# SDK 开发者指南

**版本/版次：V1.7**

**受控状态：**

**编　　写：**

**审　　核：**

**批　　准：**

**生效日期：**

**秘密等级：公开**

# 修订记录

| 修订日期 | 版本号 | 修订说明 |
|---|---|---|
| 2018/7/18 | V0.1 | 初始测试版本 |
| 2018/7/19 | V0.2 | 增加修订记录等 |
| 2018/7/26 | V0.3 | 补充 android 接口 |
| 2018/7/26 | V0.4 | 整理文档格式 |
| 2018/8/14 | V0.5 | 增加同步和对齐接口说明 |
| 2018/9/07 | V0.6 | 增加 Java 层同步、对齐、设置分辨率说明 |
| 2018/9/18 | V0.7 | 增加新的同步模式说明 |
| 2018/11/16 | V0.8 | 增加 DSP 接口设置、镜像、翻转配置说明 |
| 2018/11/22 | V0.9 | 修改 DSP 接口设置 |
| 2018/11/22 | V1.0 | 增加图像对齐模式说明 |
| 2018/12/12 | V1.1 | 增加 VideoMode 和 VideoFrameRef 的说明 |
| 2018/12/13 | V1.2 | 增加第 2 种打开设备权限的方法 |
| 2019/1/2 | V1.3 | 增加 RGB 的曝光设置 |
| 2019/1/15 | V1.4 | 修改对齐模式说明、更新客户端说明 |
| 2019/1/22 | V1.5 | 对齐模式增加 IMAGE_REGISTRATION_COLOR_UNDISTORTION_ONLY |
| 2019/2/26 | V1.6 | 修改 DSP 接口设置中的示例代码错误 |
| 2019/11/26 | V1.7 | IMAGE_REGISTRATION_DEPTH_IR_TO_COLOR 要求先设置同步 |

# 1. 适用场景

本指南面向购买本公司产品进行二次应用开发的技术与维护人员。

# 2. SDK 安装

目前 SDK 支持 Windows、Linux、Android、Arm 平台，推荐使用配置如下

A. **操作系统**

**Windows:**

Windows 7, 8, 10 on x86 (32/64 bit)

**Ubuntu:**

Ubuntu 12.04 (32/64/arm)及以上

**Android:**

Android4.0 以上

B. **处理器**

Pentium 4, 1.4GHz 及以上

AMD Athlon 64/FX 1GHz 及以上

Arm Cortex A8 及以上

**C. 内存**

大于 512MB

**D. 接口**

USB2.0

**E. 开发环境**

VS2010，VS2015，Eclipse，AndroidStudio

**F. 显卡**

部分示例程序需高于 ATI RADEON x1300 or NVIDIA GeForce 7300

## 2.1. windows 平台

### 2.1.1. 安装包类型

32 位安装包

AXonLink-OpenNI-Windows-x86-2.3.msi

64 位安装包

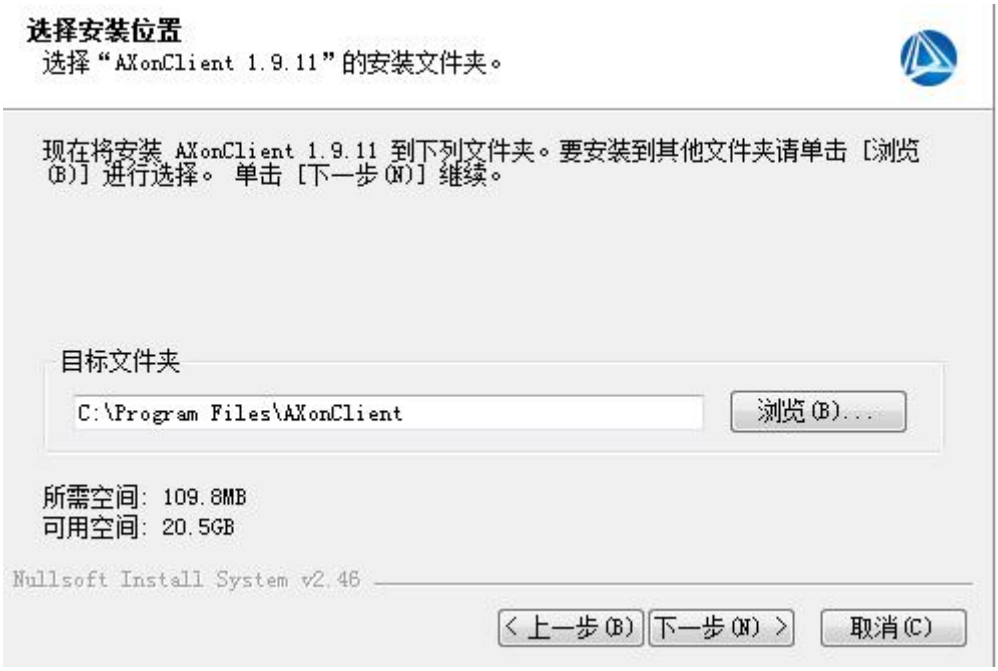
AXonLink-OpenNI-Windows-x64-2.3.msi

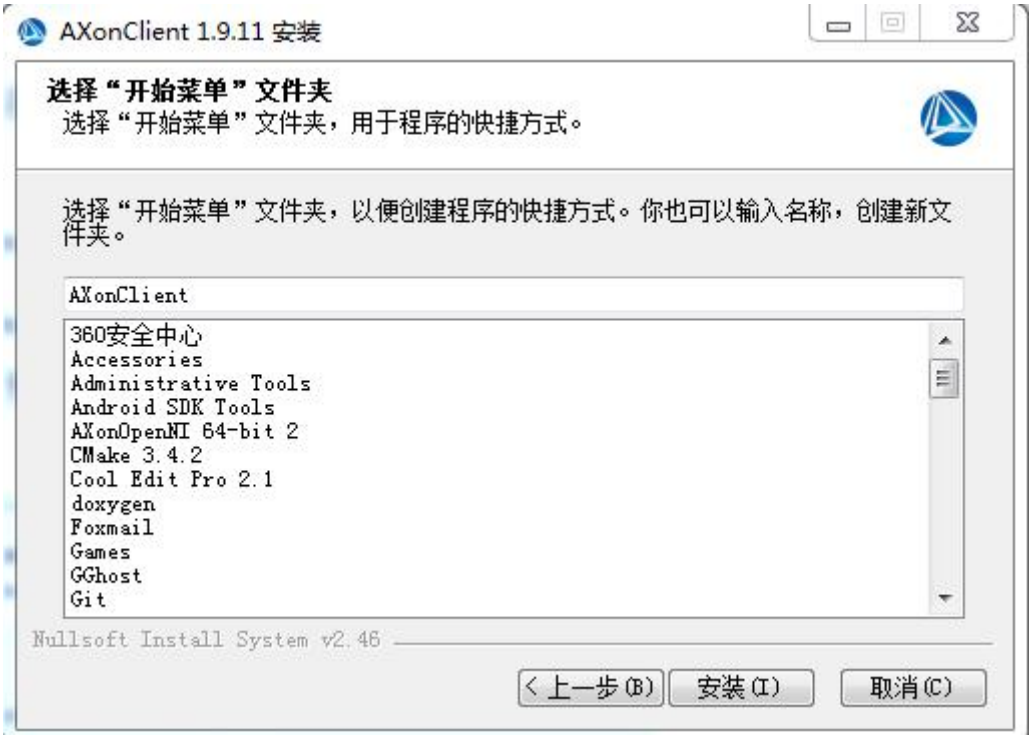
### 2.1.2. 安装步骤

- 打开安装包，选择安装路径

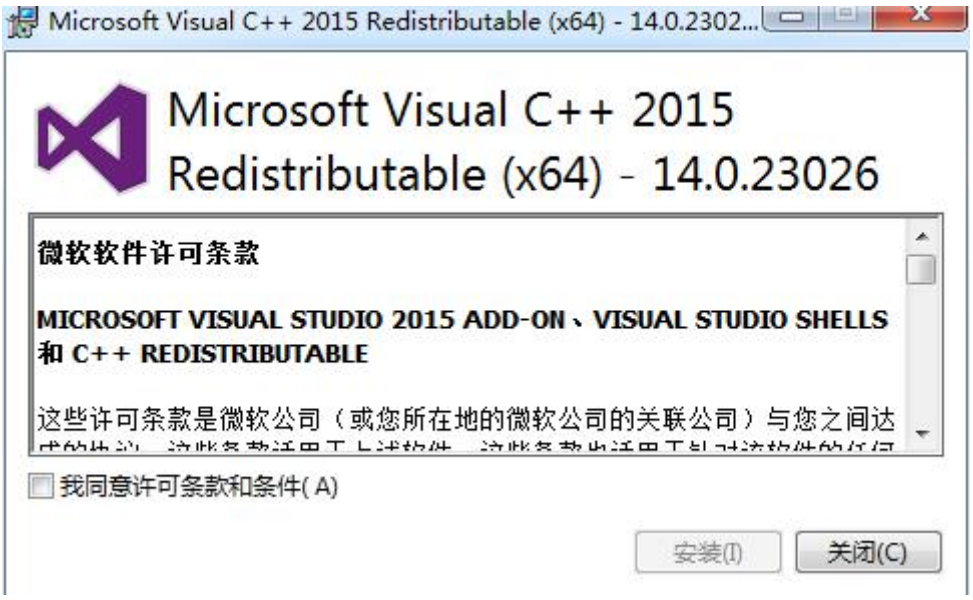

- 在安装过程中会弹出客户端的安装提示


- 点击下一步并配置安装路径

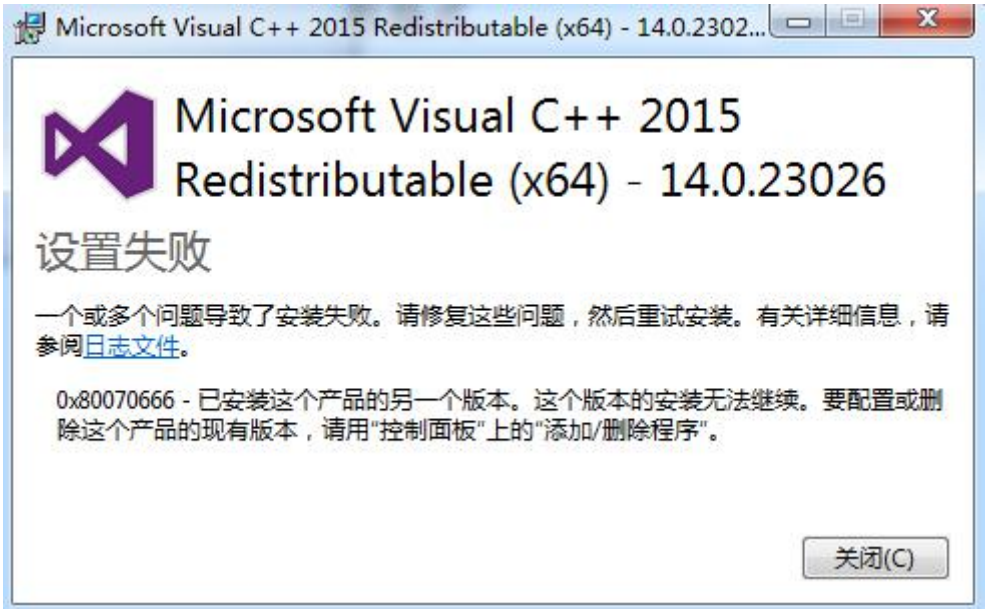




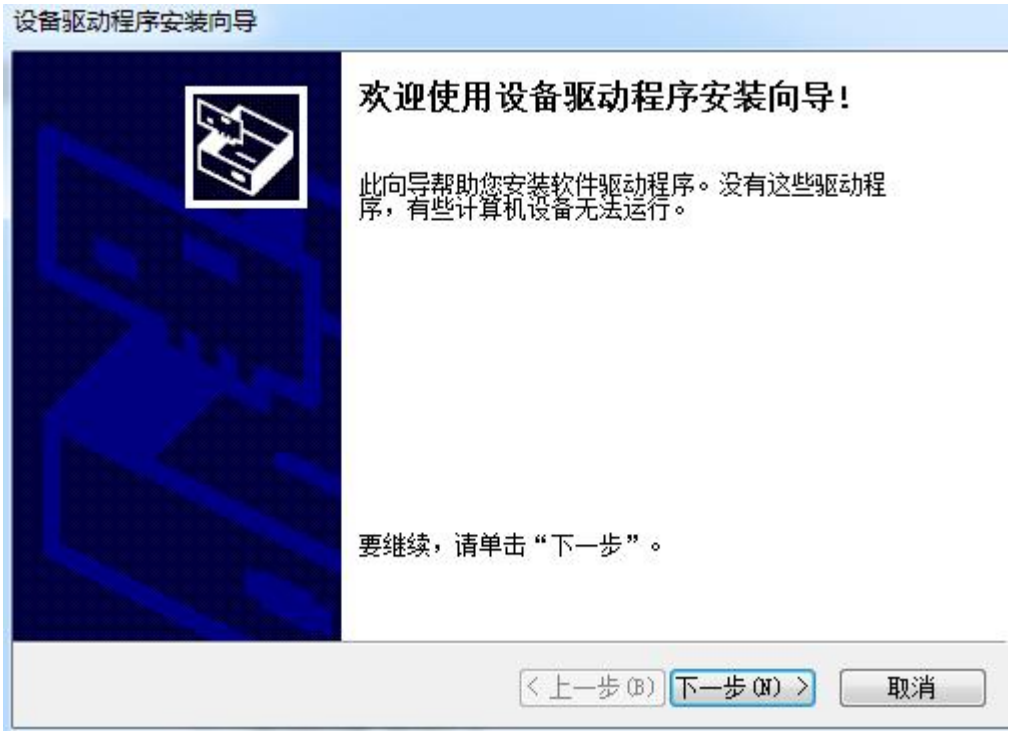

- 安装过程中会出现 VC++2015 安装，点击是

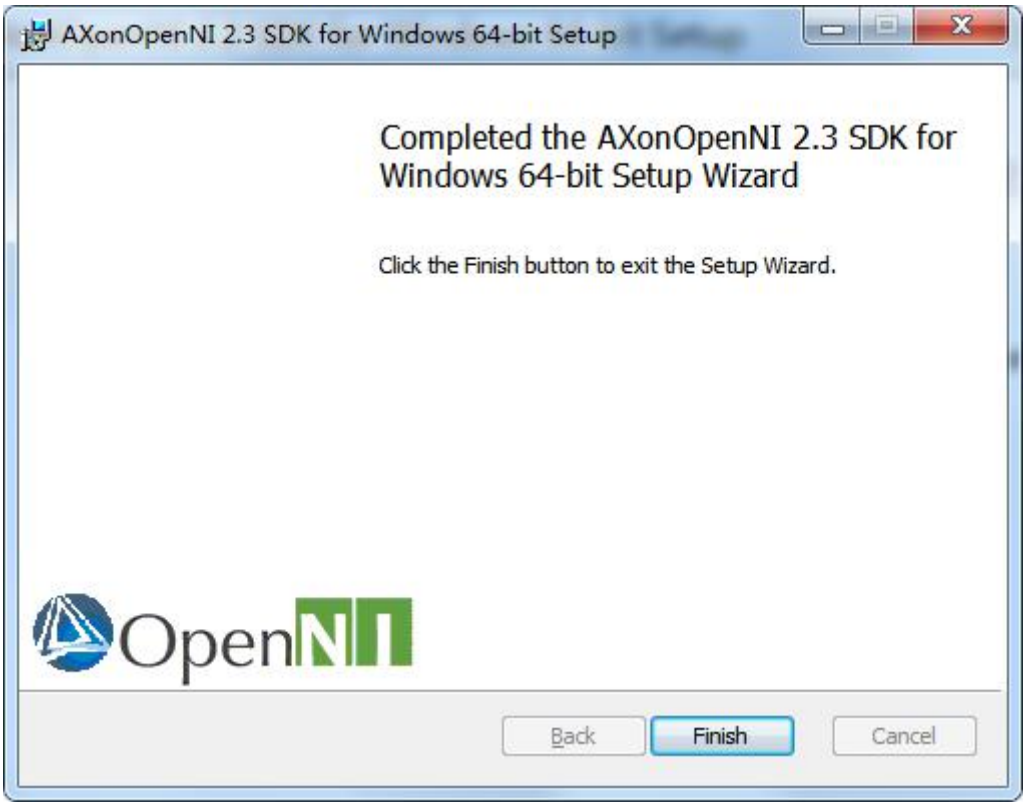



如果已安装会弹出失败，点击关闭继续安装；否则正常安装



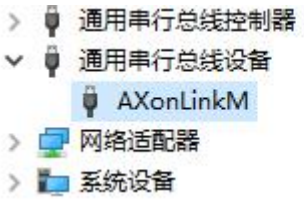

- 继续安装设备驱动




- 安装完成后连接设备，可在我的电脑->设备管理器中查看设备是否被正确识别，正确识别到设备显示如下图
  - Windows7


- Windows10




## 2.2. Linux 平台

以 Ubuntu16.04 为例

### 2.2.1. 安装包名称格式

安装包名称：AXonOpenNI-Linux-x64-2.3.tar.bz2

### 2.2.2. 安装过程

1. 安装所需驱动

```
sudo apt-get install libudev-dev libusb-1.0-0-dev
```

2. 安装图形化窗口

```
sudo apt-get install build-essential freeglut3 freeglut3-dev
```

3. 安装包解压

```
tar -jxvf AXonOpenNI-Linux-x64-2.3.tar.bz2
```

4. 执行安装脚本

```
sudo chmod +x install.sh
```

```
sudo ./install.sh
```

5. 设置环境变量

```
source OpenNIDevEnvironment
```

6. 编译 sample 文件(例如 AXon)

```
cd Samples/AXon
make
```

7. 运行可执行程序

```
cd Bin/x64-Release
sudo ./AXon
```

## 2.3. Android 平台

### 2.3.1. 安装包名称

安装包名称：OpenNI-Android-2.3.tar

### 2.3.2. 安装过程

解压安装包，将安装包中的示例 apk 安装到 android 设备即可运行。

**注意：**

程序正常运行需要 Android 设备获得 root 权限，并关闭 SELinux。

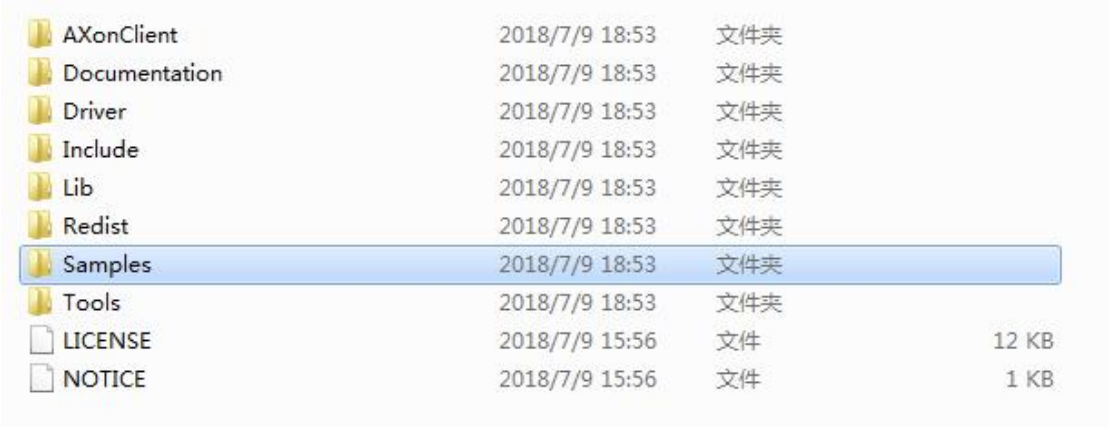
## 2.4. Arm 平台

根据需求定制。

# 3. SDK 使用说明

## 3.1. windows 平台

Windows 平台下的目录结构



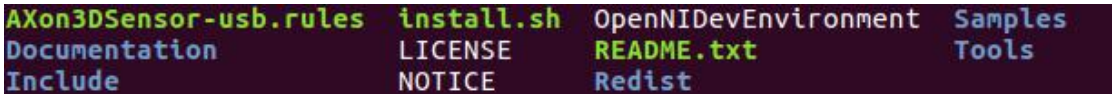

- AXonClient ： windows 下客户端的安装包；
- Documentation ： OpenNI 的标准接口指南，可通过 OpenNI 的接口对设备进行操作；
- Driver ：设备驱动安装程序；
- Include ： OpenNI 相关头文件，AXonLink.h 为自定义的一些数据结构和枚举；
- Lib ： OpenNI2.lib；
- Redist ：动态链接库，只有将 Redist 下的文件拷贝到应用程序所在的目录，应用程序才可正常运行。开发应用时需要包含 OpenNI2.lib 和相关头文件；以 VS2015 为例，在 c/c++附加包含目录中包含 Include


在***链接器->常规->附加***包含目录中包含 OpenNI2.lib 所在的路径；在***链接器->输入->附加依赖项***中加入 OpenNI2.lib；在程序中添加#include <AXonLink.h>和#include <OpenNI.h>。

- Samples ：目录下已包含多个示例工程，开发者可作参考；
- Tools ：目录下运行 NiViewer.exe 可预览相机获取的视频流；

## 3.2. Linux 平台

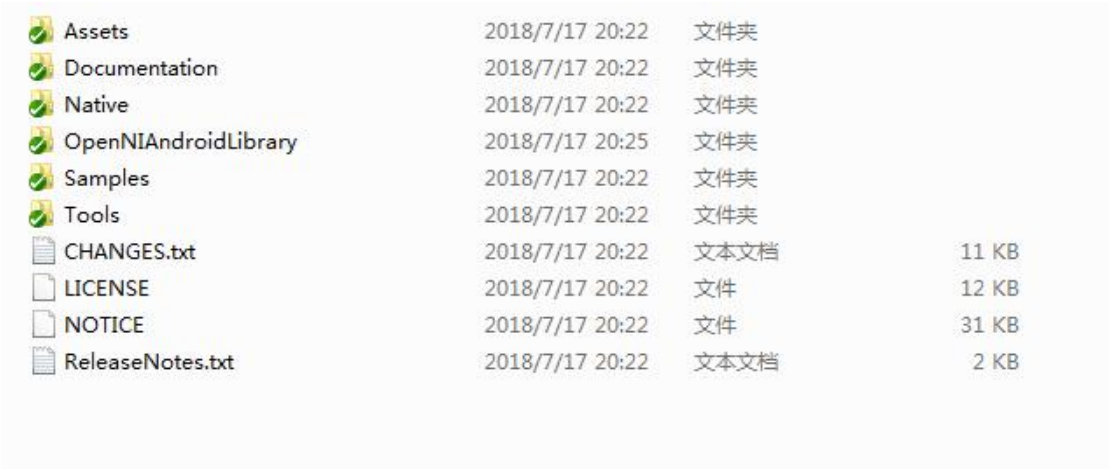
目录结构




- Documentation ： Openni 的帮助文档；
- Samples ：示例工程，make 后可运行使用 SDK 构建新项目的 Makefile 可参照 sample 示例中的 Makefile 编写。Linux 下同样可以运行 Tools 下的 Niviewer 程序，sudo ./NiViewer

## 3.3. Android 平台

目录结构




- Assets ：相关配置文件；
- Documentation ： c++和 java 下的使用文档，对于开发 android 应用，请参考 java 下的文档，其中 getting-started.html 指导如何调试一个开发包中的示例工程；
- Native ：头文件和库文件；
- OpenNIAndroidLibrary ：库文件生成 jar 包的工程；
- Sample ：三个应用应用示例的工程，prebuild 子目录下包含工程生成的.apk 文件，可在 android 设备上直接安装；
- Tools ：Niviewer 的工程源码和可安装 apk 文件，以上工程可用

eclipse 直接打开。

### 3.3.1. 工程介绍

Eclipse 包含 SimpleRead、SimpleWrite、NiViewer 工程，存在于 Sample->eclipseProject 中；

Android 仅包含 NiViewer 工程，存在于 Sample->androidStudioProject 中。

# 4. SDK 接口介绍

SDK 支持 OpenNI2 的大部分接口，部分自定义参数可通过 OpenNI2 的 property 接口和 invoke 接口获得，Android 暂不支持 invoke 命令。

## 4.1. 打开设备创建流，获取流数据信息

设备打开创建流，获取流的方式可根据 OpenNI 标准流程操作，OpenNI 帮助文档在安装目录的 Document 文件夹下。

用法如下：

```
#include <stdio.h>
#include <OpenNI.h>
#include "OniSampleUtilities.h"
#define SAMPLE_READ_WAIT_TIMEOUT 2000 //2000ms
using namespace openni;
int main(int argc, char* argv[])
{
 Status rc = OpenNI::initialize(); /*初始化 OpenNI*/
 if (rc != STATUS_OK)
 {
     printf("Initialize failed\n%s\n", OpenNI::getExtendedError());
```

```
        return 1;
    }

    Device device;

       if (argc < 2)
          rc = device.open(ANY_DEVICE);    /*打开任意可识别*/
       else
          rc = device.open(argv[1]);

       if (rc != STATUS_OK)
    {
        printf("Couldn't open device\n%s\n", OpenNI::getExtendedError());
        return 2;
    }
    OniVersion drver;
    int nsize;
    nsize = sizeof(drver);
    device.getProperty(ONI_DEVICE_PROPERTY_DRIVER_VERSION, &drver, &nsize);  /*获取驱
动版本号*/
    printf("AXon driver version V%d.%d.%d.%d\n", drver.major, drver.minor,
drver.maintenance, drver.build);
    VideoStream depth;

    if (device.getSensorInfo(SENSOR_DEPTH) != NULL)
    {
        rc = depth.create(device, SENSOR_DEPTH); /*有深度 Sensor 则创建深度流*/
        if (rc != STATUS_OK)
```

```
        {
            printf("Couldn't create depth stream\n%s\n", OpenNI::getExtendedError());
            return 3;
        }
    }

    rc = depth.start();  /*开启深度流*/
    if (rc != STATUS_OK)
    {
        printf("Couldn't start the depth stream\n%s\n", OpenNI::getExtendedError());
        return 4;
    }

    VideoFrameRef frame;

    while (!wasKeyboardHit())
    {
        int changedStreamDummy;
        VideoStream* pStream = &depth;
        rc    =    OpenNI::waitForAnyStream(&pStream,    1,    &changedStreamDummy,
SAMPLE_READ_WAIT_TIMEOUT);    /*等待流*/
        if (rc != STATUS_OK)
        {
            printf("Wait failed! (timeout is %d ms)\n%s\n", SAMPLE_READ_WAIT_TIMEOUT,
OpenNI::getExtendedError());
            continue;
        }
```

```
        rc = depth.readFrame(&frame); /*读深度流帧数据*/
        if (rc != STATUS_OK)
        {
            printf("Read failed!\n%s\n", OpenNI::getExtendedError());
            continue;
        }

        if (frame.getVideoMode().getPixelFormat() != PIXEL_FORMAT_DEPTH_1_MM &&
            frame.getVideoMode().getPixelFormat() != PIXEL_FORMAT_DEPTH_100_UM &&
            frame.getVideoMode().getPixelFormat() != PIXEL_FORMAT_DEPTH_1_3_MM)
        {
            printf("Unexpected frame format\n");
            continue;
        }

        DepthPixel* pDepth = (DepthPixel*)frame.getData(); /*当前帧的深度数据*/

        int middleIndex = (frame.getHeight()+1)*frame.getWidth()/2;
            /*打印当前帧的时间戳和中心深渡值*/
    printf("[%08llu]%8d\n",(long long)frame.getTimestamp(), pDepth[middleIndex]);
    }

    depth.stop();  /*停止深度流*/
    depth.destroy(); /*销毁深度流*/
    device.close(); /*关闭设备*/
    OpenNI::shutdown(); /*关闭 openni*/

    return 0;
```

```
}
```

## 4.2. VideoMode 说明

VideoMode 表示数据流的格式，包括图像宽度、高度、像素格式和帧率。可以通过 SensorInfo::getSupportedVideoModes()来获得设备某一种 sensor(Color/IR/Depth)支持的所有的 VideoMode：

***C++***

```
const SensorInfo* colorSensorInfoP =
 device.getSensorInfo(SENSOR_COLOR);
if(colorSensorInfoP)
{
    const Array<VideoMode>& allVMs =
    colorSensorInfoP->getSupportedVideoModes();
    ......
}
```

***Java***

```
if (device.getSensorInfo(SensorType.COLOR) != null) {
     List<VideoMode> supportedModes =
         mVideoStream.getSensorInfo().getSupportedVideoModes();
}
```

可以通过 VideoStream::setVideoMode()来修改 VideoMode。比如下列代码把 Color 图像设置为 jpeg 格式：

***C++***

```
VideoMode vm = ColorStream.getVideoMode();
vm.setPixelFormat(PIXEL_FORMAT_JPEG);
ColorStream.setVideoMode(vm);
```

***Java***

```
VideoMode vm = ColorStream.getVideoMode();
vm.setPixelFormat(PixelFormat.JPEG);
ColorStream.setVideoMode(vm);
```

## 4.3. VideoFrameRef 说明

用户程序一般通过 VideoStream::readFrame 来获得一个 VideoFrameRef。一个 VideoFrameRef 对象代表一个数据帧，可以是 Color，IR 或者 Depth。VideoFrameRef 的几个常用的方法：

| | |
|---|---|
| getWidth | 获取图像的宽度（像素点） |
| getHeight | 获取图像的高度（像素点） |
| getData | 获取图像数据 |
| getDataSize | 获取图像数据的大小（字节） |

getData 得到的数据，不同图像类型的格式是不同的：

- Depth：数据类型为 uint16_t，按照先行后列的顺序排列数据。一个 uint16_t 代表一个点的深度值，深度值的单位参见获取深度数据的单位。
- IR：数据类型为 uint8_t，按照先行后列的顺序排列数据。一个 uint8_t 代表一个点的 IR 亮度，值越大表示越亮。
- Color：
  - 如果 ColorStream 的 VideoMode 中的 pixelFormat 为 PIXEL_FORMAT_RGB888，那么数据类型为 RGB888Pixel。一个 RGB888Pixel_t 代表一个点的 RGB 值。例如，opencv 中可以通过以下方法将 ColorFrame 转化为 cv::Mat：

    ***C++***

    ```
    cv::Mat  im_RGB(Colorframe.getHeight(),  Colorframe.getWidth(),  CV_8UC3,
    (void*)Colorframe.getData());
    ```

    ***Java***

    ```
    Java 版本中 Colorframe.getData()为 bytebuffer 格式，其余和 C++相似，用户可根据
    条件自行适配至 opencv。
    ```

    注意，这时候 Mat 中的像素数据顺序是 RGB 的。

- 如果 ColorStream 的 VideoMode 中的 pixelFormat 为 PIXEL_FORMAT_JPEG，那么数据类型为 jpeg 图片数据。例如，opencv 中可以通过以下方法将 ColorFrame 转化为 cv::Mat：

***C++***

```
cv::Mat im_RGB =
cv::imdecode(std::vector<uchar>((uchar*)Colorframe.getData(),
(uchar*)Colorframe.getData()+Colorframe.getDataSize()),
CV_LOAD_IMAGE_COLOR);
```

***Java***

```
Java 版本中 Colorframe.getData()为 bytebuffer 格式，其余和 C++相似，用户可根据条件自行适配至 opencv。
```

注意，这时候 Mat 中的像素数据顺序是 BGR 的。

## 4.4. 获取深度数据的单位

如打开设备创建流，获取流数据信息中的代码所示，

frame.getVideoMode().getPixelFormat()得到的值表示深度数据的单位；具体说明可参见变量定义处的代码注释。

也可以通过 openni::OpenNI::getDepthValueUnit_mm 来得到深度值的单位。

部分支持的值和表示的单位如下：

| C++ | Java | VALUE |
|---|---|---|
| PIXEL_FORMAT_DEPTH_1_MM<br>ONI_PIXEL_FORMAT_DEPTH_1_MM | DEPTH_1_MM | 1 毫米 |
| PIXEL_FORMAT_DEPTH_100_UM<br>ONI_PIXEL_FORMAT_DEPTH_100_UM | DEPTH_100_UM | 100 微米(0.1 毫米) |
| PIXEL_FORMAT_DEPTH_1_3_MM<br>ONI_PIXEL_FORMAT_DEPTH_1_3_MM | DEPTH_1_3_MM | 1/3 毫米(0.33 毫米) |
| PIXEL_FORMAT_DEPTH_1_2_MM<br>ONI_PIXEL_FORMAT_DEPTH_1_2_MM | DEPTH_1_2_MM | 1/2 毫米(0.50 毫米) |

## 4.5. 获取设备的固件版本号

***C++***

```
    openni::Status status = STATUS_OK;
    AXonLinkFWVersion version;
    int dataSize = (int)sizeof(AXonLinkFWVersion);
    status=device->getProperty(ONI_DEVICE_PROPERTY_FIRMWARE_VERSION,&version,
&dataSize);
```

***Java***

```
    FWVersion fwversion = null;
    fwversion = device.getFWVersion();
```

## 4.6. 获取固件镜像文件的版本号

***C++***

```
    // 假定 windows 下固件保存的路径为
    char filePath[256] = “E:\resource”;
    filePath .Replace(_T("\\"), _T("\\\\"));
    int filepathSize  = sizeof(filePath );
    AxonLinkFirmWarePacketVersion packetVersion;
    memcpy(packetVersion.filename, filePath ,filepathSize);
    device->invoke(AXONLINK_DEVICE_INVOKE_GET_FWVERSION,&packetVersion,sizeof(packet
Version);
```

***Java***

```
    //Android 系统，将升级文件 update.img 放在 Download 文件夹，开发时注意添加文件读写权限申
请
    File sdCard = Environment.getExternalStorageDirectory();
```

```
    File directory_download = new File(sdCard, "Download");
    String filePath = directory_download.getAbsolutePath()+"/update.img";
    FirmWarePacketVersion                    fwpacketVersion                    =
device.getFirmWarePacketVersion(filePath);
```

## 4.7. 固件升级

开发中可根据设备的固件版本号和当前镜像的版本号进行比较来确定是否需要升级。

- **设置升级文件**

***C++***

```
    openni::Status rc = STATUS_OK;
    rc  =  device->invoke(AXONLINK_DEVICE_INVOKE_SET_UPLOADFILE,  (  char*)FileName,
filepathSize);
```

***Java***

```
    device.SetUpgradeFile(FileName);
```

- **设备文件成功后开始升级**

***C++***

```
    uint8_t enable =1;
    rc=device->invoke(AXONLINK_DEVICE_INVOKE_SET_UPGRADE_ENABLE,&enable,sizeof(uint8
_t));
```

***Java***

```
    device.UpgradeEnable(true);
```

- **获取当前升级状态**

设置一个定时器，推荐每 500ms 查询一次当前的升级状态

***C++***

```
    AXonLinkSendFileStatus m_upgradeStatus = AXON_LINK_SENDFILE_STATUS_STOP;
```

```
    device->invoke(AXONLINK_DEVICE_INVOKE_GET_UPGRADE_STATUS,&m_upgradeStatus,sizeof
(m_upgradeStatus));
```

***Java***

```
    UpgradeStatus status = device.getUpgradeStatus();
    Log.i(TAG,status.toString());
```

- **结束升级**

若状态为 AXON_LINK_SENDFILE_STATUS_SUCCESS 或 AXON_LINK_SENDFILE_STATUS_FAILED 后需要关闭升级，否则会造成内存泄漏。

***C++***

```
    uint8_t enable =0;
    rc=device->invoke(AXONLINK_DEVICE_INVOKE_SET_UPGRADE_ENABLE,&enable,sizeof(uint8
_t));
```

***Java***

```
    device.UpgradeEnable(false);
```

## 4.8. 重启设备

***C++***

```
    device->invoke(AXONLINK_DEVICE_INVOKE_SET_REBOOT, 0, 0);
```

***Java***

```
    device.reboot();
```

## 4.9. 获取相机内外参

***C++***

```
    AXonLinkCamParam camParam;
```

```
    int dataSize = sizeof(AXonLinkCamParam);
    rc=device->getProperty(AXONLINK_DEVICE_PROPERTY_GET_CAMERA_PARAMETERS,
&camParam,&dataSize);
```

***Java***

```
    CamParam camparam = null;
    camparam = mDevice.getCamParam();
```

获得的内参中包含各种分辨率下的内参，需要取对应分辨率下的内参来使用。如果某种分辨率找不到对应的内参，就表示这种分辨率没有可用的内参。

## 4.10. 获取设备串号

***C++***

```
    char serial[AXON_LINK_SERIALNUMBER_SIZE];
    memset(serial, 0, AXON_LINK_SERIALNUMBER_SIZE);
    int nsize = AXON_LINK_SERIALNUMBER_SIZE;
    rc = device->getProperty(ONI_DEVICE_PROPERTY_SERIAL_NUMBER, serial, &nsize);
```

***Java***

```
    String serialnumber = device.getDeviceSerialNumber();
```

## 4.11. 设置分辨率

***C++***

```
    //设置 Color 的分辨率和 depth 相同
    VideoStream depthStream;
    VideoStream colorStream;
    ...
    const VideoMode depthVideoMode = depthStream.getVideoMode();
```

```
    VideoMode colorVideoMode = colorStream.getVideoMode();

    colorVideoMode.setResolution(depthVideoMode.getResolutionX(),
depthVideoMode.getResolutionY());
    colorStream.setVideoMode(colorVideoMode);
```

***Java***

```
    //设置 Color 的分辨率和 depth 相同
    VideoStream depthStream;
    VideoStream colorStream;
    ...
    const VideoMode depthVideoMode = depthStream.getVideoMode();
    VideoMode colorVideoMode = colorStream.getVideoMode();
    colorVideoMode.setResolution(depthVideoMode.getResolutionX(),
depthVideoMode.getResolutionY());
    colorStream.setVideoMode(colorVideoMode);
```

## 4.12. 设置 RGB 和 Depth 同步

USB 传输和数据处理等原因可能会导致收到的 RGB 和 Depth 数据在时间上是不同步的。可以调用同步接口来实现同步。

在调用同步机制时，若多个流都打开，每个流都要调用 readFrame 把自己的数据读走，否则其他流也无法读取后续的数据。

***C++***

```
    rc = device.setDepthColorSyncEnabled(true); //true 表示同步，false 表示不同步。
```

***Java***

```
    device.setDepthColorSyncEnabled(true)
```

同步算法有两个参数可以设置，在和 AXonLink.dll 或者 AXonLink.so 同目录的

AXonLink.ini 文件中：

SyncThresholdTime：如果不同流的两帧图像的时间差大于这个数值，那么这两个帧被认为是不同步的，会先缓存起来等待新的帧。否则认为是同步的，会发送给上层。

SyncMaxFrameNum：缓存的帧数，等效于缓存的最大时间长度。比如如果帧率为 30FPS，SyncMaxFrameNum=5，那么同步窗口为 1000/30*5=165ms。

## 4.13. 设置 RGB 和 Depth 对齐

设置对齐会增加主机端的 CPU 资源消耗。目前支持的同步模式见下表。注意，部分模式要求先设置 RGB 和 Depth 同步、或者 RGB 和 Depth 分辨率必须相同。

| 模式 | 说明 | 是否要求同步 | RGB 和 Depth 分辨率是否必须相同 |
|---|---|---|---|
| IMAGE_REGISTRATION_OFF | 不对齐，Color 无去畸变校正 | 否 | 否 |
| IMAGE_REGISTRATION_DEPTH_TO_COLOR | Depth 对齐到 Color，IR 不对齐，Color 去畸变校正 | 否 | 否 |
| IMAGE_REGISTRATION_DEPTH_IR_TO_COLOR | Depth 和 IR 对齐到 Color，Color 去畸变校正 | 是 | 否 |
| IMAGE_REGISTRATION_COLOR_TO_DEPTH | Color 对齐到 Depth 和 IR，Color 去畸变校正 | 是 | 是 |
| IMAGE_REGISTRATION_COLOR_UNDISTORTION_ONLY | 仅对 Color 去畸变校正 | 否 | 否 |

代码示例

***C++***

```
device.setImageRegistrationMode(IMAGE_REGISTRATION_DEPTH_TO_COLOR);//对齐
device.setImageRegistrationMode(IMAGE_REGISTRATION_OFF);//不对齐
```

***Java***

```
device.setImageRegistrationMode(ImageRegistrationMode.DEPTH_TO_COLOR)
```

```
device.setImageRegistrationMode(ImageRegistrationMode.OFF)
```

如果 `setImageRegistrationMode` 返回错误，请检查：

1. 对应的流是否已经创建。
2. VideoMode 中的 pixelFormat 需要设置为 PIXEL_FORMAT_RGB888。
3. 同步和分辨率是否满足上表的要求。
4. 不能设置对应流的“左右镜像(Mirror)”和“上下翻转(Flip)”。
5. 不能设置裁剪(Cropping)。
6. 相机是否有对应分辨率的内参。具体可以咨询支持工程师。

当 RGB 和 Depth 的分辨率不同（比如 RGB 为 1280x960，Depth 为 640x480）的时候，对齐后（比如模式 IMAGE_REGISTRATION_DEPTH_TO_COLOR），Depth 的分辨率并未改变。如果要得到 RGB 上某一点在 Depth 上对应的坐标，坐标需要相应的改变。比如在这个例子中，RGB 的坐标（x_rgb, y_rgb）和 Depth 的坐标(x_depth, y_depth)的关系为：

x_rgb = 1280/640 * x_depth = 2 * x_depth

y_rgb = 960/480 * y_depth = 2 * y_depth

其他分辨率下也要按照一定关系转换。

另外，如果没有设置对齐(未设置或者设置为 IMAGE_REGISTRATION_OFF)，那么输出的 RGB 图片是没有经过去畸变校正的。如果需要去畸变矫正，那么可以：

(1) 设置为其他对齐模式。如果仅仅需要去畸变矫正，可以设置为 IMAGE_REGISTRATION_COLOR_UNDISTORTION_ONLY。

(2) 利用相机的内外参自行处理。请参见获取相机内外参。

## 4.14. DSP 接口设置

DSP 接口设置一般不需要修改。

示例配置“去飞行像素”、“去飞行像素 2”、“3D 降噪”，可以根据需要设置。其他参数建议不要修改。

关于“去飞行像素”和“去飞行像素 2”的说明：

1. “去飞行像素 2”的效果更好，但是会消耗主机 CPU 资源。默认是关闭的。
2. “去飞行像素”效果差一些，但是不消耗 CPU 资源。默认是打开的。

***C++***

```
//获取当前状态
AXonDSPInterface m_dsp_process_interface;
```

```
device.invoke(AXONLINK_DEVICE_INVOKE_GET_DSP_DPINTERFACE,&m_dsp_process_interface,
sizeof(m_dsp_process_interface)) ;
// 0 表示关闭，1 表示打开
m_dsp_process_interface. NR3 = 1;                    // 打开 3D 降噪
m_dsp_process_interface. FLYING = 1;                  // 打开去飞行像素
m_dsp_process_interface. FLYING_2 = 1;                // 打开去飞行像素 2
device.invoke(AXONLINK_DEVICE_INVOKE_SET_DSP_DPINTERFACE,&m_dsp_process_interface,
sizeof(m_dsp_process_interface)) ;
```

***Java***

```
//配置 3D 降噪
int status = device.getDSPInterfaceStatus();                    //获取当前接口状态
// setDSPInterfaceStatus 第 2 个参数, 1 打开 0 关闭
device.setDSPInterfaceStatus(AXonDSPInterface.NR3, 1);          // 打开 3D 降噪
device.setDSPInterfaceStatus(AXonDSPInterface.FLYING, 1);       // 打开去飞行像素
device.setDSPInterfaceStatus(AXonDSPInterface. FLYING_2, 1);    // 打开去飞行像素 2
```

## 4.15. 设置左右镜像

***C++***

```
// true 打开 false 关闭
mStream.setMirroringEnabled(true);
// 获取状态
bool status = mStream. getMirroringEnabled();
```

***Java***

```
// true 打开 false 关闭
mStream.setMirroringEnabled(true);
// 获取状态
boolean status = mStream. getMirroringEnabled();
```

## 4.16. 设置上下翻转

***C++***

```
// 1 打开 0 关闭
OniBool wValue = 1;
mStream.setProperty(AXONLINK_STREAM_PROPERTY_FLIP, &wValue, sizeof(OniBool));
// 获取状态
OniBool wValue;
int size = sizeof(OniBool);
mStream.getProperty(AXONLINK_STREAM_PROPERTY_FLIP, &wValue, &size);
```

***Java***

```
// true 打开 false 关闭
mStream. setFlipEnabled(true);
// 获取状态
boolean status = mStream. getFlipEnabled ();
```

## 4.17. 设置 RGB 曝光

***C++***

```
//设置
int exposure = 40;  //具体取值范围与设备型号有关
ColorStream.getCameraSettings()->setExposure(exposure);
// 获取
int exposure = ColorStream.getCameraSettings()->getExposure();
```

***Java***

```
//设置
int exposure = 40;  //具体取值范围与设备型号有关
ColorStream.getCameraSettings().setExposure(exposure);
```

```
// 获取
int exposure = ColorStream.getCameraSettings().getExposure();
```

# 5. Windows 客户端使用说明

Windows 下的客户端界面如下

## 5.1. 显示页


- **设备连接**

当有可识别设备插入，设备列表将自动更新，多个设备时可通过下拉框选择想要操作的

设备

点击***连接选中相机***按钮，进行连接；

- **图像显示**

勾选 ***Depth***，***IR***，***Color*** 流，***点云***选项，点击 ***Preview*** 后可预览选中流的图像；

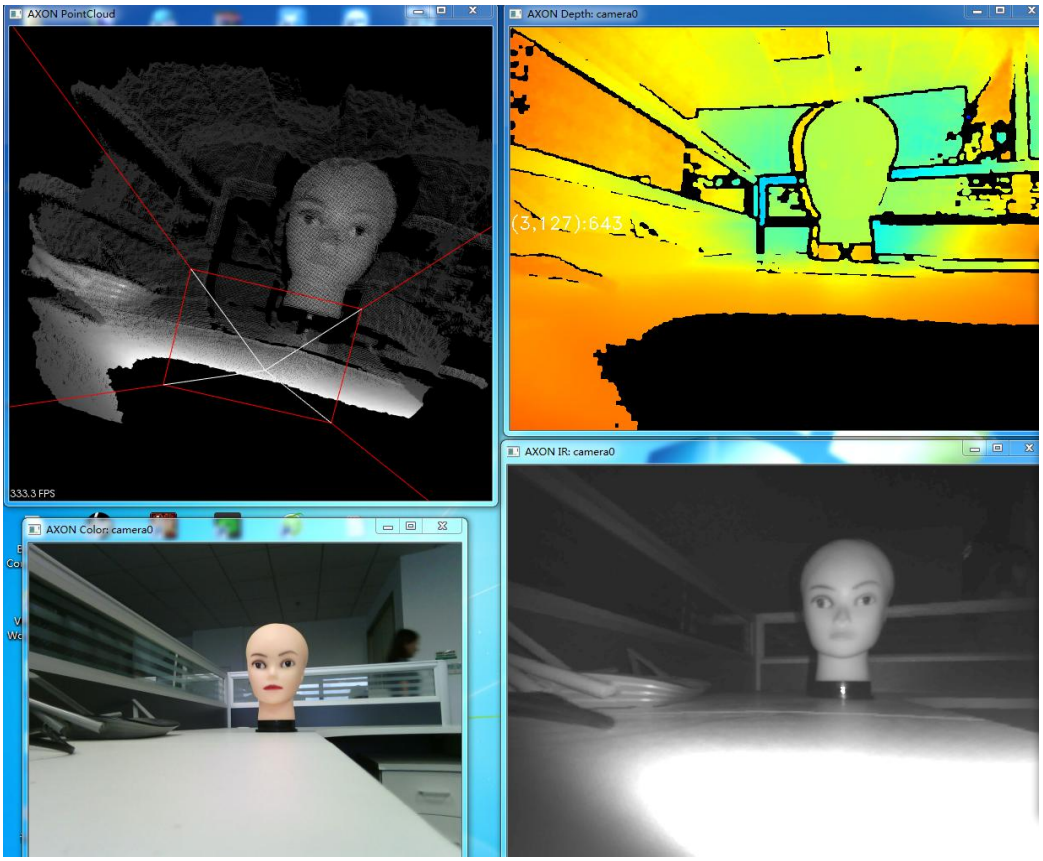
- **图像保存**

点击 ***Capture*** 按钮，可保存所选的流的图片，默认采集 30 张，可手动修改需要保存的张数，图片保存在程序所在文件夹下的 image 文件夹下，点云数据保存在 depthdata/pointCloud 文件夹下；

- **参数设置**

***距离上下限***的设置可调整 depth 流的伪彩显示有效范围，不在限值范围内的深度值不显示，***Color 显示比例***可调整 color 流显示窗口的大小，重新点击 ***Preview*** 按钮时生效。

所有显示效果图如下：




- **RGB 曝光等级**

可以修改 RGB 曝光等级。一般情况下不建议修改。并非所有设备都支持。可用范围请

咨询支持工程师。

- **RGB 模式**

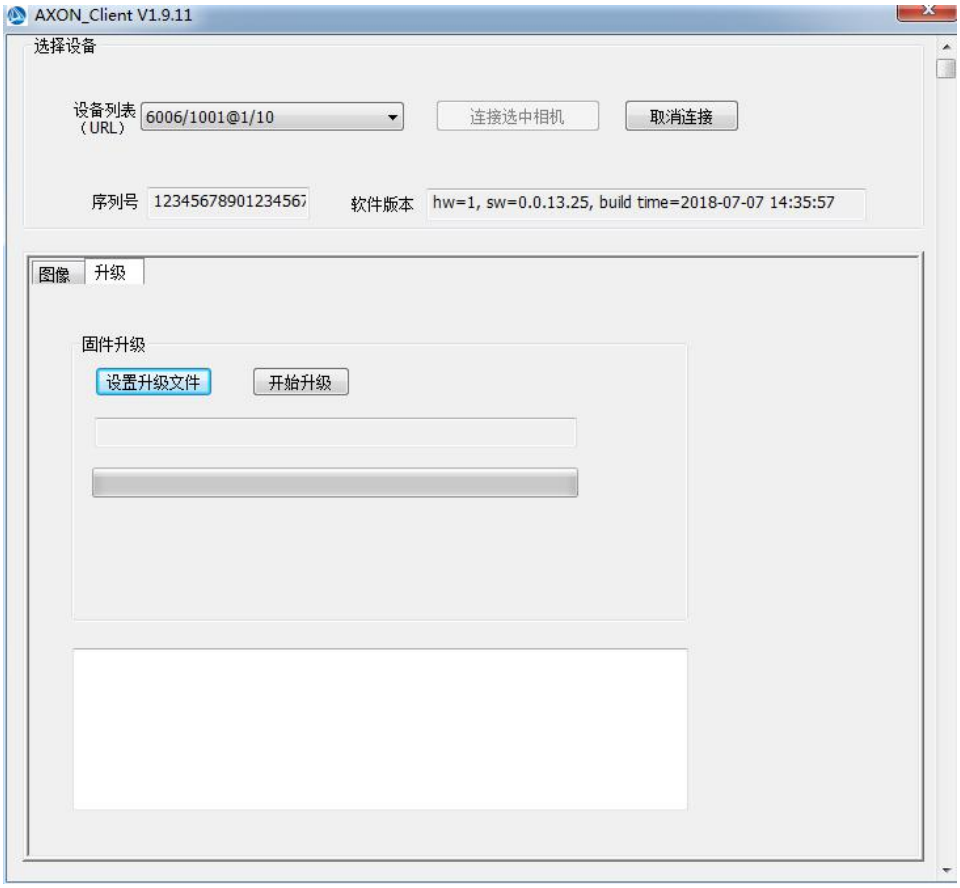
  可以查看和设置 RGB 的 VideoMode。

- **对齐模式**

  四种模式

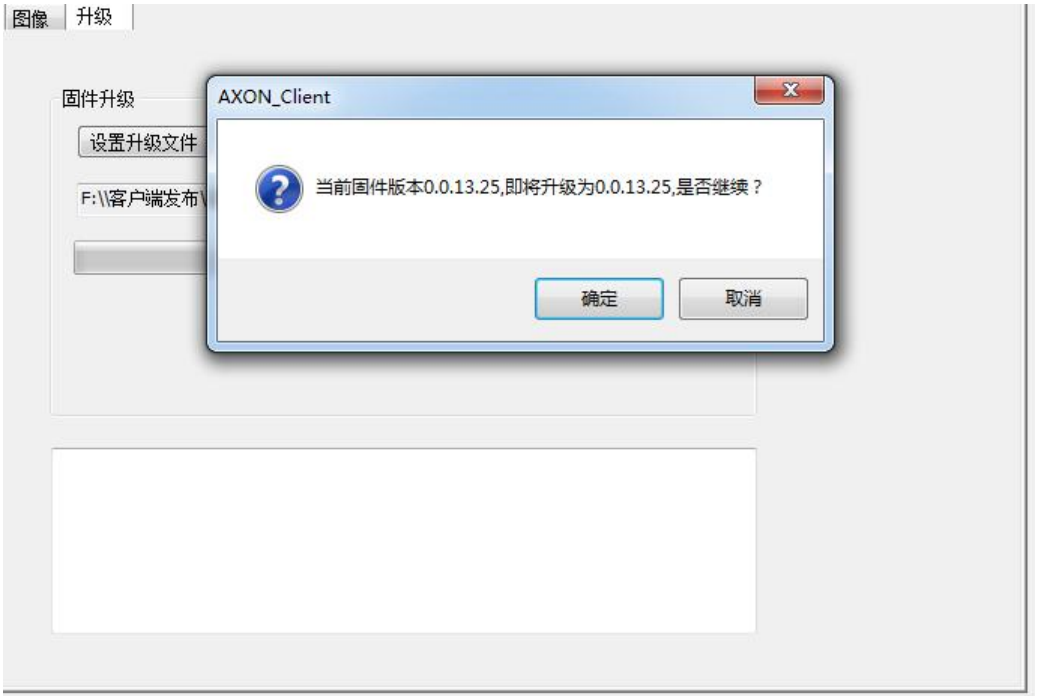
| | |
|---|---|
| Off | 关闭对齐模式（Color 无去畸变校正） |
| Depth to Color | 深度图对齐彩色图（包含 Color 去畸变校正） |
| Depth&IR to Color | 深度图与红外图对齐彩色图（包含 Color 去畸变校正） |
| Color to Depth | 彩色图对齐深度图（包含 Color 去畸变校正） |
| Color Undistort Only | 仅做 Color 去畸变校正 |

## 5.2. 升级页面

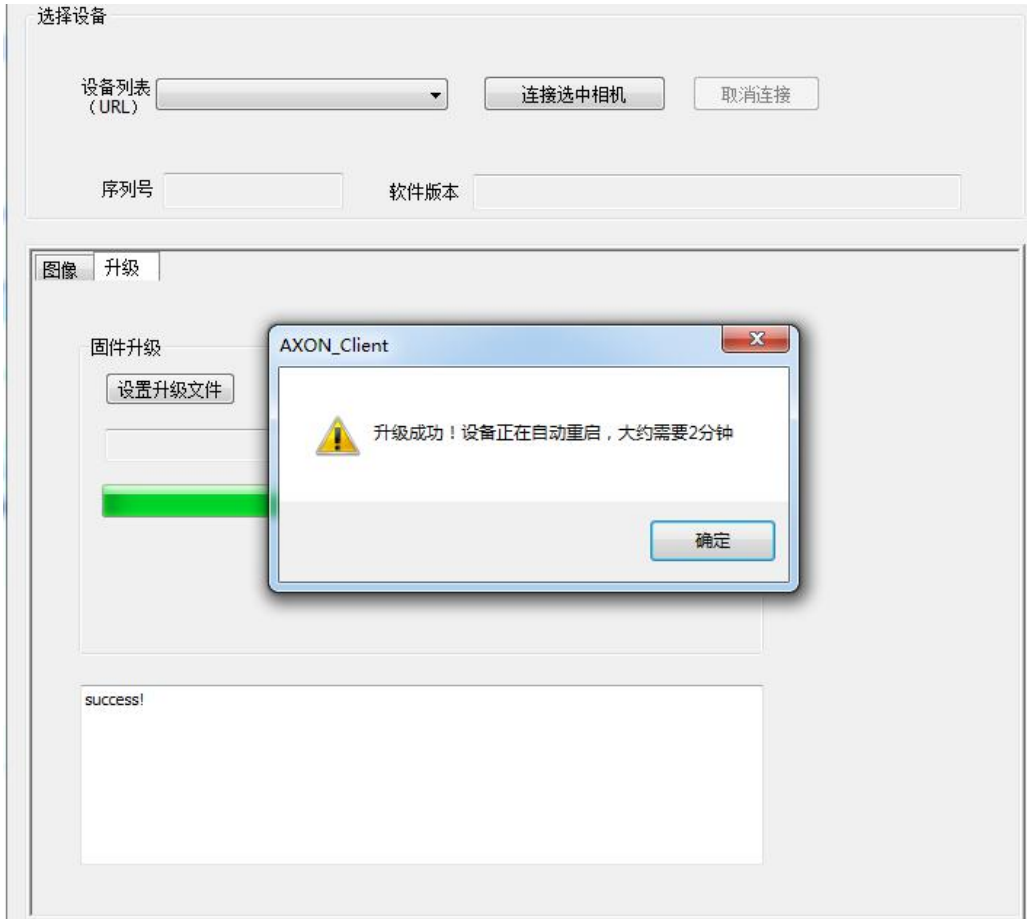



- 点击“设置升级文件”

选择要升级的镜像文件（扩展名“.img”）；点击“开始升级按钮”

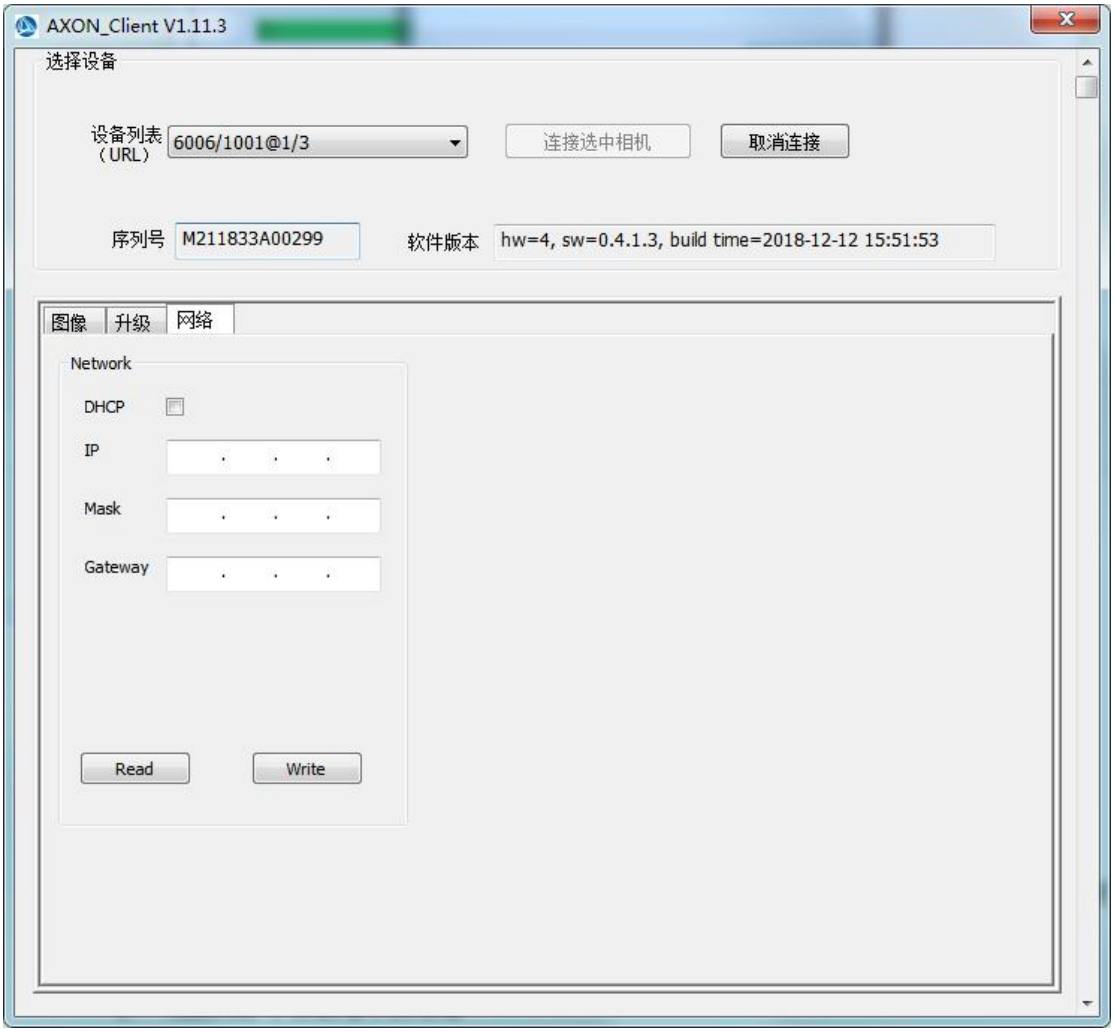



- 确定要升级，请点击“确定”，升级成功后有如下提示，点击“确定”，等待设备重启



## 5.3. 网络页面

可以设置设备的 IP 地址。注意，只有支持网络的设备和固件才支持这个功能。具体咨询支持工程师。




# 6. FAQ

**1. win10 系统更新后设备无法识别**

在 win10 环境下，若更新了系统，有可能会导致设备无法识别，此时尝试还原系统至更新前。

或者利用安装包（比如 AXonLink-OpenNI-Windows-x64-2.3.msi）卸载原来的安装，重启电脑，再次安装。

**2. Android 平台获取不到设备**

取得 root 权限后，关闭 Selinux。

**3. Linux 平台不能打开设备**

两种方法

（1） 应用程序使用 sudo 执行；

（2） 使用 sudo 执行根目录下 install.sh 脚本，重新插拔设备 USB。