

NetSDK_JAVA Auto Register

User's Manual








Foreword

General

This document describes the basic process of Demo demonstration and development of NetSDK JAVA auto register.

Safety Instructions

The following symbols may appear in this document and their meanings are as follows:

Sign	Description
 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.	October 2020

Table of Contents

Foreword	I
1 Introduction	1
1.1 Overview	1
1.2 Process of NetSDK Auto Register.....	1
2 Use of Java Auto Register Demo	2
3 Auto Register Interfaces	5
3.1 Configure Device Auto Register Information.....	5
3.1.1 Introduction.....	5
3.1.2 Interface Overview	5
3.1.3 Process Description	6
3.1.4 Sample Code	7
3.2 Enable/Stop Auto Register Service	9
3.2.1 Introduction.....	9
3.2.2 Interface Overview	10
3.2.3 Process Description	10
3.2.4 Sample Code	11
Appendix 1 Cybersecurity Recommendations	14

1 Introduction

1.1 Overview

NetSDK auto register is used to fix problems like devices in the intranet cannot be found by servers in the extranet, or device IP addresses being dynamic (when devices connected to 4G or Wi-Fi network, IP addresses might be dynamic). It is helpful for servers to monitor device configuration so that applications in different scenes can be realized or recovered.

NetSDK auto register function includes three aspects:

- Get and configure auto register info of target devices
- Monitoring the start-up and stop of servers.
- Quickly add and delete devices that are under monitoring of the server.

In addition, Demo of Java version provides the following functions: device info importing and exporting, stream pulling, voice call, and more.

1.2 Process of NetSDK Auto Register

Figure 1-1 The basic process of auto register

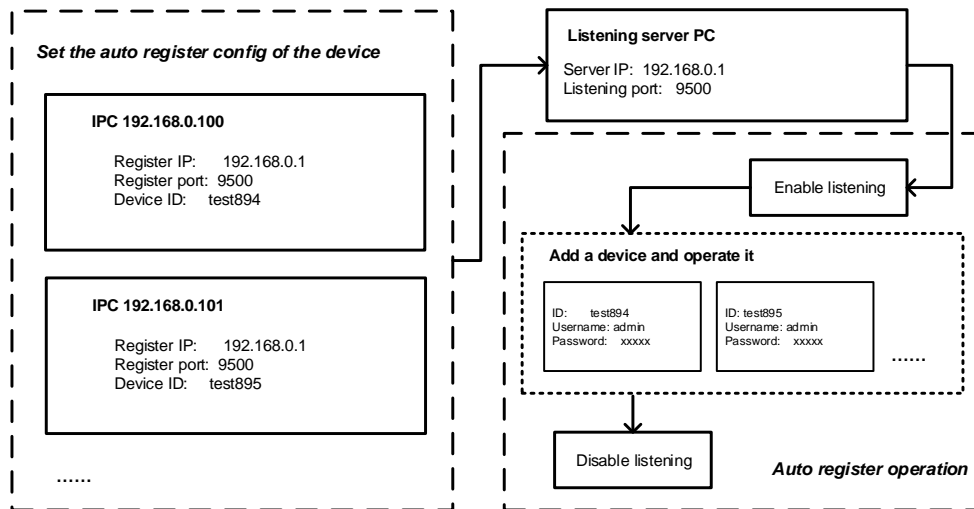


Table 1-1 Main interface

No.	Functions	Description
1	Get and set the auto register config of the device	The device has two auto register config methods: Web config and SDK config. You can use SDK to get and set the auto register config of devices with fixed IP. For details, see "2 Use of Java Auto Register Demo."
2	Start and stop server listening.	SDK provides some basic functions for auto register listening, including starting and stopping listening.

2 Use of Java Auto Register Demo

The device auto registers to the platform. The auto register is supported by Demo or on the device web. This manual takes auto register by Demo as an example.

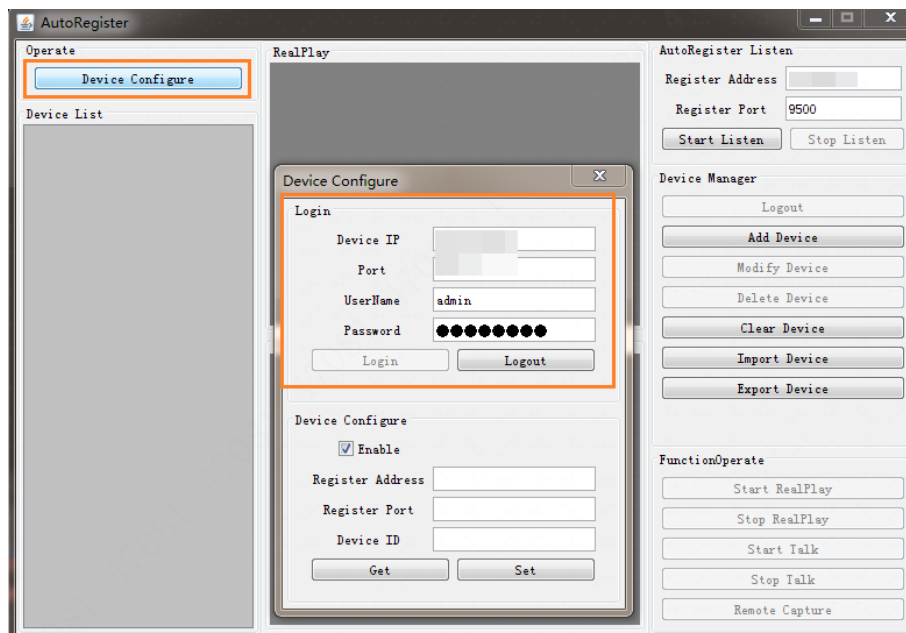


Please configure auto register on the web interface if the platform cannot access to device due to network restriction. For details of configuring auto register on web, refer to device user's manual.

Step 1 Start Demo. Run AutoRegister.java to open the auto registration Demo.

Step 2 Log in to device. Click **Device Configure**, and then enter device IP, port, user name, and password. Click **Login**.

Figure 2-1 Log in to device



Step 3 Get and configure auto register.

- 1) Click Get to get auto register config information.
- 2) (Optional) If the obtained config information is not the platform that the device should register to, please modify to the platform information.



Device ID is randomly set but cannot be repeated.


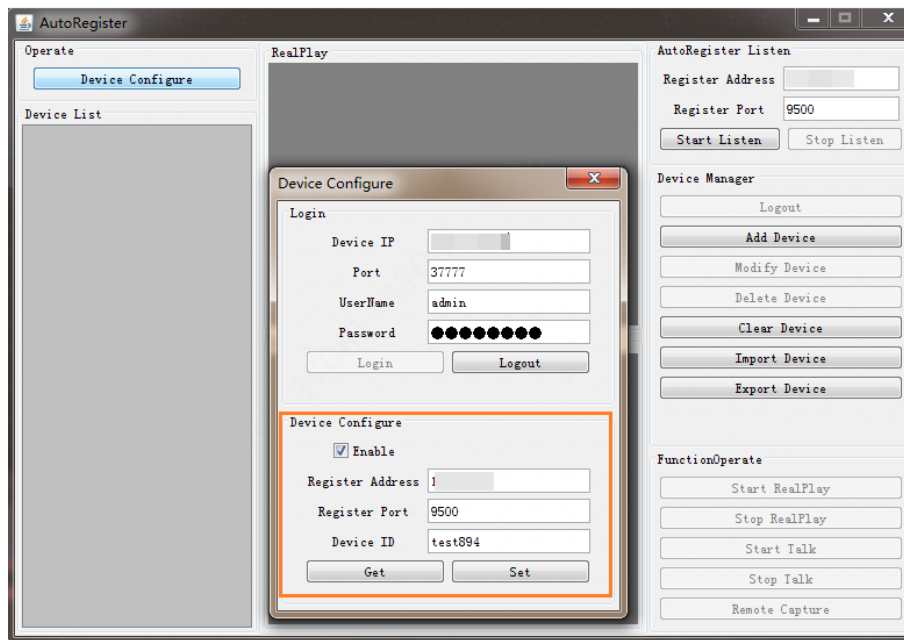
- 3) Select the **Enable** check box, and then click **Set**.
The platform sends auto register information to device, and the auto registration is succeeded.
- 4) Click Logout, and then click  of pop-up window to log out.

Figure 2-2 Get and config auto register



Step 4 Configure **Register Address** and **Register Port**, and then click **Start Listen** to start auto register listening.

Step 5 Click **Add Device** to open the corresponding window. Enter device ID, user name, and password, and then click **Add** to add a device. Wait for the device to get registered to the platform. When the device shows green and the channel can be viewed, it means the device has registered to the platform successfully.

Figure 2-3 Start listening and add device to device list

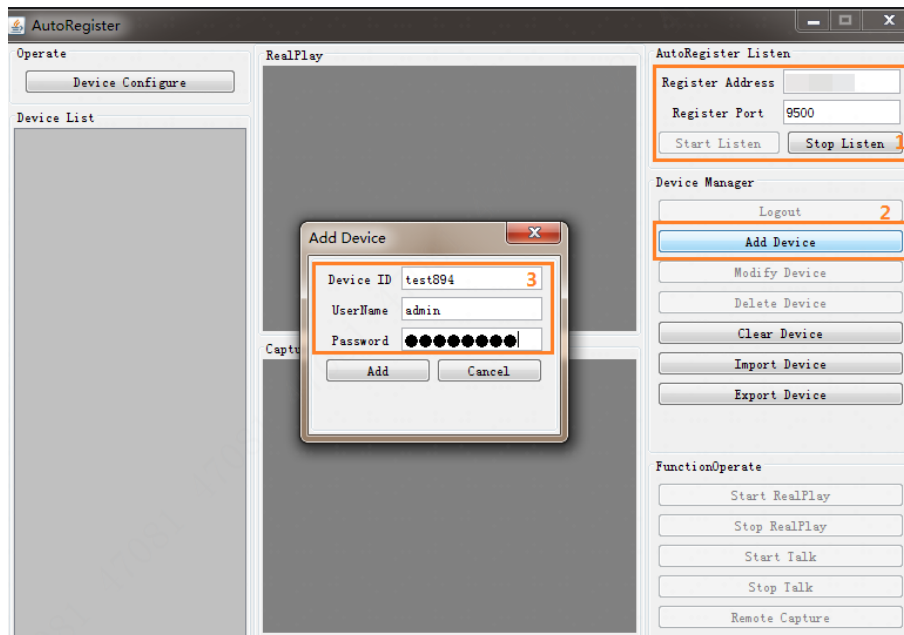
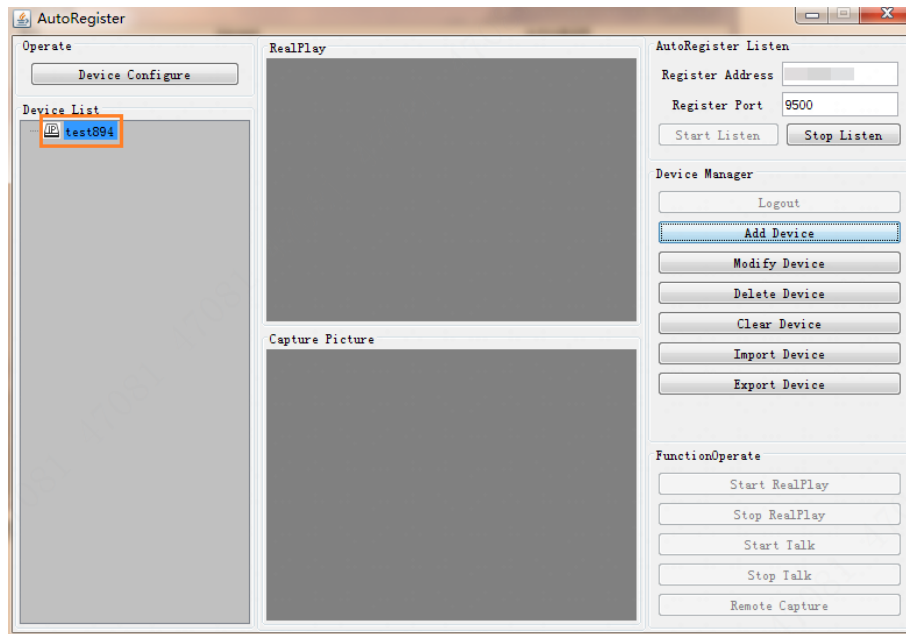
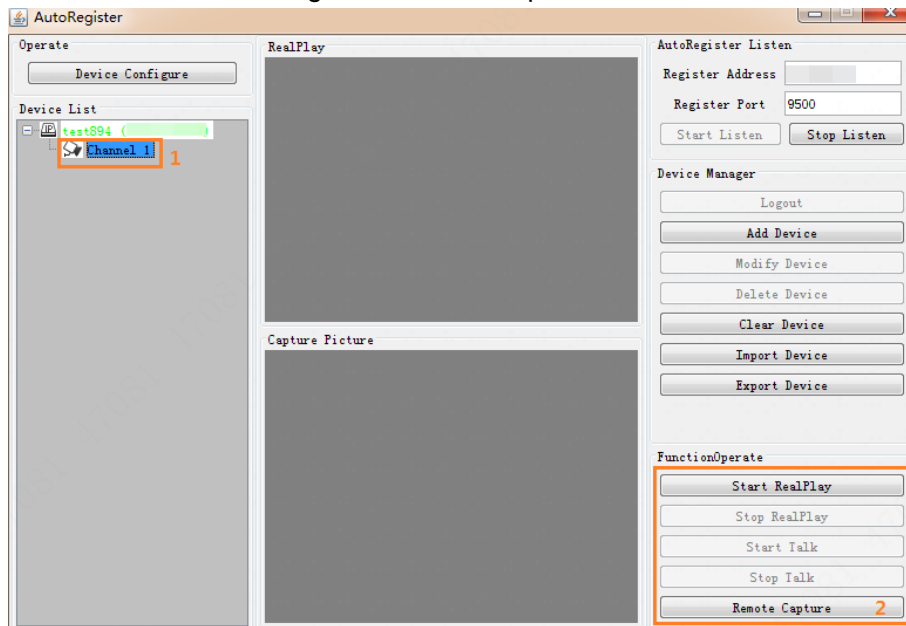


Figure 2-4 Successfully added



Step 6 Operate device. Select a channel of the device and then do the operations as instructed in the Function Operate area, such as live view, snapshot, and calling with device.

Figure 2-5 Device operations



3 Auto Register Interfaces

3.1 Configure Device Auto Register Information

3.1.1 Introduction

Users call SDK to configure device auto register info, including enable auto register, device ID, platform IP address and more.

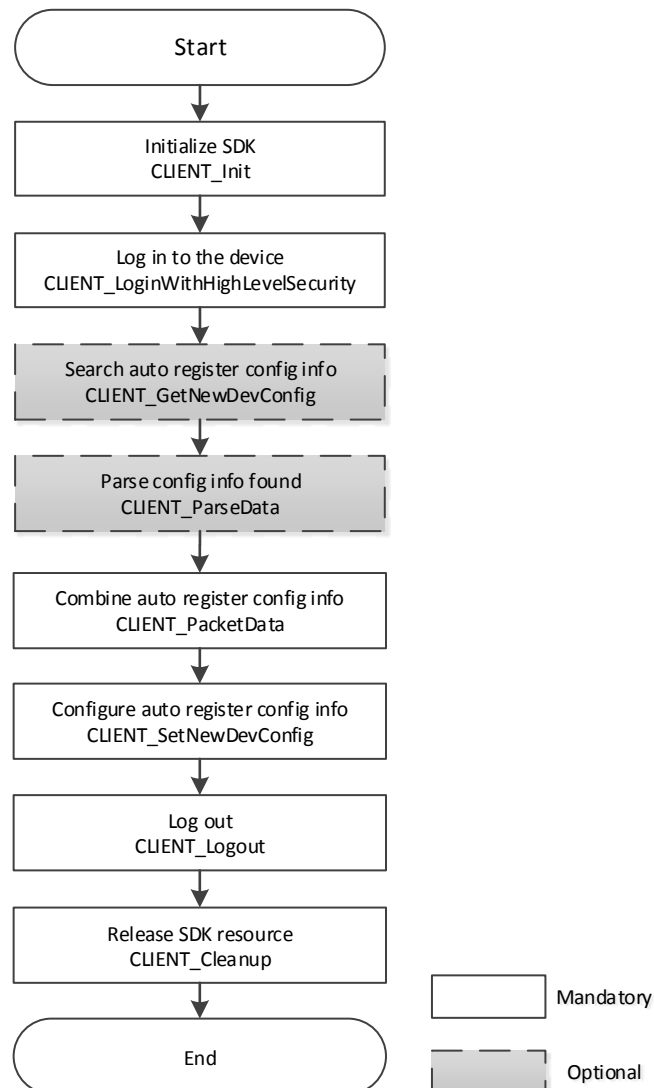
3.1.2 Interface Overview

Table 3-1 Interface description

Interface	Description
CLIENT_Init	Initialize SDK
CLIENT_LoginWithHighLevelSecurity	Login of high security level
CLIENT_GetNewDevConfig	Get config info
CLIENT_ParseData	Parse config info
CLIENT_PacketData	Combine config info to be sent
CLIENT_SetNewDevConfig	Send config info
CLIENT_Logout	Log out
CLINET_Cleanup	Clear SDK resource

3.1.3 Process Description

Figure 3-1 Auto register configuration process



Process

- Step 1 Call **CLIENT_Init** to initialize SDK.
- Step 2 Call **CLIENT_LoginWithHighLevelSecurity** interface to log in to the device.
- Step 3 (Optional) Call **CLIENT_GetNewDevConfig** interface to get device auto register config info and call **CLIENT_ParseData** interface to parse config info.
- Step 4 Call **CLIENT_PacketData** to combine configured auto register info.
- Step 5 Call **CLIENT_SetNewDevConfig** to send auto register config info.
- Step 6 Call **CLIENT_Logout** interface to log out.
- Step 7 Call **CLIENT_Cleanup** interface to release SDK resource.

Notes

- **CLIENT_Init** and **CLIENT_Cleanup** must be called in pair, and single thread being called multiple times in pairs is supported. It is recommended that the single thread be called once globally. Do not initialize or clean resources repeatedly.
- You can program parameters and send the config, but you had better change the config on the basis of original config to avoid key fields missing.
- Figure out whether the device support auto register function or not.
- Before get/configure auto register, complete SDK basic operations including initializing and logging in to the target device.

3.1.4 Sample Code

```
// For general codes, such as initialization and cleanup, please refer to 《NetSDK_JAVA Programming Manual》
```

```
//Get and write auto register config
```

```
//Get auto register config information
```

```
/**
```

```
 * Get network protocol
```

```
 * @param m_hLoginHandle Login handle
```

```
 * @return
```

```
 */
```

```
public static CFG_DVRIP_INFO getDVRIPConfig(LLong m_hLoginHandle) {
```

```
    CFG_DVRIP_INFO msg = new CFG_DVRIP_INFO();
```

```
    if(!ToolKits.GetDevConfig(m_hLoginHandle, -1, NetSDKLib.CFG_CMD_DVRIP, msg)){
```

```
        return null;
```

```
    }
```

```
    return msg;
```

```
}
```

```
//Use ToolKits as shown below. For details, refer to ToolKits.java
```

```
/*
```

```
 * Get one config
```

```
 * @param hLoginHandle Login handle
```

```
 * @param nChn Channel number -1 refers to all channels
```

```
 * @param strCmd Config name
```

```

* @param cmdObject Config the corresponding structured object
* @return Successfully returns true
*/

public static boolean GetDevConfig(LLong hLoginHandle,int nChn,String
strCmd,Structure cmdObject){
    //...Initialize parameters
    //Get device config
    if(netsdkapi.CLIENT_GetNewDevConfig( hLoginHandle, strCmd , nChn, strBuffer,
nBufferLen,error,3000)){
        cmdObject.write();
        //Parse the obtained config information to cmdObject
        if(configapi.CLIENT_ParseData(strCmd,strBuffer,cmdObject.getPointer(),cmdObject.
size(),null)){
            cmdObject.read();
            //...
        }
    }
}

//Set auto register config information
/**
* Config network protocol
* @param m_hLoginHanle Login handle
* @param enable
* @param address Server IP
* @param nPort Server Port
* @param deviceId Device ID
* @param info Obtained network protocol config
* @return
*/

public static boolean setDVRIPConfig(LLong m_hLoginHandle, boolean enable, String
address, int nPort, byte[] deviceId, CFG_DVRIP_INFO info){
    CFG_DVRIP_INFO msg=info;
    //Parameter config, for details, refer to AutoRegisterModule.java
    //Send config
    return ToolKits.SetDevConfig(m_LoginHandle,-1,NetSDKLib.CFG_CMD_DVRIP,msg);
}

```

```

/*
 * Set single config, and this code is in ToolKits.java
 * @param hLoginHandle Login handle
 * @param nChn Channel number ,-1, refers to all channels
 * @param strCmd Config name
 * @param cmdObject Config corresponding structure
 * @return successfully returns true
 */
public static boolean SetDevConfig(LLong hLoginHandle,int nChn,String
strCmd,Structure cmdObject){
    //Parameter config...
    cmdObject.write();
    //Pack up config data
    if(configapi.CLIENT_PacketData(strCmd,cmdObject.getPointer(),cmdObject.size(),s
zBuffer,nBufferLen)){
        cmdObject.read();
        //Send config
        if(netsdkapi.CLIENT_SetNewDevConfig(hLoginHandle,strCmd,nChn,szBuffer,n
BufferLen,error,restart,30000)){
            result=true;
        }else{
            result=false;
        }
    }
    return result;
}

```

3.2 Enable/Stop Auto Register Service

3.2.1 Introduction

SDK provides auto register service, which is used to detect and monitor devices that are registered to the management platform. The detection and monitoring include starting up and stopping.

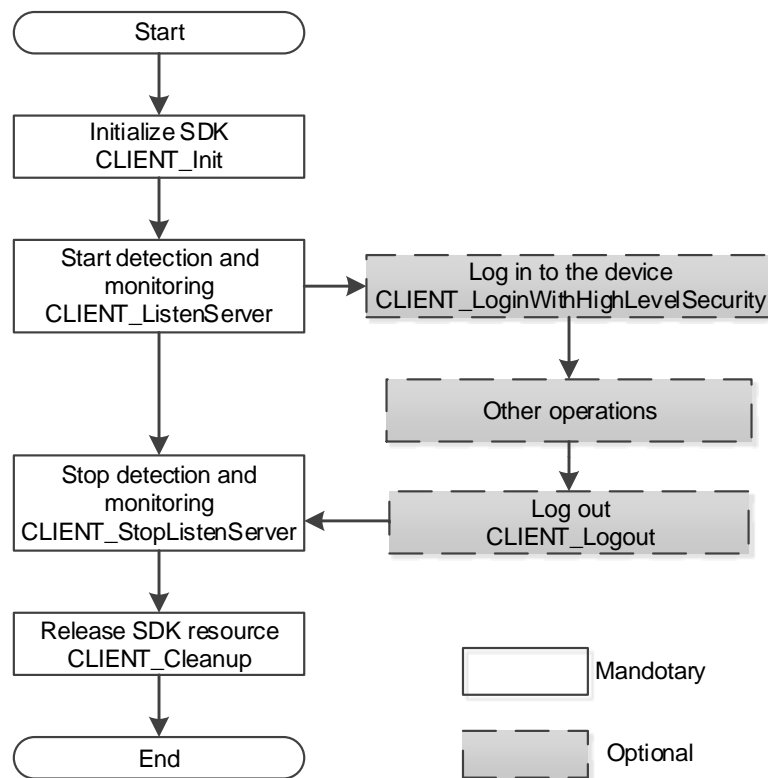
3.2.2 Interface Overview

Table 3-2 Starting up and stopping detection and monitoring interface

Functions	Description
CLIENT_Init	Initialize SDK
CLIENT_ListenServer	Start auto register detection and monitoring
CLIENT_StopListenServer	Stop auto register detection and monitoring
CLIENT_Cleanup	Clear SDK resources

3.2.3 Process Description

Figure 3-2 Starting up and stopping detection and monitoring



Process

- Step 1 Call **CLIENT_Init** to initialize interfaces to complete SDK initialization.
- Step 2 Call **CLIENT_ListenServer** interface to start device detection and monitoring. You need to import auto register detection and monitoring callback function **fServiceCallback** for the **CLIENT_ListenServer** interface. Through this callback function, you can get IP address, ID, port and more info about devices that are registered to the management platform. In the detection and monitoring status, callback function will return IP address, ID, port and more info of the devices that are

registered to the management platform. You can log in to devices and do other operations after getting info from the callback function.

Step 3 Call **CLIENT_StopListenServer** interface to end detection and monitoring.

Step 4 Call **CLIENT_Cleanup** interface to clear and release SDK resources.

Notes

- **CLIENT_Init** and **CLIENT_Cleanup** must be called in pairs, and single thread being called multiple times in pairs is supported. It is recommended that the single thread be called once globally. Do not initialize or clean resources repeatedly.
- Make sure the target IP of auto register config match with the server IP and port.
- On in the detection and monitoring status, the server can communicate with auto register devices. Therefore, all sending command operations to devices by the server must be done in the detection and monitoring status, which means the detection and monitoring cannot be stopped.
- If SDK interfaces needed to be called in the callback function, please create a separate thread; otherwise the program will crash.

3.2.4 Sample Code

```
// For general codes, such as initialization and cleanup, please refer to 《NetSDK_JAVA
Programming Manual》
// Config detection callback
/**
 * Detect server callback
 */
public class ServiceCB implements fServiceCallBack {
    @Override
    public int invoke(LLong lHandle, final String plp, final int wPort,
        int lCommand, Pointer pParam, int dwParamLen, Pointer dwUserData) {
        // Convert pParam to serial number
        byte[] buffer = new byte[dwParamLen];
        pParam.read(0, buffer, 0, dwParamLen);
        String deviceId = "";
        try {
            deviceId = new String(buffer, "GBK").trim();
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }
}
```

```

    }
    switch(ICommand) {
        // Verify devices disconnection callback
        case EM_LISTEN_TYPE.NET_DVR_DISCONNECT: {
            //For details, refer to AutoRegister.javabreak;

        }
        // Carry serial number during device registration
        case EM_LISTEN_TYPE.NET_DVR_SERIAL_RETURN: {
            //Interface operations...
            //Start a new thread to log in
            new SwingWorker<LLong, String>() {
                @Override
                protected LLong doInBackground() {
                    return login(deviceTreeNode);
                }
                @Override
                protected void done() {
                    try {
                        if(get() == null) {return;}
                        //Update interface operations...
                        }.execute();
                        break;
                    }
                }
                break;
            }
            default:
                break;
        }
        return 0;
    }
}

//Start listening and registering detection service callback
private ServiceCB serviceCallback=new ServiceCB();
/**
 * Enable service

```

```

* @param address Local IP address
* @param port Local port, randomly
* @param callback
*/
public static boolean startServer(String address, int port, fServiceCallBack callback) {
    mServerHandler = LoginModule.netsdk.CLIENT_ListenServer(address, port, 1000,
callback, null);
    if (0 == mServerHandler.longValue()) {
        System.err.println("Failed to start server." + ToolKits.getErrorCodePrint());
    } else {
        System.out.printf("Start server, [Server address %s][Server port %d]\n", address,
port);
    }
    return mServerHandler.longValue() != 0;
}

//End listening

/**
* End service
*/
public static boolean stopServer() {
    boolean bRet = false;
    //Use the monitoring handle obtained by enabled service
    if(mServerHandler.longValue() != 0) {
        bRet = LoginModule.netsdk.CLIENT_StopListenServer(mServerHandler);
        mServerHandler.setValue(0);
        System.out.println("Stop server!");
    }
    return bRet;
}

```


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.