Orbbec-OpenNI2.0 SDK Android Interface

Android Platform Only

Table of Contents

1 Introduction

This document was prepared to standardize the calls of the critical APIs, let users quickly access Orbbec's openni2.0-android SDK (Orbbec-SDK) in their own projects, as well as prevent other problems caused by non-standard call in the process of using Orbbec SDK APIs. This document is for the Android platform only.

2 Orbbec-SDK

2.1 Library Files

Two parts were provided by Orbbec-SDK to their users: five ".so dynamic" files and "1 .jar" file.

libOpenNI2.jni.so libOpenNI2.so liborbbec.so liborbbecusb2.so

openni2.3.jar

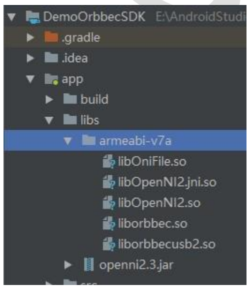
2.2 ini Files

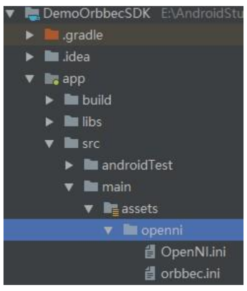
Two .ini setting files：

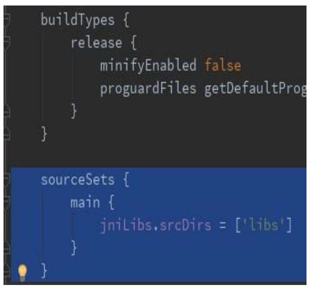
OpenNI.ini orbbec.ini

2.3 Access SDK

library files need to be placed under the libs directory in project Module, and ini files need to be placed under app->src->main->asset->openni directory in project Module. Then, in build.gradle of Module, reconfiguring jniLibs directory, and synchronizing the entire project. (Figure)





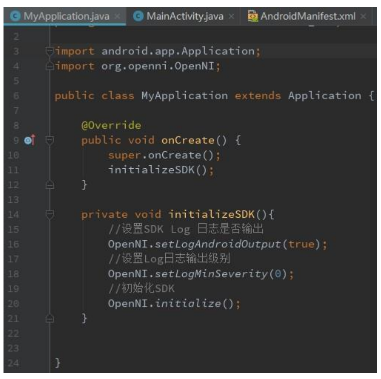


```
buildTypes {
    release {
        minifyEnabled false
        proguardFiles getDefaultProg
    }
}

sourceSets {
    main {
        jniLibs.srcDirs = ['libs']
    }
}
```

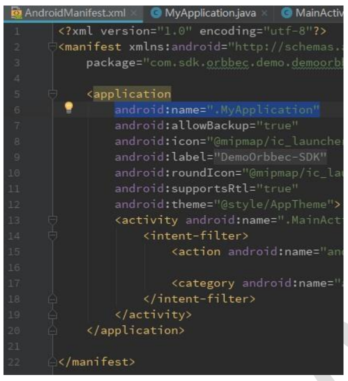
3 SDK Initialization

Initialization is necessary before calling Orbbec-SDK, and it usually processed when the program stars.

For example, the figure below shows utilizing in onCreate method of Android. (Figure 1,2)

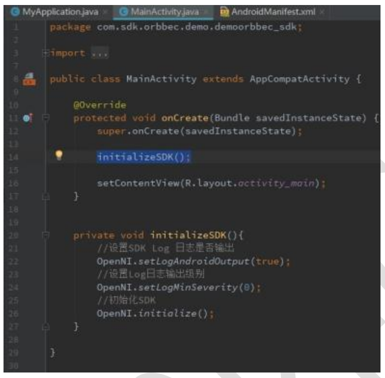


```java
import android.app.Application;
import org.openni.OpenNI;

public class MyApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();
        initializeSDK();
    }

    private void initializeSDK(){
        //设置SDK Log 日志是否输出
        OpenNI.setLogAndroidOutput(true);
        //设置Log日志输出级别
        OpenNI.setLogMinSeverity(0);
        //初始化SDK
        OpenNI.initialize();
    }


}
```



```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.
    package="com.sdk.orbbec.demo.demoorb

    <application
        android:name=".MyApplication"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launche
        android:label="DemoOrbbec-SDK"
        android:roundIcon="@mipmap/ic_la
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainAct
            <intent-filter>
                <action android:name="an

                <category android:name="
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Figure 1. Users can write self-defined application. Self-defined Application needs to be registered under AndroidManifest.xml file in Module.



```java
package com.sdk.orbbec.demo.demoorbbec_sdk;

import ...

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        initializeSDK();

        setContentView(R.layout.activity_main);
    }


    private void initializeSDK(){
        //设置SDK Log 日志是否输出
        OpenNI.setLogAndroidOutput(true);
        //设置Log日志输出级别
        OpenNI.setLogMinSeverity(0);
        //初始化SDK
        OpenNI.initialize();
    }

}
```

Figure 2. Finishing under onCreate in Activity.

Initializing call interface:

Interface	API	Parameter	Description
OpenNI	OpenNI .setLogAndroidOutput(true);	ture: output Android Log false：No output	Setting if SDK Log output or not.

	OpenNI .setLogMinSeverity(0);	0 - Verbose;
		1 - Info;
		2 - Warning;
		3 – Error;
			Setting Log output level.

	OpenNI .Initialize();		Initializing SDK(Must be called).

4 UsbDevice Request

In order to use USB device in Android, Developers need to Apply for Permission in advance. In Orbbec SDK, after enumerating OpenNIHelper to Orbbec USB device， it will send a broadcast to the system. Please registering the broadcast listener, and sending back to UsbDevice by registered callback. Callback will show OpenFailed if OpenNIHelper cannot be enumerated to the device or UsbDevice Request failed. The table below shows the interface prototype:

Interface Class	API	Parameter	Description
OpenNIHelper	OpenNIHelper(Context context)；	Recommended Context：getApplicationContext()	OpenNIHelper Initialization
	requestDeviceOpen(OpenNIHelper.DeviceOpenListener listener)	DeviceOpenListener	Requsting and opening UsbDevice connection.

Method:

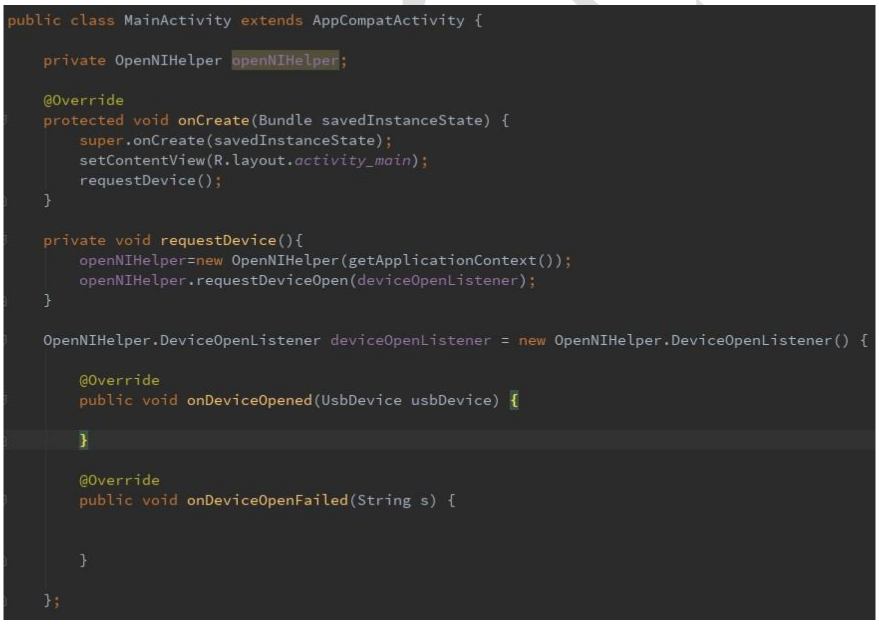
(1):OpenNIHelper openNIHelper=new OpenNIHelper(getApplicationContext());

-Developers can apply for permission when initializing the program according to their demand. This method only needs to be called once.

OpenNIHelper。

(2):openNIHelper.requestDeviceOpen(deviceOpenListener);

-Registering callback, getting device.


```
public class MainActivity extends AppCompatActivity {

    private OpenNIHelper openNIHelper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        requestDevice();
    }

    private void requestDevice(){
        openNIHelper=new OpenNIHelper(getApplicationContext());
        openNIHelper.requestDeviceOpen(deviceOpenListener);
    }

    OpenNIHelper.DeviceOpenListener deviceOpenListener = new OpenNIHelper.DeviceOpenListener() {

        @Override
        public void onDeviceOpened(UsbDevice usbDevice) {

        }

        @Override
        public void onDeviceOpenFailed(String s) {


        }

    };
```

(3) Pro series' (Color is standard UVC Camera) users needs to apply permission of color camera by themselves (Android 6.0 and up needs dynamically applying for permission)，and re-write onRequestPermissionsResult function in Activity.   Android 6.0 and up can only access Camera devices within user's authorization. Thus, UVC device of Pro device

needs to be opened in onRequestPermissionsResult’s callback.

(4) In DeviceOpenListener’s callback， processing according to the result.

Interface Class	Callback Function	Parameter	Description
DeviceOpenListener	onDeviceOpened(UsbDevice usbDevice)；	UsbDevice	Successfully return device
	onDeviceOpenFailed(String msg)；	msg	Error message; Failed to open the device.

5 Open Device

The device mentioned here is the Orbbec device. To use the correct device, users need to do the following steps: enumerating, matching, and opening.

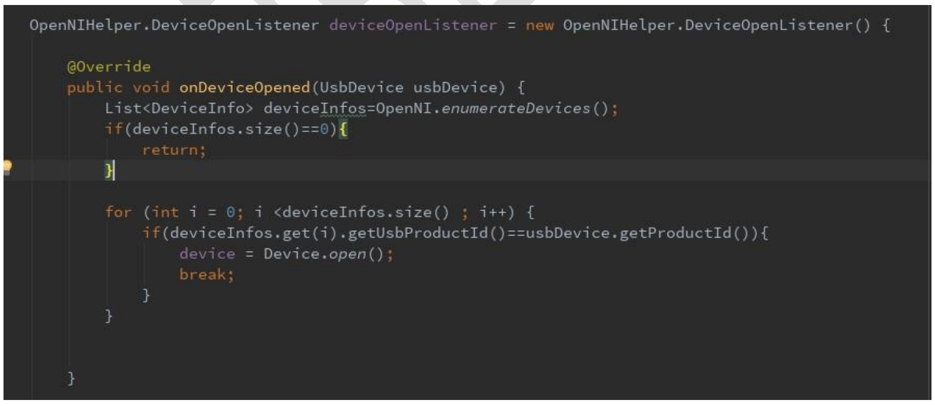
5.1 Enumeration

Enumerating Device by OpenNI.enumerateDevices().

Interface Class	Call back function	Parameter	Description
OpenNI	enumerateDevices ()；	N/A	Return List<DeviceInfo> (device list).

5.2 Open Device

After receiving the permission of USB device, Users need to enumerate and match the VID and PID of the device, and open device by OpenNI interface. (Figure)



```
OpenNIHelper.DeviceOpenListener deviceOpenListener = new OpenNIHelper.DeviceOpenListener() {

    @Override
    public void onDeviceOpened(UsbDevice usbDevice) {
        List<DeviceInfo> deviceInfos=OpenNI.enumerateDevices();
        if(deviceInfos.size()==0){
            return;
        }

        for (int i = 0; i <deviceInfos.size() ; i++) {
            if(deviceInfos.get(i).getUsbProductId()==usbDevice.getProductId()){
                device = Device.open();
                break;
            }
        }


    }
```

6 Create VideoStream

Creating VideoStream After Device opened. VideoStream has three types: iDepth, Color (OpenNI2.0 protocol), and IR.

6.1 Create VideoStream

Method: VideoStream stream = VideoStream.create(device, SensorType.DEPTH);

-The device is the one from step 5.

Interface Class	Callback Function	Parameter	Description

SensorType	IR(1);
	COLOR(2);
	DEPTH(3);
		N/A	Creating different type of VideoStream according to SensorType

6.2 Get VideoMode

After creating VideoStream, users can start to get and set the resolution, output format, and frame rate of the Stream.

Method: List<VideoMode> videoModes = stream.getSensorInfo().getSupportedVideoModes();

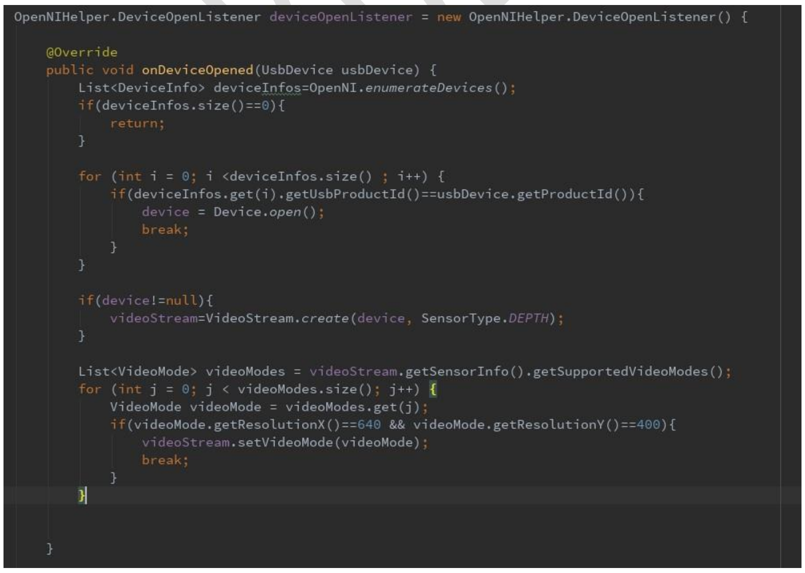
-The stream here is mentioning about the VideoStream from step 6.1.

-Users can set default resolution and output format in Orbbec.ini. VideoMode Set in code will cover the default one.

6.3 Set VideoMode

Method: stream.setVideoMode(videoMode);

- The stream here is mentioning about the VideoStream from step 6.1, and videoMode is from 6.2.



```
OpenNIHelper.DeviceOpenListener deviceOpenListener = new OpenNIHelper.DeviceOpenListener() {

    @Override
    public void onDeviceOpened(UsbDevice usbDevice) {
        List<DeviceInfo> deviceInfos=OpenNI.enumerateDevices();
        if(deviceInfos.size()==0){
            return;
        }

        for (int i = 0; i <deviceInfos.size() ; i++) {
            if(deviceInfos.get(i).getUsbProductId()==usbDevice.getProductId()){
                device = Device.open();
                break;
            }
        }

        if(device!=null){
            videoStream=VideoStream.create(device, SensorType.DEPTH);
        }

        List<VideoMode> videoModes = videoStream.getSensorInfo().getSupportedVideoModes();
        for (int j = 0; j < videoModes.size(); j++) {
            VideoMode videoMode = videoModes.get(j);
            if(videoMode.getResolutionX()==640 && videoMode.getResolutionY()==400){
                videoStream.setVideoMode(videoMode);
                break;
            }
        }


    }
```

7 Start VideoStream

Getting Video Stream Data methods: Active Acquirement and Callback.

7.1 Active Acquirement Mode

Staring a child thread.

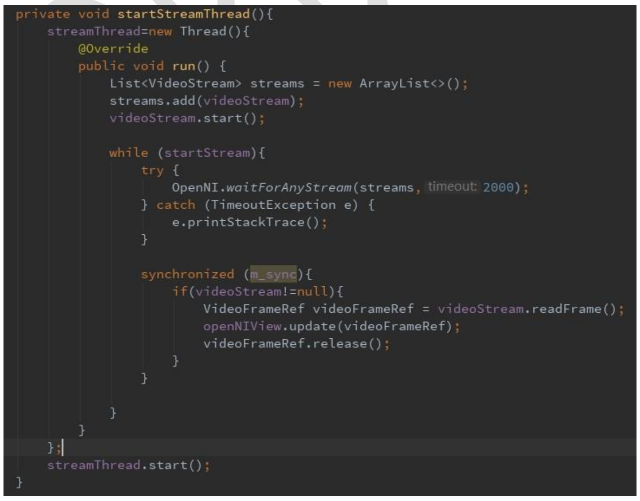
Method: videoStream.start();

OpenNI.waitForAnyStream(streams,2000); (2000 is timeout waiting time, and the unit is ms)

-users can add more than one VideoStream in stream. (Example)

```
VideoStream depthstream = VideoStream.create(device, SensorType.DEPTH);
VideoStream colorstream = VideoStream.create(device, SensorType.COLOR);
List<VideoStream> streams = new ArrayList<VideoStream>();
Streams.add(depthstream);
Streams.add(colorstream);
```

Interface Class	API	Parameter	Description
OpenNI	waitForAnyStream(List<VideoStream> streams, int timeout)；	streams: VideoStream list；timeout: timeout waiting time for one frame;	This is a blocking function until returning a new usable frame of data.
VideoStream	start();	N/A	Generate and output the stream data
VideoFrameRef	readFrame()；	N/A	Read one frame
	release()；	N/A	Release one frame
	readFrame() and release() interfaces must be utilized together in order to prevent memory from leaking.		



```
private void startStreamThread(){
    streamThread=new Thread(){
        @Override
        public void run() {
            List<VideoStream> streams = new ArrayList<>();
            streams.add(videoStream);
            videoStream.start();

            while (startStream){
                try {
                    OpenNI.waitForAnyStream(streams, timeout: 2000);
                } catch (TimeoutException e) {
                    e.printStackTrace();
                }

                synchronized (m_sync){
                    if(videoStream!=null){
                        VideoFrameRef videoFrameRef = videoStream.readFrame();
                        openNIView.update(videoFrameRef);
                        videoFrameRef.release();
                    }
                }

            }
        }
    };
    streamThread.start();
}
```

7.2 Callback Mode

Realized by setting Callback interface (Preferred).

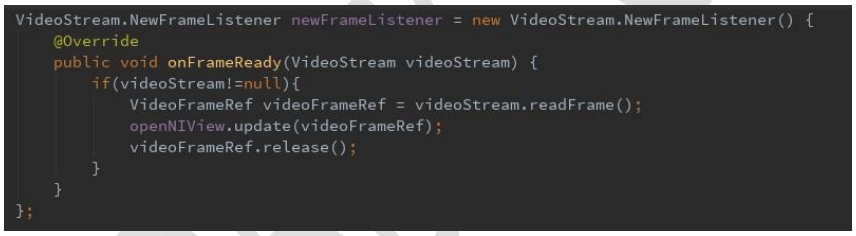
Method: (1). videoStream.start(); -Start video stream

(2). videoStream.addNewFrameListener(newFrameListener); -Set frame listener of of video stream

Interface Class	API	Parameter	Description
VideoStream	start();	N/A	Generate and output video stream
	addNewFrameListener(newFrameListener);	Callback interface	Set video stream call back
VideoFrameRef	readFrame();	N/A	Read one frame
	release();	N/A	Release one frame

Interface Class	API	Parameter	Description
NewFrameListener	onFrameReady(VideoStream var1);	var1: returned video stream	Return usable video frame reference. Please Avoid running other long-running operations in this callback function.

readFrame() and release() of VideoFrameRef must be used together, and must be released after use to prevent memory from leaking. (Figure)



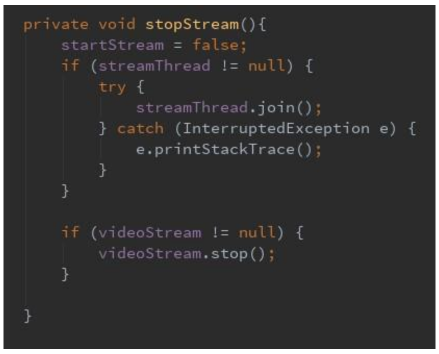
```
VideoStream.NewFrameListener newFrameListener = new VideoStream.NewFrameListener() {
    @Override
    public void onFrameReady(VideoStream videoStream) {
        if(videoStream!=null){
            VideoFrameRef videoFrameRef = videoStream.readFrame();
            openNIView.update(videoFrameRef);
            videoFrameRef.release();
        }
    }
};
```

8 Stop VideoStream

Method: VideoStream.stop();

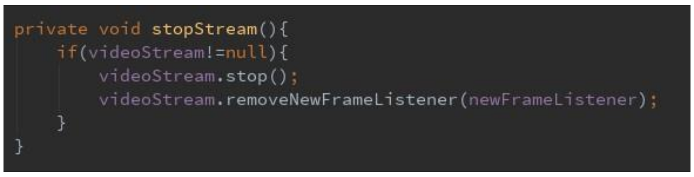
-If users want to reacquire the VideoStream after stop, they need to begin from step 7.

Interface	API	Parameter	Description
VideoStream	stop();	N/A	Stop generating VideoStream
	removeNewFrameListener(NewFrameListener streamListener)	NewFrameListener	Cancel data asynchronous callback interface listener


```
private void stopStream(){
    startStream = false;
    if (streamThread != null) {
        try {
            streamThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    if (videoStream != null) {
        videoStream.stop();
    }

}
```

Please removing callback if the data is acquired by Callback

Method: VideoStream.removeNewFrameListener (newFrameListener);

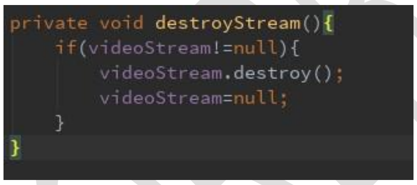

```
private void stopStream(){
    if(videoStream!=null){
        videoStream.stop();
        videoStream.removeNewFrameListener(newFrameListener);
    }
}
```

9 Destroy VideoStream

Method: VideoStream.destroy();

-If users want to reacquire the VideoStream after destroying, they need to begin from step 6.

Interface	API	Parameter	Description
VideoStream	destroy();	N/A	Destroy VideoStream (invoking together with Create VideoStream)

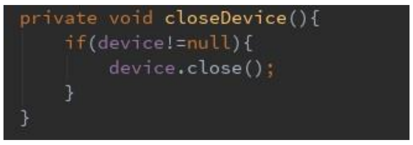

```
private void destroyStream(){
    if(videoStream!=null){
        videoStream.destroy();
        videoStream=null;
    }
}
```

10 Close Device

Method: device.close(); (Figure)

-If users want to reopen device and get stream data after calling this interface, they need to start from step 4-(2).

Interface Class	API	Parameter	Description
Device	close();	N/A	All hardware devices will be closed after calling this interface.



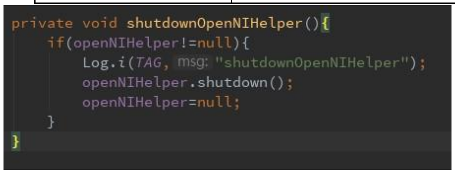
```
private void closeDevice(){
    if(device!=null){
        device.close();
    }
}
```

11 Stop UsbDevice

Method：openNIHelper.shutdown(); (Figure)

This API Corresponds to step 4. Please start from step 4-(1) if you need to Re-request device.

Interface Class	API	Parameter	Description
OpenNIHelper	shutdown()；	N/A	Disconnect with UsbDevice



```
private void shutdownOpenNIHelper(){
    if(openNIHelper!=null){
        Log.i(TAG, msg: "shutdownOpenNIHelper");
        openNIHelper.shutdown();
        openNIHelper=null;
    }
}
```

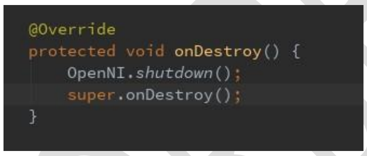
12 Release SDK

Method: OpenNI.shutdown();

Releasing SDK Corresponds to step 3. Before calling OpenNI.shutdown(), all VideoStream and Device resource should be released (Step 8-11). Please start from step 3 to reuse SDK. This interface might be invoked if exit APP.

Interface	API	Parameter	Description
OpenNI	shutdown()；	N/A	Release Orbbec-SDK resource

Recommended calling method:



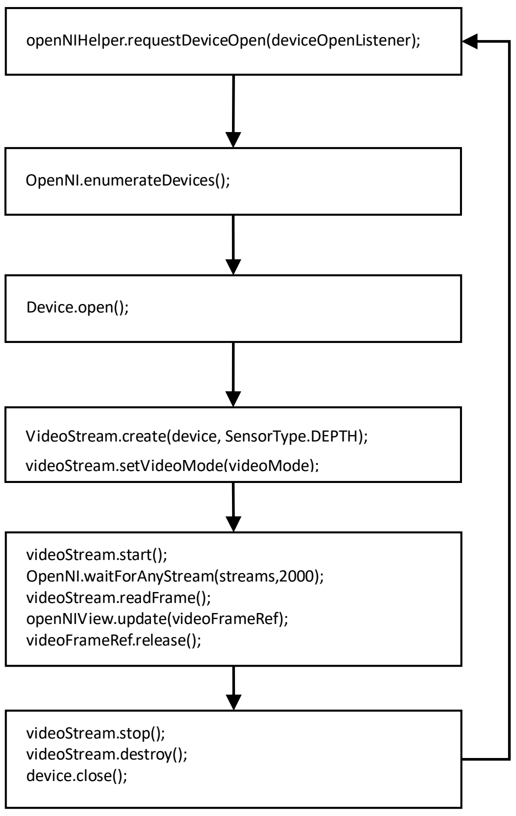
```
@Override
protected void onDestroy() {
    OpenNI.shutdown();
    super.onDestroy();
}
```

13 Interface Collaborative Calling Rules

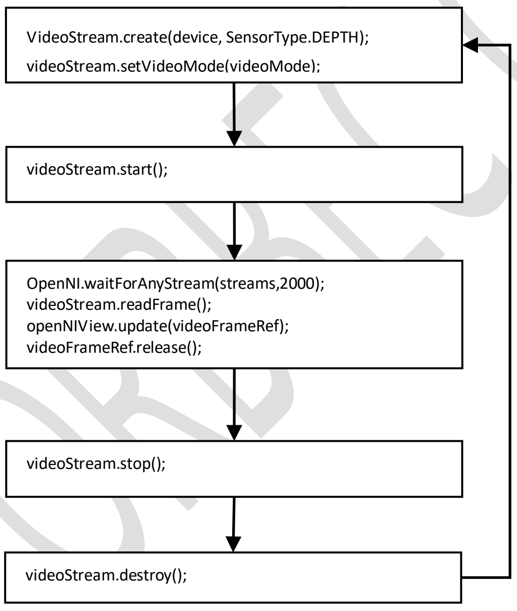
The rules mentioned here is about how to sequentially invoke API.

13.1 Active Acquirement Mode

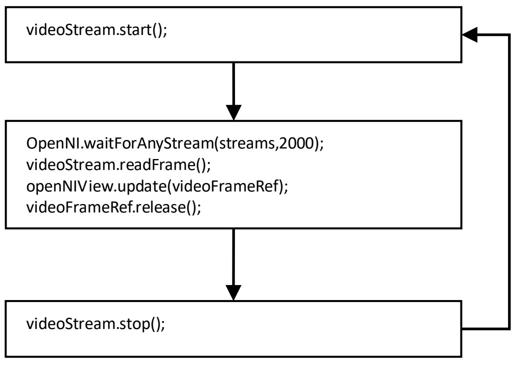
(1) Rule No.1 - Re-Require Device and open it after Device is closed:


openNIHelper.requestDeviceOpen(deviceOpenListener);
OpenNI.enumerateDevices();
Device.open();
VideoStream.create(device, SensorType.DEPTH);
videoStream.setVideoMode(videoMode);
videoStream.start();
OpenNI.waitForAnyStream(streams,2000);
videoStream.readFrame();
openNIView.update(videoFrameRef);
videoFrameRef.release();
videoStream.stop();
videoStream.destroy();
device.close();

(2) Rule No.2 - Users only need to execute Create and Destroy VideoStream if Device is not closed:

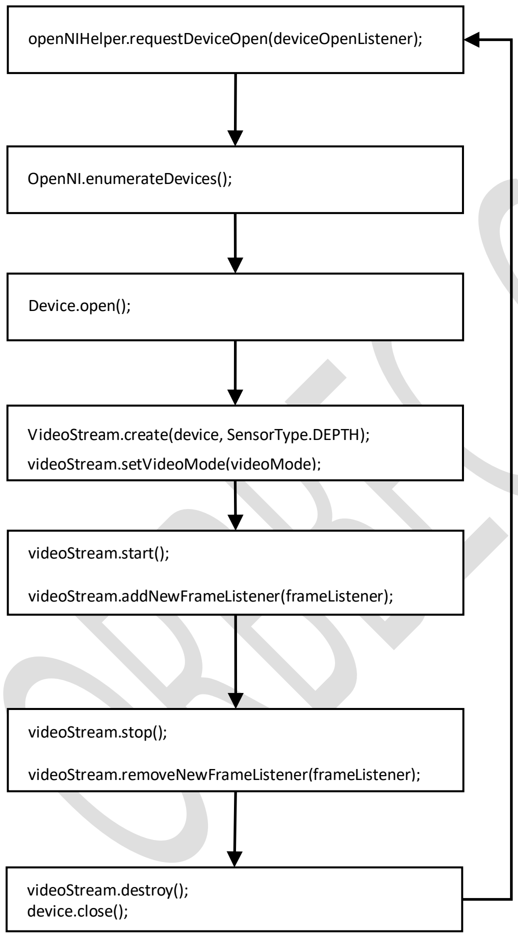

VideoStream.create(device, SensorType.DEPTH);
videoStream.setVideoMode(videoMode);
videoStream.start();
OpenNI.waitForAnyStream(streams,2000);
videoStream.readFrame();
openNIView.update(videoFrameRef);
videoFrameRef.release();
videoStream.stop();
videoStream.destroy();

(3) Rule No.3 - If VideoStream is not destroyed，users only need to execute start and stop functions of the VideoStream. (Preferred):

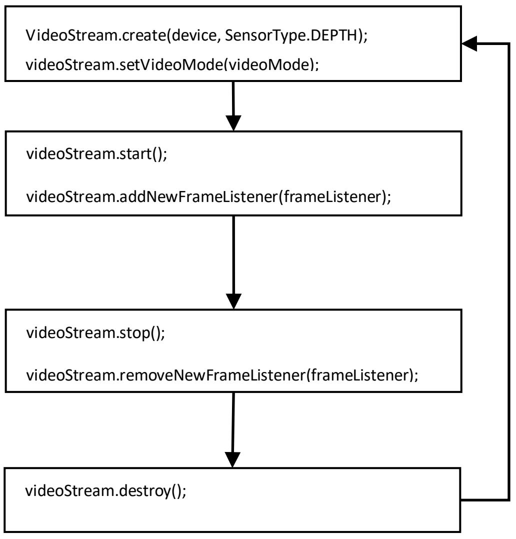


13.2 Callback Mode

(1) Rule No. 1 - Re-Require Device and open it after Device is closed:


openNIHelper.requestDeviceOpen(deviceOpenListener);
OpenNI.enumerateDevices();
Device.open();
VideoStream.create(device, SensorType.DEPTH);
videoStream.setVideoMode(videoMode);
videoStream.start();
videoStream.addNewFrameListener(frameListener);
videoStream.stop();
videoStream.removeNewFrameListener(frameListener);
videoStream.destroy();
device.close();

(2) Rule No.2 - Users only need to execute Create and Destroy VideoStream if Device is not closed:



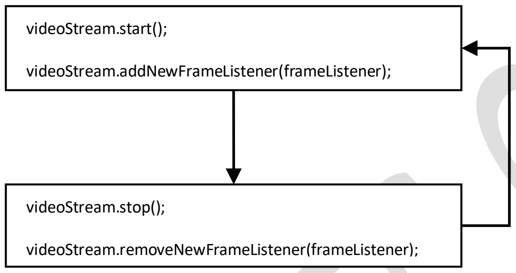
```
VideoStream.create(device, SensorType.DEPTH);
videoStream.setVideoMode(videoMode);
```

```
videoStream.start();

videoStream.addNewFrameListener(frameListener);
```

```
videoStream.stop();

videoStream.removeNewFrameListener(frameListener);
```

```
videoStream.destroy();
```

(3) Rule No.3 - If VideoStream is not destroyed, users only need to execute start and stop functions of the VideoStream. (Preferred):

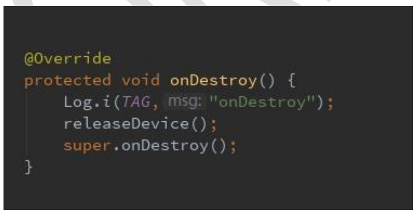


```
videoStream.start();

videoStream.addNewFrameListener(frameListener);
```

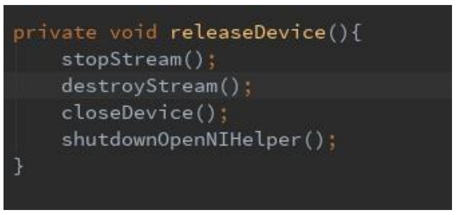
```
videoStream.stop();

videoStream.removeNewFrameListener(frameListener);
```

14 Return & Release SDK

Please release all SDK resources while pressing return button in Android. Developers need to invoke in onDestroy() function in Activity. Please calling the interface from step 12 when exit APP. (Figure)



```
@Override
protected void onDestroy() {
    Log.i(TAG, msg: "onDestroy");
    releaseDevice();
    super.onDestroy();
}
```



```
private void releaseDevice(){
    stopStream();
    destroyStream();
    closeDevice();
    shutdownOpenNIHelper();
}
```

15 Demo

The Demo calls the SDK's API interface in three ways by Active Acquirement Mode and three ways by Callback Mode. Please refer to the demo. and use the API correctly.