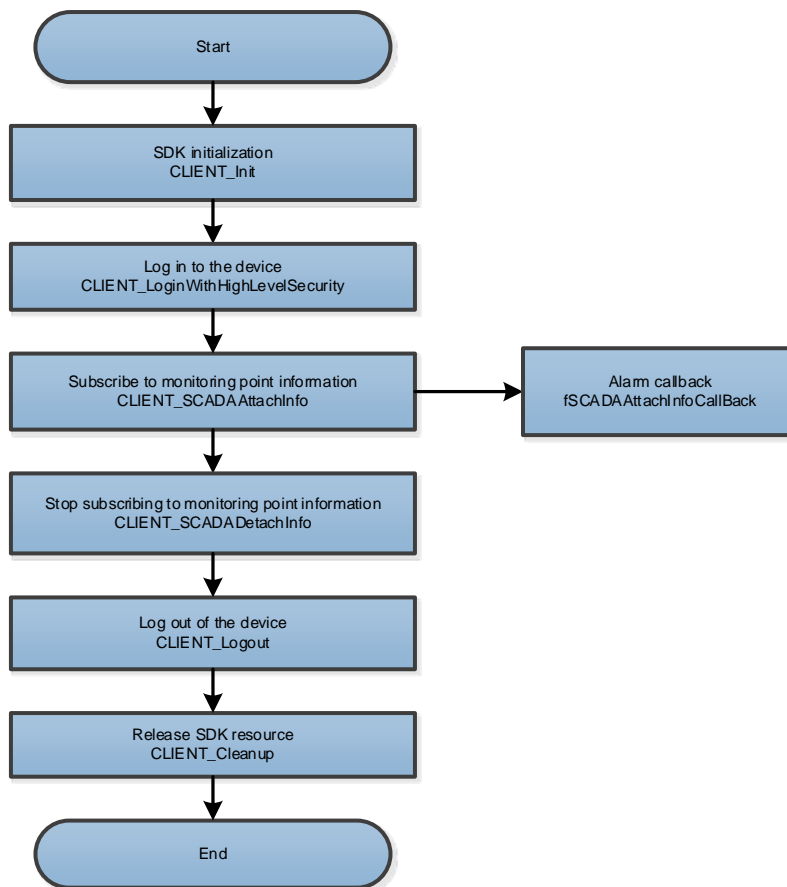
### 2.6.2 Interface Overview

Table 2-7 Monitoring point information

Interface	Description
CLIENT_SCADAAttachInfo	Subscribing to Monitoring Point Information.
CLIENT_SCADADetachInfo	Unsubscribe from monitoring point information.

## 2.6.3 Process Description

Figure 2-6 Process of monitoring point information



### Process Description

- Step 1 Call **CLIENT\_Init** to initialize NetSDK.
- Step 2 After successful initialization, call **CLIENT\_LoginWithHighLevelSecurity** to log in to the device.
- Step 3 Call **CLIENT\_SCADAAttachInfo** to subscribe to alarm from the device. The corresponding input and output structures are NET\_IN\_SCADA\_ATTACH\_INFO and NET\_OUT\_SCADA\_ATTACH\_INFO. After successful subscription, alarm events reported by the device are notified to you through the callback cbCallBack set in the NET\_IN\_SCADA\_ATTACH\_INFO structure.
- Step 4 After using the alarm reporting function, call **CLIENT\_SCADADetachInfo** to stop subscribing to alarm from the device.
- Step 5 After using the function module, call **CLIENT\_Logout** to log out of the device.
- Step 6 After using NetSDK, call **CLIENT\_Cleanup** to release NetSDK resource.

## 2.6.4 Example Code

```
/**  
 * Subscribe to real-time information of monitoring point  
 */
```

```

public void case_scada_real_attach() {
    // Input parameter
    NET_IN_SCADA_ATTACH_INFO stIn = new NET_IN_SCADA_ATTACH_INFO();
    stIn.cbCallBack = SCADAAttachInfoCallBack.getInstance();
    stIn.emPointType = EM_NET_SCADA_POINT_TYPE.EM_NET_SCADA_POINT_TYPE_ALL;

    // Output parameter
    NET_OUT_SCADA_ATTACH_INFO stOut = new NET_OUT_SCADA_ATTACH_INFO();

    m_hAttachHandle = netsdkApi.CLIENT_SCADAAttachInfo(m_hLoginHandle, stIn, stOut,
3000);
    if (0 != m_hAttachHandle.longValue()) {
        System.out.println("Successfully subscribed to real-time data of monitoring point.");
    } else {
        System.err.println("Failed to subscribe to real-time data of monitoring point." +
ToolKits.getErrorCode());
    }
}

/**
 * Cancel real-time information of monitoring point
 */
public void case_scada_real_detach() {
    if (m_hAttachHandle.longValue() != 0) {
        netsdkApi.CLIENT_SCADADetachInfo(m_hAttachHandle);
        m_hAttachHandle.setValue(0);
    }
}
}

```

## 2.7 Subscribing to General Alarm

### 2.7.1 Introduction

Alarm reporting method: Use NetSDK to log in to the device and subscribe to the alarm function from the device. When the device detects the alarm event, it will send the event to NetSDK immediately. You can get the corresponding alarm information through the alarm callback.



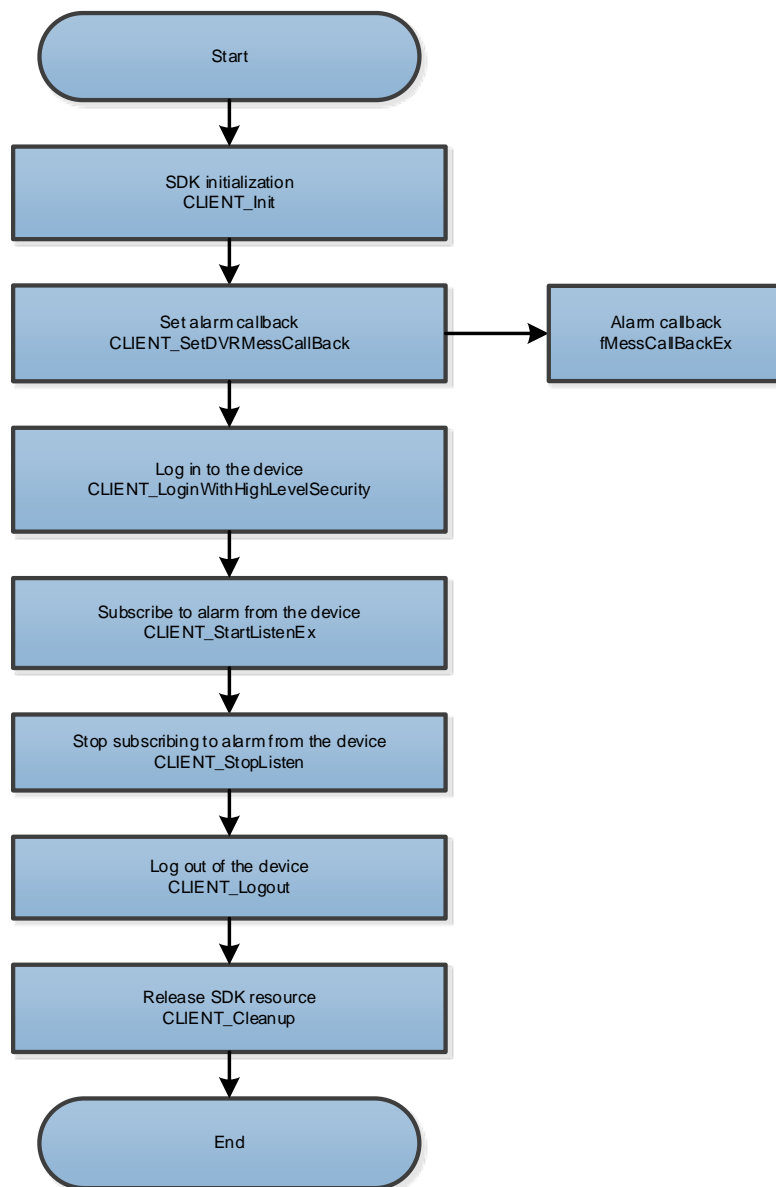
## 2.7.2 Interface Overview

Table 2-8 Description of alarm reporting interfaces

Interface	Description
CLIENT_SetDVRMessCallBack	Set alarm callback interface.
CLIENT_StartListenEx	Extension interface for subscribing to alarm.
CLIENT_StopListen	Stop subscribing to alarm.

## 2.7.3 Process Description

Figure 2-7 Process of alarm reporting



### Process Description

- Step 1 Call **CLIENT\_Init** to initialize NetSDK.
- Step 2 Call **CLIENT\_SetDVRMessCallBack** to set alarm callback. This interface should be called before alarm subscription.

- Step 3 Call **CLIENT\_LoginWithHighLevelSecurity** to log in to the device.
- Step 4 Call **CLIENT\_StartListenEx** to subscribe to alarm from the device. After successful subscription, alarm events reported by the device are notified to you through the callback set in the **CLIENT\_SetDVRMessCallBack** function.
- Step 5 After using the alarm reporting function, call **CLIENT\_StopListen** to stop subscribing to alarm from the device.
- Step 6 Call **CLIENT\_Logout** to log out of the device.
- Step 7 After using NetSDK, call **CLIENT\_Cleanup** to release NetSDK resource.

## 2.7.4 Example Code

```
/**
 * Subscribe to alarm information
 */
public boolean case_StartListenEx() {
    if (bListening) {
        return true;
    }

    // Subscribe to alarm (the alarm callback has been set in initSdk).
    bListening = netsdkApi.CLIENT_StartListenEx(m_hLoginHandle);
    if (!bListening) {
        System.err.println("Failed to subscribe to alarm. LastError = 0x%x\n" +
netsdkApi.CLIENT_GetLastError());
    } else {
        System.out.println("Successfully subscribed to alarm.");
    }
    return bListening;
}

/**
 * Unsubscribe from alarm information
 */
public void case_stopListen() {
    if (bListening) {
        System.out.println("Unsubscribe from alarm information.");
        netsdkApi.CLIENT_StopListen(m_hLoginHandle);
        bListening = false;
    }
}
```

## 3 Interface Function

### 3.1 Initializing SDK

#### 3.1.1 CLIENT\_Init

Table 3-1 SDK Initialization CLIENT\_Init

Item	Description	
Description	Initialize the SDK	
Methods	public boolean CLIENT_Init( Callback cbDisconnect, Pointer dwUser);	
Parameter	[in]cbDisconnect	Disconnection callback
	[in]dwUser	User parameters for disconnection callback
Return Value	<ul style="list-style-type: none"><li>• Success: TRUE</li><li>• Failure: FALSE</li></ul>	
Description	<ul style="list-style-type: none"><li>• Prerequisite for calling other functions of the network SDK</li><li>• When the callback is set as NULL, the device will not be called back to the user after disconnection</li></ul>	

#### 3.1.2 CLIENT\_Cleanup

Table 3-2 SDK Cleaning UP CLIENT\_Cleanup

Item	Description
Description	Clean up SDK
Methods	public void CLIENT_Cleanup();
Parameter	None
Return Value	None
Description	SDK cleaning up interface is finally called before the end

#### 3.1.3 CLIENT\_SetAutoReconnect

Table 3-3 Set Reconnection Callback CLIENT\_SetAutoReconnect

Item	Description	
Description	Set auto reconnection callback	
Methods	public void CLIENT_SetAutoReconnect( Callback cbAutoConnect, Pointer dwUser);	
Parameter	[in]cbAutoConnect	Reconnection callback
	[in]dwUser	User parameters for reconnection callback
Return Value	None	
Description	Set reconnection callback interface. If the callback is set as NULL, the device will not be reconnected automatically	

### 3.1.4 CLIENT\_SetNetworkParam

Table 3-4 Set Network Parameter CLIENT\_SetNetworkParam

Item	Description	
Description	Set related parameters of network environment	
Methods	public void CLIENT_SetNetworkParam( NET_PARAM pNetParam);	
Parameter	[in]pNetParam	Network delay, number of reconnections, buffer size and other parameters
Return Value	None	
Description	You can adjust parameters according to the actual network environment	

## 3.2 Logging in and out

### 3.2.1 CLIENT\_LoginWithHighLevelSecurity

Table 3-5 Log in to device CLIENT\_LoginWithHighLevelSecurity

Item	Description	
Description	The user logs in to the device	
Methods	public LLong CLIENT_LoginWithHighLevelSecurity( NET_IN_LOGIN_WITH_HIGHLEVEL_SECURITY pstInParam, NET_OUT_LOGIN_WITH_HIGHLEVEL_SECURITY pstOutParam);	
Parameter	[in]pstInParam	Input parameter
	[out]pstOutParam	Output parameter
Return Value	<ul style="list-style-type: none"><li>• Success: Login handle</li><li>• Failure: 0</li></ul>	
Description	This method is packaged in the NetSDKLib interface and called as follows: m_hLoginHandle = netsdk.CLIENT_CLIENT_LoginWithHighLevelSecurity(pstInParam, pstOutParam);	

### 3.2.2 CLIENT\_Logout

Table 3-6 Log out of device CLIENT\_Logout

Item	Description	
Description	The user logs out of the device	
Methods	public boolean CLIENT_Logout(LLong lLoginID);	
Parameter	[in]lLoginID	Return value of CLIENT_LoginWithHighLevelSecurity
Return Value	<ul style="list-style-type: none"><li>• Success: TRUE</li><li>• Failure: FALSE</li></ul>	
Description	This method is packaged in the NetSDKLib interface and called as follows: netsdk.CLIENT_Logout(m_hLoginHandle);	

### 3.3 Searching for Devices CLIENT\_QueryDevState

Table 3-7 Search for list of monitoring devices connected CLIENT\_QueryDevState

Item	Description	
Description	Search for the information of monitoring devices that are connected to the unit	
Methods	public boolean CLIENT_QueryDevState(LLong ILoginID, int nType, Pointer pBuf, int nBufLen, IntByReference pRetLen, int waittime);	
Parameter	[in]ILoginID	Return value of CLIENT_LoginWithHighLevelSecurity
	[in]nType	Search for information type, corresponding to NET_DEVSTATE_SCADA_DEVICE_LIST
	[out]pBuf	Memory requested by the user
	[out]nBufLen	Size of requested memory
	[in]pRetLen	Output parameter: Cache for receiving data returned by the search, corresponding to the NET_SCADA_DEVICE_LIST type
	[in]waittime	Get timeout period
Return Value	<ul style="list-style-type: none"> <li>• Success: TRUE</li> <li>• Failure: FALSE</li> </ul>	
Description	Search for the information of monitoring devices that are connected to the unit	

### 3.4 Get Device Point Information CLIENT\_SCADAGetAttributeInfo

Table 3-8 Get point threshold CLIENT\_SCADAGetAttributeInfo

Item	Description	
Description	Get point threshold	
Methods	public boolean CLIENT_SCADAGetAttributeInfo(LLong ILoginID, Pointer pInParam, Pointer pOutParam, int nWaitTime);	
Parameter	[in]ILoginID	Return value of CLIENT_LoginWithHighLevelSecurity
	[in]pInParam	Input parameter, corresponding to the NET_IN_SCADA_GET_ATTRIBUTE_INFO type
	[out]pOutParam	Output parameter, corresponding to the NET_OUT_SCADA_GET_ATTRIBUTE_INFO type
	[in]nWaitTime	Timeout period, in ms
Return Value	<ul style="list-style-type: none"> <li>• Success: TRUE</li> <li>• Failure: FALSE</li> </ul>	
Description	None	

## 3.5 Subscribing to Monitoring Point Alarm

### 3.5.1 CLIENT\_SCADAAlarmAttachInfo

Table 3-9 Enable subscription CLIENT\_SCADAAlarmAttachInfo

Item	Description	
Description	Start subscription to intelligent event	
Methods	public LLong CLIENT_SCADAAlarmAttachInfo(LLong ILoginID, NET_IN_SCADA_ALARM_ATTACH_INFO pInParam, NET_OUT_SCADA_ALARM_ATTACH_INFO pOutParam, int nWaitTime);	
Parameter	[in] ILoginID	Return value of CLIENT_LoginWithHighLevelSecurity
	[in] pInParam	Input parameter
	[out] pOutParam	Output parameter
	[in] nWaitTime	Timeout period, in ms
Return Value	<ul style="list-style-type: none"><li>Success: Subscription handle of LLong type</li><li>Failure: 0</li></ul>	
Description	If the interface fails to return, call <b>CLIENT_GetLastError</b> to get the error code	

### 3.5.2 Stopping Subscription CLIENT\_SCADAAlarmDetachInfo

Table 3-10 Unsubscribe from CLIENT\_SCADAAlarmDetachInfo

Item	Description	
Description	Stop subscribing to intelligent event	
Methods	public boolean CLIENT_SCADAAlarmDetachInfo(LLong IAttachHandle);	
Parameter	[in] IAttachHandle	Intelligent event subscription handle
Return Value	BOOL type <ul style="list-style-type: none"><li>Success: TRUE</li><li>Failure: FALSE</li></ul>	
Description	If the interface fails to return, call <b>CLIENT_GetLastError</b> to get the error code	

## 3.6 Subscription to Real-time Information of Monitoring Point

### 3.6.1 CLIENT\_SCADAAttachInfo

Table 3-11 Subscribe to real-time information of monitoring point CLIENT\_SCADAAttachInfo

Item	Description	
Description	Subscribing to real-time information of monitoring point	
Function	public LLong CLIENT_SCADAAttachInfo(LLong ILoginID, NET_IN_SCADA_ATTACH_INFO pInParam, NET_OUT_SCADA_ATTACH_INFO pOutParam, int nWaitTime);	
Parameter	[in] ILoginID	Return value of LoginWithHighLevelSecurity

Item	Description	
	[in]pInParam	Subscription input parameter
	[out]pOutParam	Subscription output parameter
	[in]waittime	Waiting time
Return Value	<ul style="list-style-type: none"> <li>Success: Non-0 value</li> <li>Failure: 0</li> </ul>	
Description	None	

### 3.6.2 CLIENT\_SCADADetachInfo

Table 3-12 Unsubscribe from Point Information CLIENT\_SCADADetachInfo

Item	Description	
Description	Unsubscribe from point information	
Function	public boolean CLIENT_SCADADetachInfo(LLong lAttachHandle);	
Parameter	[in]lAttachHandle	Return value of CLIENT_SCADAAttachInfo
Return Value	<ul style="list-style-type: none"> <li>Success: TRUE</li> <li>Failure: FALSE</li> </ul>	
Description	None	

## 3.7 Reporting Alarm

### 3.7.1 CLIENT\_SetDVRMessCallBack

Table 3-13 Set alarm callback CLIENT\_SetDVRMessCallBack

Item	Description	
Description	Set alarm callback	
Function	public void CLIENT_SetDVRMessCallBack(Callback cbMessage, Pointer dwUser);	
Parameter	[in]cbMessage	<ul style="list-style-type: none"> <li>Message callback, which can call back the device status, such as alarm status</li> <li>When the value is set as 0, it means callback is forbidden</li> </ul>
	[in]dwUser	User-defined data
Return Value	None	
Description	<ul style="list-style-type: none"> <li>Set device message callback to get the current device status information; this function is independent of the calling sequence, and the NetSDK is not called back by default</li> <li>The callback fMessCallBack must call the alarm message subscription interface StartListenEx first before it takes effect</li> </ul>	

### 3.7.2 CLIENT\_StartListenEx

Table 3-14 Subscribe to alarm CLIENT\_StartListenEx

Item	Description	
Description	Subscribe to alarm	
Function	public boolean CLIENT_StartListenEx(LLong ILoginID);	
Parameter	[in]ILoginID	Return value of LoginWithHighLevelSecurity
Return Value	<ul style="list-style-type: none"><li>• Success: TRUE</li><li>• Failure: FALSE</li></ul>	
Description	Subscribe to device message, and the message received is called back from the set value of SetDVRMessCallBack	

### 3.7.3 CLIENT\_StopListen

Table 3-15 Stop subscribing to alarm CLIENT\_StopListen

Item	Description	
Description	Stop subscribing to alarm	
Function	public boolean CLIENT_StopListen(LLong ILoginID);	
Parameter	[in]ILoginID	Return value of LoginWithHighLevelSecurity
Return Value	<ul style="list-style-type: none"><li>• Success: TRUE</li><li>• Failure: FALSE</li></ul>	
Description	None	



# 4 Callback

## 4.1 fDisConnect

Table 4-1 Disconnection callback fDisConnect

Item	Description	
Description	Disconnection callback	
Function	<pre>public interface fDisConnect extends StdCallCallback {     public void invoke(         LLong ILoginID,         String pchDVRIP, int nDVRPort,         Pointer dwUser); }</pre>	
Parameter	[out]ILoginID	Return value of Client.LoginWithHighLevelSecurity
	[out]pchDVRIP	Disconnected device IP
	[out]nDVRPort	Disconnected device port
	[out]dwUser	User parameters for callback
Return Value	None	
Description	None	

## 4.2 fHaveReConnect

Table 4-2 Reconnection callback fHaveReConnect

Item	Description	
Description	Reconnection callback	
Function	<pre>public interface fHaveReConnect extends StdCallCallback {     public void invoke(         LLong ILoginID,         String pchDVRIP, int nDVRPort,         Pointer dwUser); }</pre>	
Parameter	[out]ILoginID	Return value of Client.LoginWithHighLevelSecurity
	[out]pchDVRIP	Reconnected device IP
	[out]nDVRPort	Reconnected device port
	[out]dwUser	User parameters for callback
Return Value	None	
Description	None	

## 4.3 fSCADAAlarmAttachInfoCallBack

Table 4-3 Data callback fSCADAAlarmAttachInfoCallBack

Item	Description	
Description	Telemetry callback	
Function	<pre>public interface fSCADAAlarmAttachInfoCallBack extends Callback {     public void invoke(LLong IAttachHandle,         NET_SCADA_NOTIFY_POINT_ALARM_INFO_LIST pInfo, int nBufLen, Pointer         dwUser); }</pre>	
Parameter	[out]IAttachHandle	Return value of Client_SCADAAttachInfo
	[out]pInfo	Alarm message data
	[out]nBuffer	Address of alarm message data block
	[out]dwUser	User parameters for callback
Return Value	None	
Description	None	

## 4.4 fMessCallBack

Table 4-4 Alarm callback fMessCallBack

Item	Description	
Description	alarm callback	
Function	<pre>public interface fMessCallBack extends Callback{     public boolean invoke(int ICommand, LLong ILoginID, Pointer pStuEvent,         int dwBufLen, String strDeviceIP, NativeLong nDevicePort, Pointer dwUser); }</pre>	
Parameter	[out]ILoginID	Return value of Client.LoginWithHighLevelSecurity
	[out]Command	Monitor and collect alarm events of devices (corresponding structure ALARM_SCADA_DEV_INFO)
	[out]pStuEvent	Monitor and collect alarm events of devices
	[out]nBufLen	Length of reported information data block, in bytes
	[out]nDeviceIP	Device IP
	[out]nDevicePort	Device port
	[out]dwUser	User parameters for callback
Return Value	None	
Description	None	

# 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 device 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 device (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 device 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 device network security:**

### **1. Physical Protection**

We suggest that you perform physical protection to device, especially storage devices. For example, place the device 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 device (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 device 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. MAC Address Binding**

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

## **8. Assign Accounts and Privileges Reasonably**

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

## **9. 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.

## **10. 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.

## **11. Secure Auditing**

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

## **12. Network Log**

Due to the limited storage capacity of the device, 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.

## **13. Construct a Safe Network Environment**

In order to better ensure the safety of device 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.
- Enable IP/MAC address filtering function to limit the range of hosts allowed to access the device.