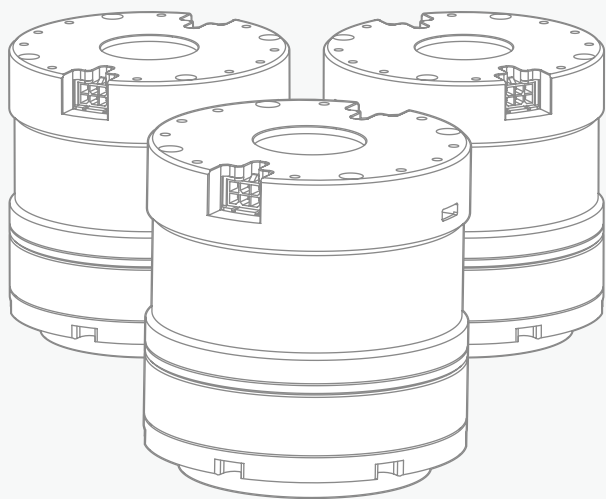




串口 & 以太网转 CAN 通信 SDK

[说明文档] V.01

Manual / 11-May 2018



执行器连接 / 环境配置和示例代码编译运行 / 环境配置 / SDK 编译 / 示例程序测试 / linux 平台 /
环境配置 / SDK 编译 / 示例程序测试 / SDK 说明 / 项目中使用 sdk / 命名空间 / 主要类说明

串口 & 以太网转 CAN 通信 SDK 说明文档

V0.1 / 08-Apr 2018

目 录

1. 版本信息	01
2. 目的	01
3. 目录结构	01
4. 执行器连接	01
5. 环境配置和示例代码编译运行	01
5.1 windows 平台	01
5.1.1 环境配置	01
5.1.2 SDK 编译	06
5.1.3 示例程序测试	08
5.2 linux 平台	10
环境配置	10
SDK 编译	11
示例程序测试	11
6.SDK 说明	14
1. 介绍	14
2. 项目中使用 sdk	14
3. 命名空间	14
4. 主要类说明	19
5. 版本历史	25

1. 版本信息

版本	日期	修改内容
V2.0.0	2018-04-17	增加 API
V3.0.0	2018-05-28	去除对 QT 模块的依赖，增加 API

2. 目的

本文档为 INNOS 执行器 SDK 说明文档，用于执行器二次开发，编写查看、控制、调节执行器的应用。

3. 目录结构

主目录 serialPort_Ethernet2CAN_sdk_v.x.x.x，其中 x.x.x 为当前的 sdk 版本号

...example 为示例程序，...example\src 为示例程序源码

...sdk 为 SDK 相关的头文件和库文件，其中 ...sdk\include 包含了 SDK 需要的头文件，...sdk\lib 包含了 windows64 位系统和 linux 64 位系统的库文件

...tools 包含了 windows 下用到的 vs2015 64 位版本运行时库

...readme.txt 包含了一些使用 SDK 需要注意的事项

4. 执行器连接

5. 环境配置和示例代码编译运行

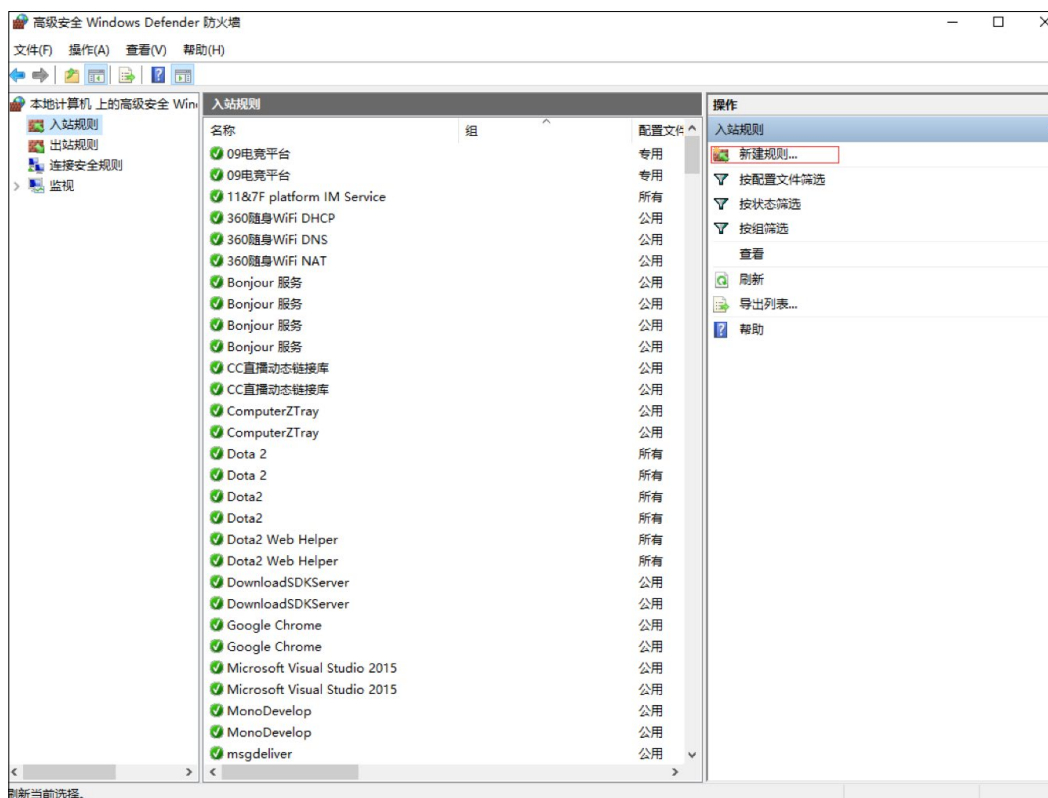
5.1 windows 平台

5.1.1 环境配置

- 1.SDK 需要 win7 sp1 以上的 64 位 windows 操作系统
2. 安装 visual studio 2015
3. 安装 cmake: 在 Cmake 官网 <https://cmake.org/download/> 下载最新版本 Cmake 安装
4. 如果电脑防火墙已开启，打开控制面板，选择系统和安全，打开 Windows Defender 防火墙，点击左侧高级设置，点击入站规则



新建规则



选择端口，然后点击下一步



选择 udp，并选择所有本地端口，点击下一步



新建入站规则向导

协议和端口

指定应用此规则的协议和端口。

步骤:

- 规则类型
- 协议和端口
- 操作
- 配置文件
- 名称

此规则应用于 TCP 还是 UDP?

☐ TCP

☒ UDP

此规则应用于所有本地端口还是特定的本地端口?

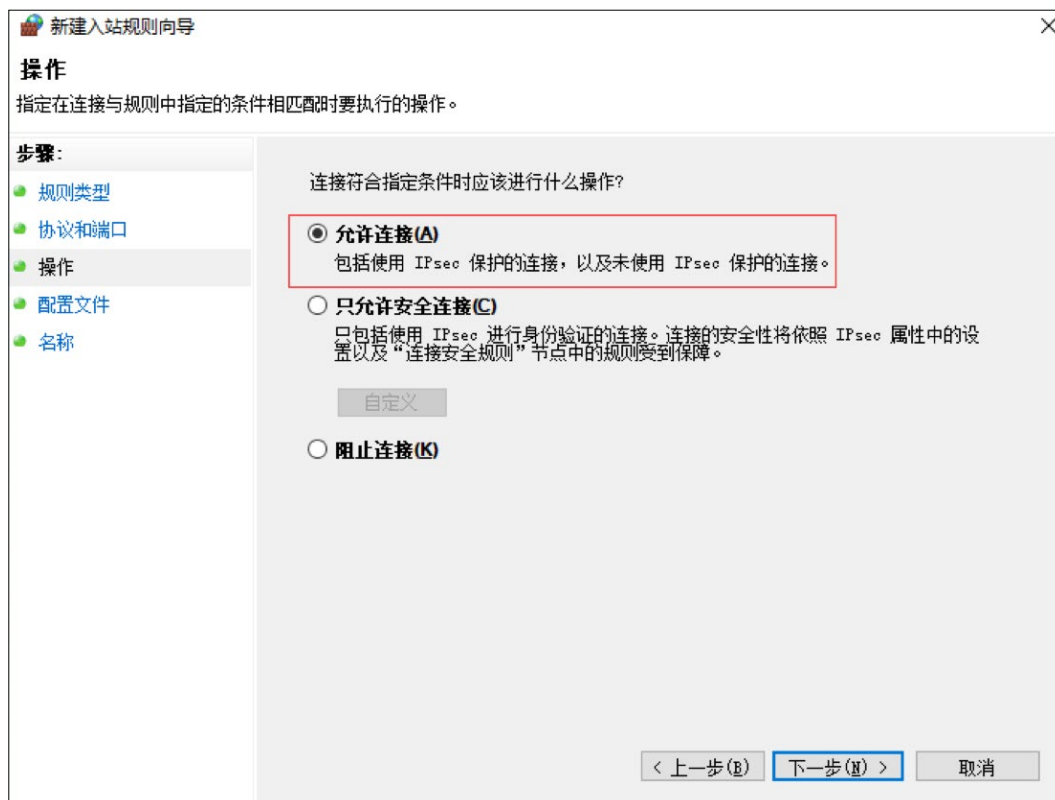
☒ 所有本地端口(A)

☐ 特定本地端口(S):

示例: 80, 443, 5000-5010

< 上一步(B) 下一步(N) > 取消

允许连接，点击下一步，



新建入站规则向导

操作

指定在连接与规则中指定的条件相匹配时要执行的操作。

步骤:

- 规则类型
- 协议和端口
- 操作
- 配置文件
- 名称

连接符合指定条件时应该进行什么操作?

☒ 允许连接(A)

包括使用 IPsec 保护的连接，以及未使用 IPsec 保护的连接。

☐ 只允许安全连接(C)

只包括使用 IPsec 进行身份验证的连接。连接的安全性将依照 IPsec 属性中的设置以及“连接安全规则”节点中的规则受到保障。

自定义

☐ 阻止连接(K)

< 上一步(B) 下一步(N) > 取消

默认勾选所有设置，点击下一步

新建入站规则向导

配置文件

指定此规则应用的配置文件

步骤：

- 规则类型
- 协议和端口
- 操作
- 配置文件
- 名称

何时应用该规则？

☒ 域(D)

计算机连接到其企业域时应用。

☒ 专用(P)

计算机连接到专用网络位置(例如，家或工作单位)时应用。

☒ 公用(U)

计算机连接到公用网络位置时应用。

< 上一步(B)

下一步(N) >

取消

填写自定义名字，完成即可

新建入站规则向导

名称

指定此规则的名称和描述。

步骤：

- 规则类型
- 协议和端口
- 操作
- 配置文件
- 名称

名称(N):

INNFOSS

描述(可选)(O):

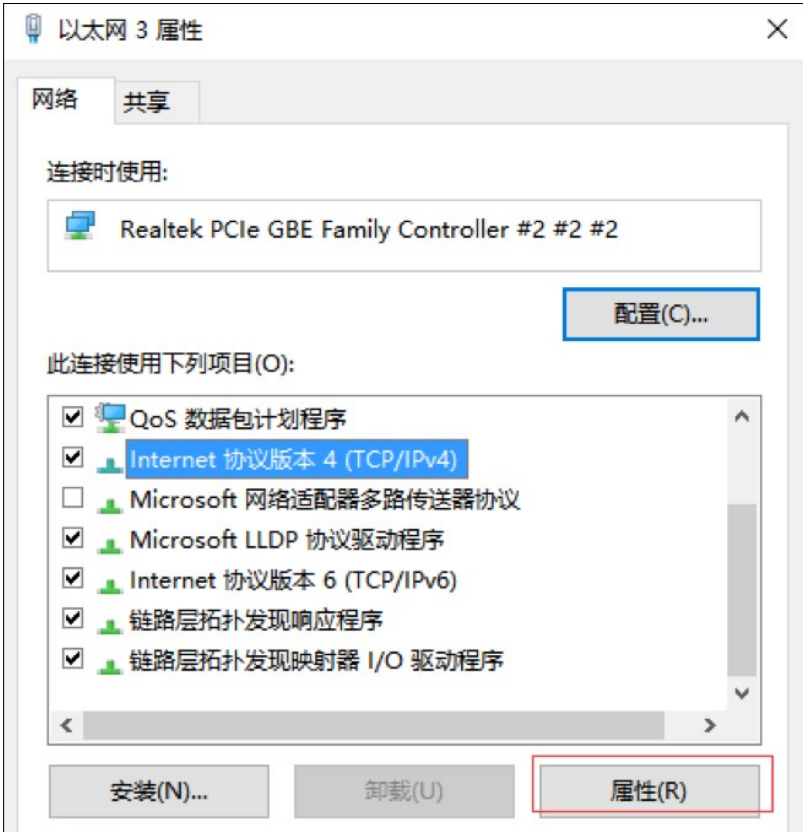
< 上一步(B)

完成(F)

取消

5.ip 地址配置，

打开控制面板，选择网络和 Internet, 再选择网络和共享中心，再选择更改适配器设置，右键单击以太网，选择属性



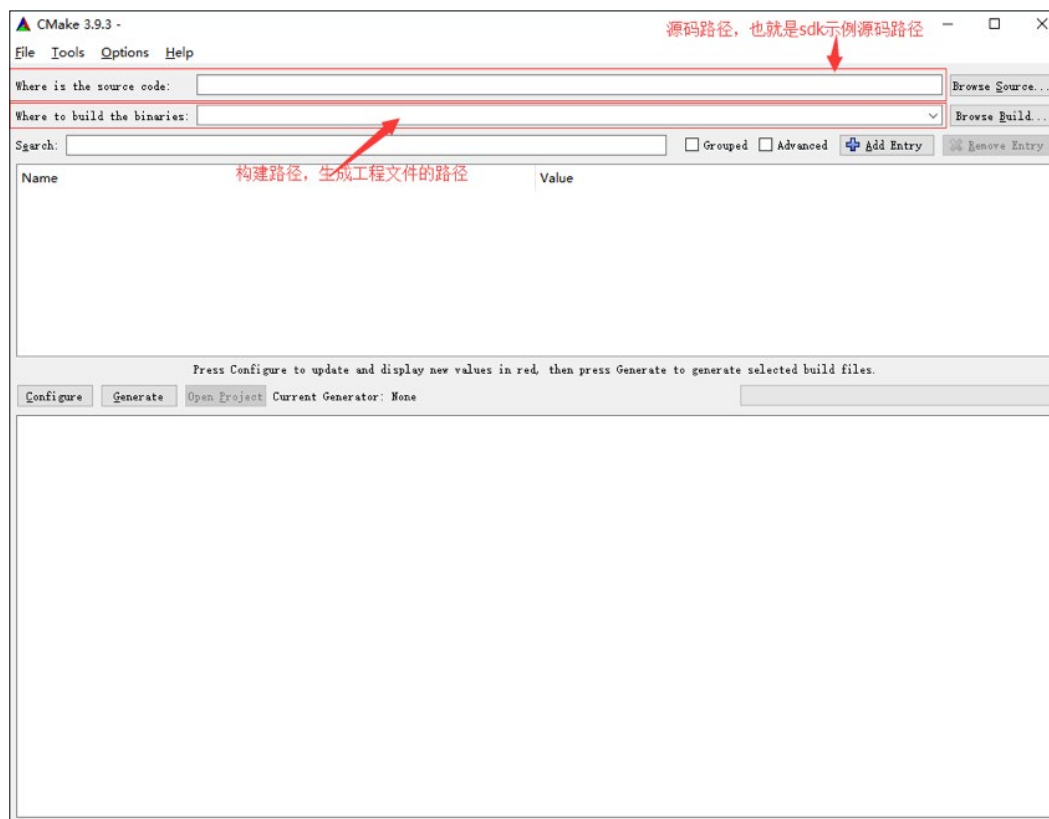
选择 TCP/IPv4, 然后选择属性，配置如图：

其中 ip 地址中的 192.168.1.119 中的 119 可以替换成 100~200 之间的任意整数，配置完成点击确定



5.1.2 SDK 编译

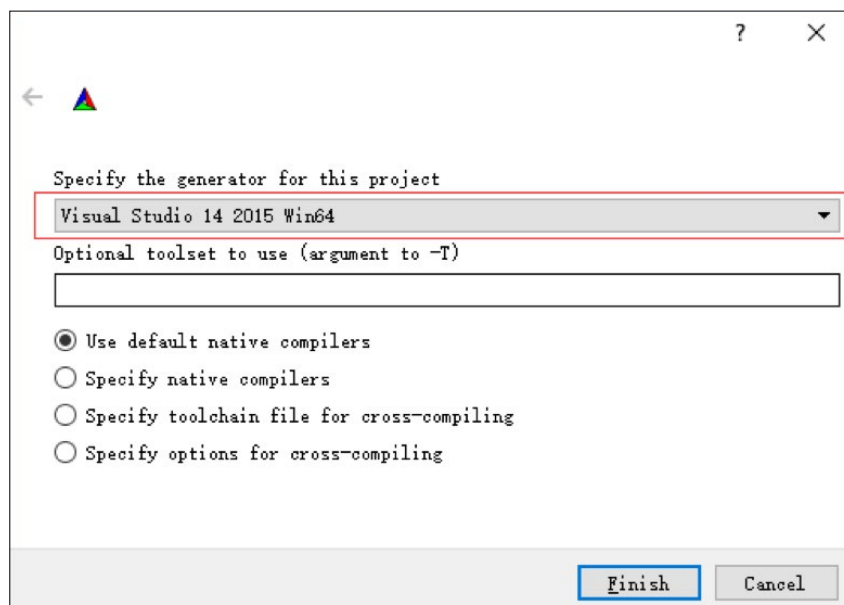
运行 cmake-gui 出现如下界面：



其中源码路径就是目录结构中的 ... \example 所在的路径，该目录下包含了 CMakeLists.txt 文件；

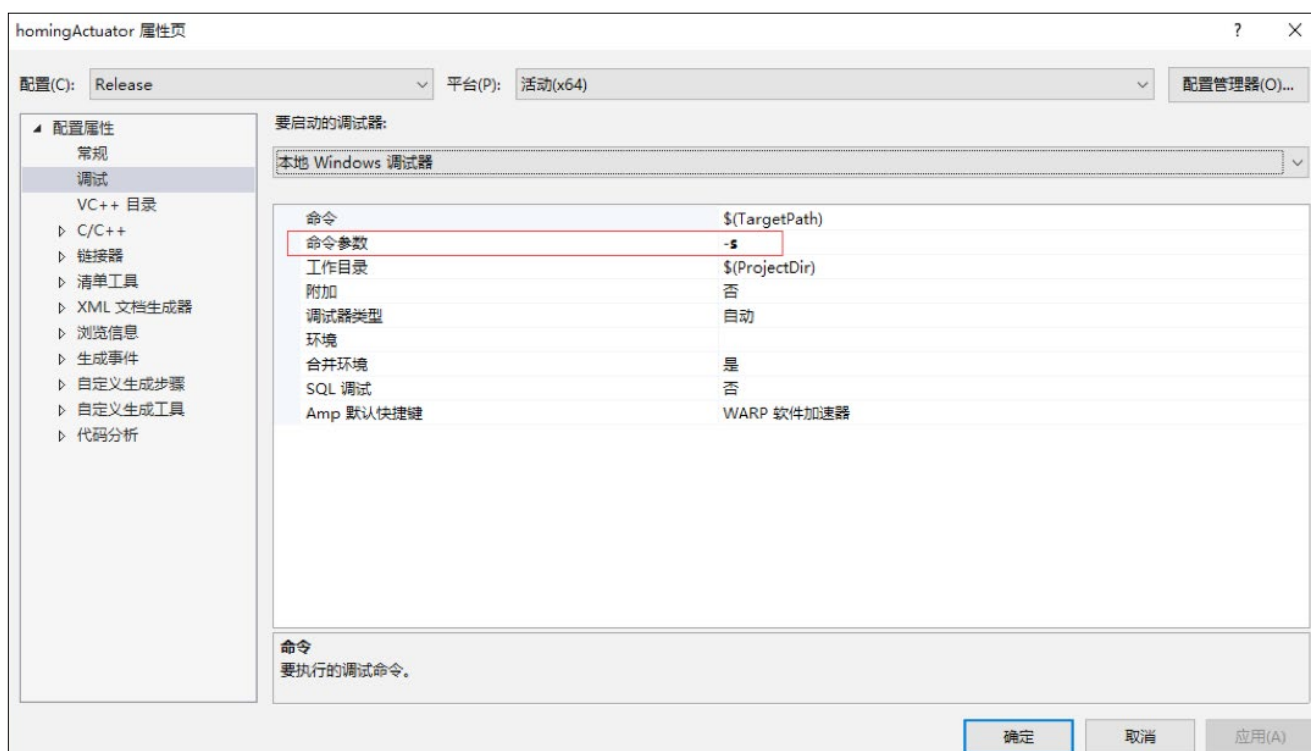
构建路径可自行定义，用于生成工程文件

两个路径配置完成后点击 Generate 按钮弹出如下界面



如果红色框内不是 64 位生成器，点击下拉三角，选择 64 位生成器，然后点击 Finish 按钮，生成成功后就生成了 Visual Studio 的工程文件，可用 Visual Studio 打开编译，右键点击项目 -> 属性。

配置属性 -> 调试当中的命令参数中需要添加命令参数: -s 代表 USB 串口通信方式通信; -e 代表通过以太网方式通信, 根据执行器的连接方式选择不同命令参数 (五个示例项目 homingActuator、lookupActuators、monitorActuator、operateActuator、tuneActuator 都需要增加命令参数) :



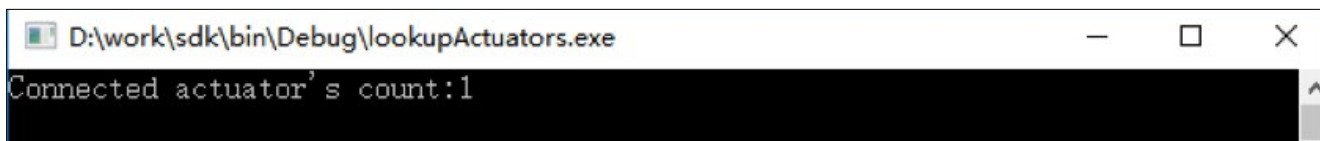
编译整个工程, 在工程目录下会生成一个 bin 目录, 里面有 Debug 或者 Release 文件夹 (对应于编译的版本), 将某一个项目设置为启动项, 就可以执行对应的示例程序了。

5.1.3 示例程序测试

确认执行器正确连接并供电以后，执行器会有黄色指示灯闪烁，此时可以测试示例代码。

1. 查找已连接的执行器

运行项目 lookupActutors，弹出 cmd 窗口



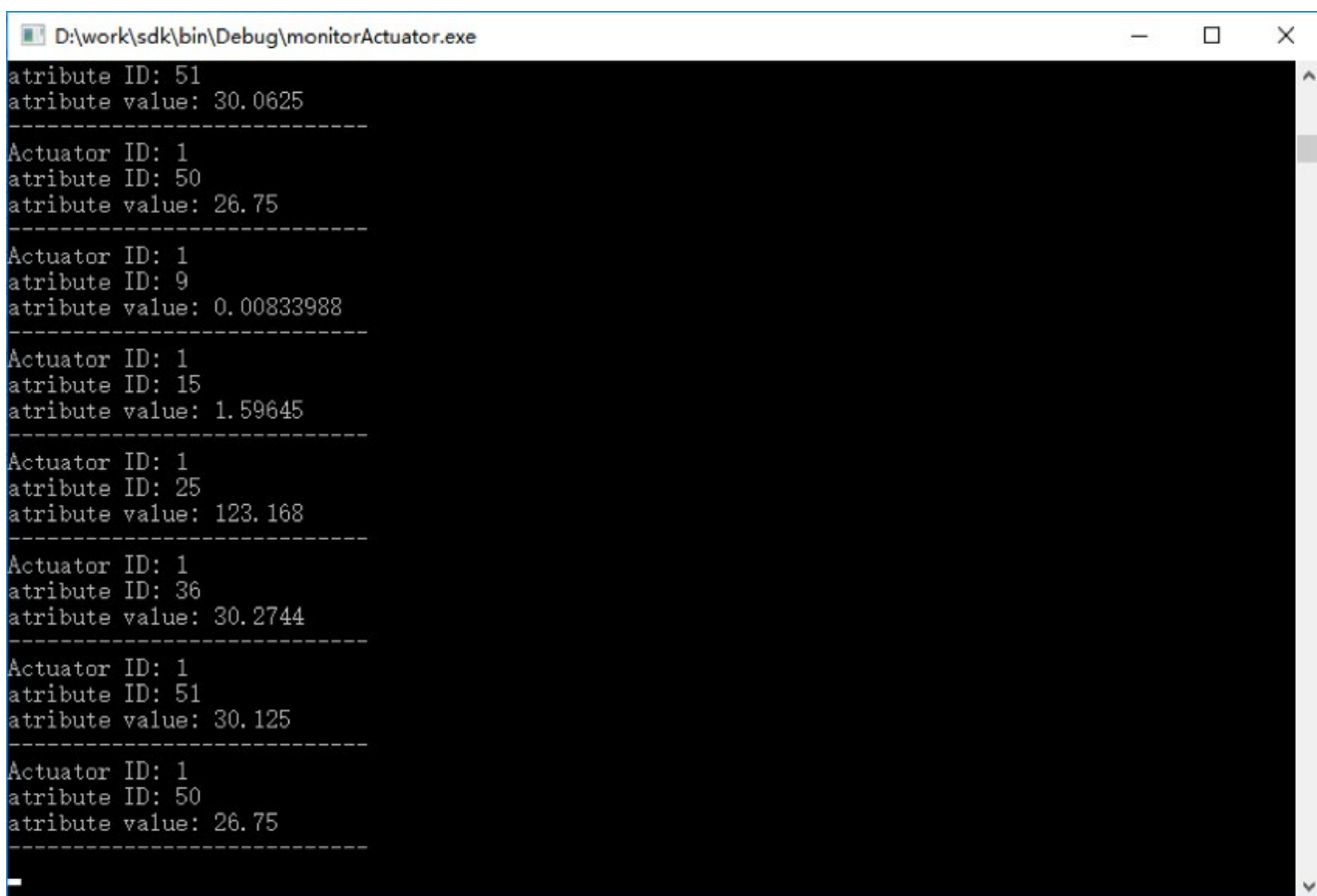
此窗口会显示当前已连接的执行器数量，可以 ctrl+c 结束程序（如果此时报错，可按快捷键 ctrl+alt+e 弹出异常设置窗口



将红框内的异常选中取消，就可以消除错误）

2. 监测执行器状态

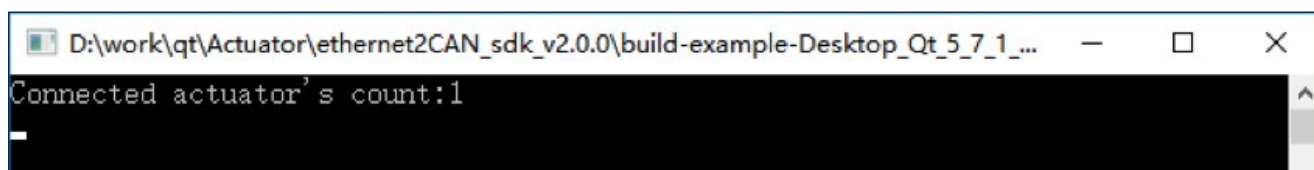
运行项目 monitorActuator，弹出 cmd 窗口



其中 Actuator ID 为执行器 id,attribute ID 为监测的执行器属性 Id，attribute value 为对应的属性值，可以 ctrl+c 结束程序

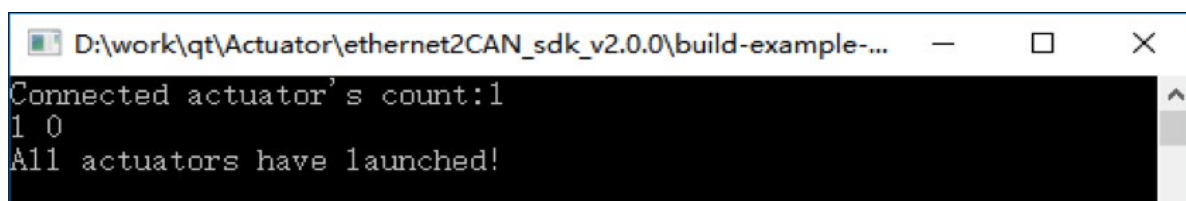
3. 控制执行器

运行项目 operateActuator, 弹出 cmd 窗口



```
D:\work\qt\Actuator\ethernet2CAN_sdk_v2.0.0\build-example-Desktop_Qt_5_7_1_...
Connected actuator's count:1
```

表示执行器已经找到, 输入命令 l 0, 该命令会启动所有已连接的执行器, 如果启动成功, 执行器会有绿色指示灯闪烁, 表示已经启动成功, cmd 窗口如下显示

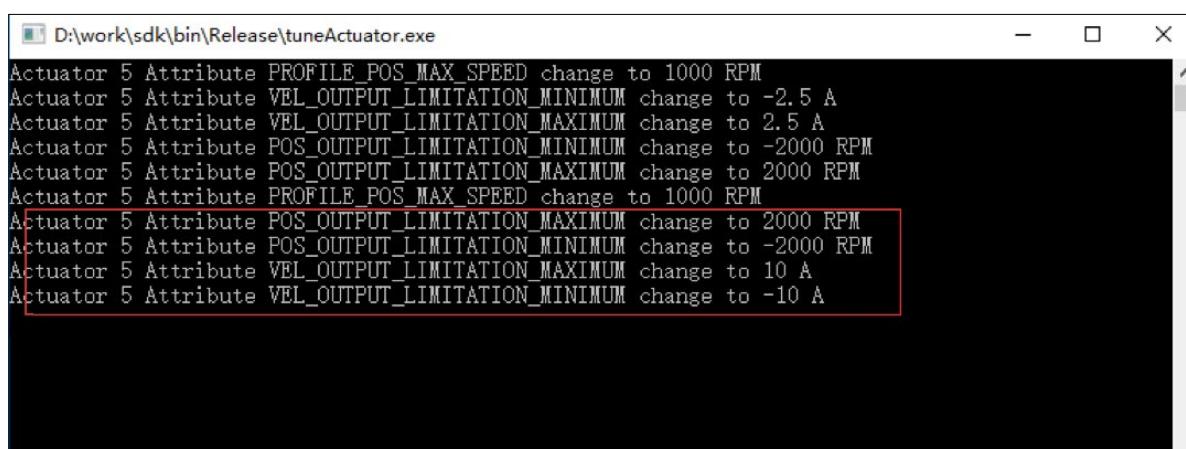


```
D:\work\qt\Actuator\ethernet2CAN_sdk_v2.0.0\build-example-...
Connected actuator's count:1
l 0
All actuators have launched!
```

此时可激活执行器对应模式, 比如输入 a 6 可以激活 profile position 模式, 再输入 p 2, 执行器会转动到 2 圈的位置; 输入 a 7 可以激活 profile velocity 模式, 再输入 v 200, 执行器将以 200RPM 的速度转动, 停止转动输入 v 0; 输入 a 1 可以激活电流模式, 再输入 c 0.6, 执行器将以恒定 0.6A 的电流转动 (如果执行器不动, 可用手轻轻转动一下执行器), 可以 ctrl+c 结束程序

4. 控制器参数调整

运行项目 tuneActuator. 弹出 cmd 窗口

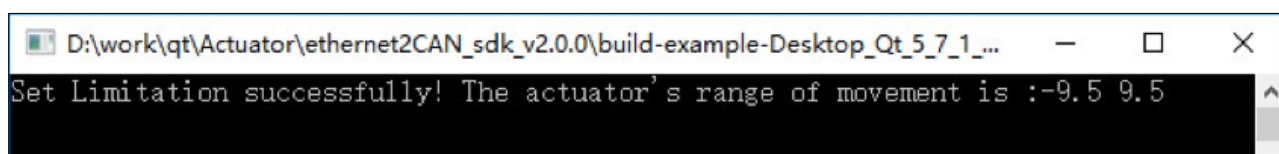


```
D:\work\sdk\bin\Release\tuneActuator.exe
Actuator 5 Attribute PROFILE_POS_MAX_SPEED change to 1000 RPM
Actuator 5 Attribute VEL_OUTPUT_LIMITATION_MINIMUM change to -2.5 A
Actuator 5 Attribute VEL_OUTPUT_LIMITATION_MAXIMUM change to 2.5 A
Actuator 5 Attribute POS_OUTPUT_LIMITATION_MINIMUM change to -2000 RPM
Actuator 5 Attribute POS_OUTPUT_LIMITATION_MAXIMUM change to 2000 RPM
Actuator 5 Attribute PROFILE_POS_MAX_SPEED change to 1000 RPM
Actuator 5 Attribute POS_OUTPUT_LIMITATION_MAXIMUM change to 2000 RPM
Actuator 5 Attribute POS_OUTPUT_LIMITATION_MINIMUM change to -2000 RPM
Actuator 5 Attribute VEL_OUTPUT_LIMITATION_MAXIMUM change to 10 A
Actuator 5 Attribute VEL_OUTPUT_LIMITATION_MINIMUM change to -10 A
```

此示例程序自动启动执行器并将位置环速度输出设置为 2000RPM, 最小输出速度设置为 -2000RPM 速度环的电流最大输出为 10A, 最小为 -10A, 如果使用 profile position 模式转动执行器, 执行器的最大速度不会超过 2000RPM; 如果使用 profile velocity 模式转动执行器, 执行器最大电流不会超过 10A, 可以 ctrl+c 结束程序

5. 执行器归零

运行项目 homingActuator, 弹出 cmd 窗口



```
D:\work\qt\Actuator\ethernet2CAN_sdk_v2.0.0\build-example-Desktop_Qt_5_7_1_...
Set Limitation successfully! The actuator's range of movement is :-9.5 9.5
```

表示已经将执行器当前位置设置为零位, 范围是 -9.5R 到 9.5R, 并且开启了位置限制, 如果 profile position 模式下, 输入此范围之外的位置, 执行器不会转动, 可以 ctrl+c 结束程序

5.2 linux 平台

环境配置

1. 本文档使用的是 ubuntu16.04 LTS 系统
2. cmake 安装: 打开终端输入命令 `sudo apt-get install cmake`
3. ip 地址配置: 打开终端输入 `ifconfig`, 查看网络配置

```
innfos@innfos-ThinkPad-T410s: ~
innfos@innfos-ThinkPad-T410s: ~ 93x31
innfos@innfos-ThinkPad-T410s:~$ ifconfig
enp0s25  Link encap:Ethernet  HWaddr f0:de:f1:3f:81:ae
        UP BROADCAST MULTICAST  MTU:1500  Metric:1
        RX packets:20 errors:0 dropped:0 overruns:0 frame:0
        TX packets:135 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:2002 (2.0 KB)  TX bytes:21602 (21.6 KB)
        Interrupt:20 Memory:f2500000-f2520000

lo        Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:65536  Metric:1
        RX packets:7253 errors:0 dropped:0 overruns:0 frame:0
        TX packets:7253 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:784419 (784.4 KB)  TX bytes:784419 (784.4 KB)

wlp3s0    Link encap:Ethernet  HWaddr 18:3d:a2:0b:58:1c
        inet addr:192.168.2.112  Bcast:192.168.2.255  Mask:255.255.255.0
        inet6 addr: fe80::78d:e776:92f1:261/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:1457539 errors:0 dropped:0 overruns:0 frame:0
        TX packets:104937 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:240542525 (240.5 MB)  TX bytes:12525127 (12.5 MB)

innfos@innfos-ThinkPad-T410s:~$
```

示例中有线网卡的名字是 enp0s25, 输入命令 `sudo ifconfig enp0s25 static 192.168.1.111`

```
innfos@innfos-ThinkPad-T410s: ~
innfos@innfos-ThinkPad-T410s: ~ 93x31
        TX packets:135 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:2002 (2.0 KB)  TX bytes:21602 (21.6 KB)
        Interrupt:20 Memory:f2500000-f2520000

lo        Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:65536  Metric:1
        RX packets:7253 errors:0 dropped:0 overruns:0 frame:0
        TX packets:7253 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:784419 (784.4 KB)  TX bytes:784419 (784.4 KB)

wlp3s0    Link encap:Ethernet  HWaddr 18:3d:a2:0b:58:1c
        inet addr:192.168.2.112  Bcast:192.168.2.255  Mask:255.255.255.0
        inet6 addr: fe80::78d:e776:92f1:261/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:1457539 errors:0 dropped:0 overruns:0 frame:0
        TX packets:104937 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:240542525 (240.5 MB)  TX bytes:12525127 (12.5 MB)

innfos@innfos-ThinkPad-T410s:~$ ifconfig enp0s25 static 192.168.1.111
SIOCSIFADDR: Operation not permitted
SIOCSIFFLAGS: Operation not permitted
SIOCSIFADDR: Operation not permitted
SIOCSIFFLAGS: Operation not permitted
innfos@innfos-ThinkPad-T410s:~$ sudo ifconfig enp0s25 static 192.168.1.111
[sudo] password for innfos:
innfos@innfos-ThinkPad-T410s:~$
```

配置完成后输入 ifconfig，可看到配置成功后的 ip 地址

```
innfos@innfos-ThinkPad-T410s: ~
innfos@innfos-ThinkPad-T410s: ~ 93x31
innfos@innfos-ThinkPad-T410s:~$ sudo ifconfig enp0s25 static 192.168.1.111
[sudo] password for innfos:
innfos@innfos-ThinkPad-T410s:~$ ifconfig
enp0s25  Link encap:Ethernet  HWaddr f0:de:f1:3f:81:ae
          inet addr:192.168.1.111  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:20 errors:0 dropped:0 overruns:0 frame:0
          TX packets:135 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2002 (2.0 KB)  TX bytes:21602 (21.6 KB)
          Interrupt:20 Memory:f2500000-f2520000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:7279 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7279 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:786843 (786.8 KB)  TX bytes:786843 (786.8 KB)

wlp3s0    Link encap:Ethernet  HWaddr 18:3d:a2:0b:58:1c
          inet addr:192.168.2.112  Bcast:192.168.2.255  Mask:255.255.255.0
          inet6 addr: fe80::78d:e776:92f1:261/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1458760 errors:0 dropped:0 overruns:0 frame:0
          TX packets:105006 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:240745298 (240.7 MB)  TX bytes:12536061 (12.5 MB)

innfos@innfos-ThinkPad-T410s:~$
```

4. 串口通信方式可能存在读写权限问题，执行命令 `sudo usermod -aG dialout xxx,xxx` 为当前用户名，执行完成以后 `logout` 一下，就可以获得串口的读写权限了

SDK 编译

打开终端进入 `...example` 目录，该目录下有 `CMakeLists.txt`，输入命令 `cmake CMakeLists.txt`，执行成功后，再输入命令 `make`，执行完成后，在该目录下会生成一个 `bin` 文件夹，该目录存放了生成的示例程序。

确认执行器正确连接并供电以后，执行器会有黄色指示灯闪烁，此时可以测试示例代码。

示例程序测试

1. 查找已连接的执行器

打开终端，进入 `example/bin` 目录，

输入命令 `./lookupActuators -e` (串口通信方式的命令为 `./lookupActuators -s`)

```
liangzhenjie@liangzhenjie-virtual-machine: ~/work/ActuatorController_SDK/serialPort_Eth
liangzhenjie@liangzhenjie-virtual-machine:~/work/ActuatorController_SDK/serialPo
rt_Ethernet2CAN_sdk_v3.0.0/example/bin$ ./lookupActuators -e
Number of connected actuators:1
```

此窗口会显示当前已连接的执行器数量，可以 `ctrl+c` 结束程序

2. 监测执行器状态

打开终端，进入 example/bin 目录，

输入命令 ./monitorActuator -e(串口通信方式的命令为 ./monitorActuator -s)

```
liangzhenjie@liangzhenjie-virtual-machine: ~/work/ActuatorController_SDK/serialPort_Eth
liangzhenjie@liangzhenjie-virtual-machine:~/work/ActuatorController_SDK/serialPort_Ethernet2CAN_sdk_v3.0.0/example/bin$ ./monitorActuator -e
Actuator ID: 0
attribute ID: 61
attribute value: 5
-----
Actuator ID: 5
attribute ID: 62
attribute value: 4.29497e+09
-----
Actuator ID: 5
attribute ID: 59
attribute value: 11
-----
Actuator ID: 5
attribute ID: 58
attribute value: 0
-----
Number of connected actuators:1
Actuator ID: 5
attribute ID: 58
attribute value: 1
-----
Actuator ID: 5
```

3. 控制执行器

打开终端，进入 example/bin 目录，

输入命令 ./operateActuator -e(串口通信方式的命令为 ./operateActuator -s)

```
liangzhenjie@liangzhenjie-virtual-machine: ~/work/ActuatorController_SDK/serialPort_Eth
liangzhenjie@liangzhenjie-virtual-machine:~/work/ActuatorController_SDK/serialPort_Ethernet2CAN_sdk_v3.0.0/example/bin$ ./operateActuator -e
Number of connected actuators:1
```

表示执行器已经找到，输入命令 l 0(此处命令第一个字符为小写英文字母 l, 代表 launch)，该命令会启动所有已连接的执行器，如果启动成功，执行器会有绿色指示灯闪烁，表示已经启动成功，终端窗如下显示

```
liangzhenjie@liangzhenjie-virtual-machine: ~/work/ActuatorController_SDK/serialPort_Eth
liangzhenjie@liangzhenjie-virtual-machine:~/work/ActuatorController_SDK/serialPort_Ethernet2CAN_sdk_v3.0.0/example/bin$ ./operateActuator -e
Number of connected actuators:1
l 0
All actuators have launched!
```

此时可激活执行器对应模式，比如输入 a 6 可以激活 profile position 模式，再输入 p 2，执行器会转动到 2 圈的位置；输入 a 7 可以激活 profile velocity 模式，再输入 v 200，执行器将以 200RPM 的速度转动，停止转动输入 v 0；输入 a 1 可以激活电流模式，再输入 c 0.6，执行器将以恒定 0.6A 的电流转动（如果执行器不动，可用手轻轻转动一下执行器），可以 ctrl+c 以后再 ctrl+d 结束程序（因为有多线程等待键盘输入）

```
liangzhenjie@liangzhenjie-virtual-machine: ~/work/ActuatorController_SDK/serialPort_Eth
liangzhenjie@liangzhenjie-virtual-machine:~/work/ActuatorController_SDK/serialPo
rt_Ethernet2CAN_sdk_v3.0.0/example/bin$ ./operateActuator -e
Number of connected actuators:1
l 0
All actuators have launched!

a 6
p 2
a 7
v 200
v 0
a 1
c 0.6
c 0
s 0
```

4. 控制器参数调整

打开终端，进入 example/bin 目录，

输入命令 ./tuneActuator -e(串口通信方式的命令为 ./tuneActuator -s)

```
liangzhenjie@liangzhenjie-virtual-machine: ~/work/ActuatorController_SDK/serialPort_Eth
liangzhenjie@liangzhenjie-virtual-machine:~/work/ActuatorController_SDK/serialPo
rt_Ethernet2CAN_sdk_v3.0.0/example/bin$ ./tuneActuator -e
Actuator 5 Attribute POS_OUTPUT_LIMITATION_MAXIMUM change to 2000 RPM
Actuator 5 Attribute POS_OUTPUT_LIMITATION_MINIMUM change to -2000 RPM
Actuator 5 Attribute VEL_OUTPUT_LIMITATION_MAXIMUM change to 2.5 A
Actuator 5 Attribute VEL_OUTPUT_LIMITATION_MINIMUM change to -2.5 A
Actuator 5 Attribute PROFILE_POS_MAX_SPEED change to 1000 RPM
Actuator 5 Attribute PROFILE_POS_MAX_SPEED change to 1000 RPM
Actuator 5 Attribute POS_OUTPUT_LIMITATION_MAXIMUM change to 2000 RPM
Actuator 5 Attribute POS_OUTPUT_LIMITATION_MINIMUM change to -2000 RPM
Actuator 5 Attribute VEL_OUTPUT_LIMITATION_MAXIMUM change to 10 A
Actuator 5 Attribute VEL_OUTPUT_LIMITATION_MINIMUM change to -10 A
```

此示例程序自动启动执行器并将位置环速度输出设置为 2000RPM, 最小输出速度设置为 -2000RPM 速度环的电流最大输出为 10A, 最小为 -10A, 如果使用 profile position 模式转动执行器，执行器的最大速度不会超过 2000RPM; 如果使用 profile velocity 模式转动执行器，执行器最大电流不会超过 10A，可以 ctrl+c 结束程序

5. 执行器归零

打开终端，进入 example/bin 目录，

输入命令 ./homingActuator -e(串口通信方式的命令为 ./homingActuator -s)

```
liangzhenjie@liangzhenjie-virtual-machine: ~/work/ActuatorController_SDK/serialPort_Eth
liangzhenjie@liangzhenjie-virtual-machine:~/work/ActuatorController_SDK/serialPort_Ethernet2CAN_sdk_v3.0.0/example/bin$ ./homingActuator -e
Set Limitation successfully! The actuator's range of movement is :-9.5 9.5
```

表示已经将执行器当前位置设置为零位，范围是 -9.5R 到 9.5R，并且开启了位置限制，如果 profile position 模式下，输入此范围之外的位置，执行器不会转动，可以 ctrl+c 结束程序

6.SDK 说明

1. 介绍

本 SDK 提供了与 INNFOS 执行器通信的接口，可通过串口或者以太网对已经连接好的执行器进行查找、状态查询、属性调整和自定义控制。如果想快速了解 sdk 基本内容和使用方法，请查看 example/src 中的相关代码。

2. 项目中使用 sdk

本 sdk 遵循 c++11 标准，所以在构建项目之前请确认编译选项支持 c++11（比如 gcc 中使用 -std=c++11）；将 sdk 集成到项目中的基本步骤（最好先参考 example 中的 CMakeLists.txt）：

1. 将 sdk/include、sdk/include/asio 加入到项目的包含目录，用于关联共享库中的方法；
2. 将库文件目录 sdk/lib/linux_x86_64（windows 目录为 sdk/lib/debug 和 sdk/lib/release），以便可执行文件能链接到共享库，并保证运行时能够关联到共享库；
3. 将必要的元素加入到构建过程中（比如 CMake 中的 target_link_libraries）

3. 命名空间

在 ../sdk/include/actuatordefine.h 定义了命名空间 Actuator，并且枚举了 sdk 中所有用到的类型和类型值：

连接状态，用于执行器和 CAN 的连接状态判断 [ConnectStatus]	
指令符	说明
NO_CONNECT,	无连接
CAN_CONNECTED=0x02,	CAN 通信连接成功
ACTUATOR_CONNECTED=0x04,	执行器连接成功

通道 ID, 用于标识执行器图表数据的通道索引 [Channel_ID]

指令符	说明
channel_1=0,	图表数据 1 通道, 给定理想曲线
channel_2,	图表数据 2 通道, 实际电流曲线
channel_3,	图表数据 3 通道, 实际速度曲线
channel_4,	图表数据 4 通道, 实际位置
channel_cnt	

错误类型定义, 定义了执行器内部和连接等错误代码 [ErrorsDefine]

指令符	说明
ERR_NONE = 0,	无错误
ERR_ACTUATOR_OVERVOLTAGE=0x01,	执行器过压错误
ERR_ACTUATOR_UNDERVOLTAGE=0x02,	执行器欠压错误
ERR_ACTUATOR_LOCKED_ROTOR=0x04,	执行器堵转错误
ERR_ACTUATOR_OVERHEATING=0x08	执行器过温错误
enum OnlineStatus{	执行器读写错误
ERR_ACTUATOR_MULTI_TURN=0x20,	执行器多圈计数错误
ERR_INVERTOR_TEMPERATURE_SENSOR=0x40,	执行器逆变器温度器错误
ERR_CAN_COMMUNICATION=0x80,	执行器温度传感器错误
ERR_ACTUATOR_TEMPERATURE_SENSOR=0x100,	执行器 CAN 通信错误
ERR_DRV_PROTECTION=0x400,	执行器 DRV 保护
ERR_ID_UNUNIQUE=0x800	执行器 ID 不唯一错误
ERR_ACTUATOR_DISCONNECTION=0x801,	执行器未连接错误
ERR_CAN_DISCONNECTION=0x802,	CAN 通信转换板未连接错误
ERR_IP_ADDRESS_NOT_FOUND=0x803,	无可用的 IP 地址错误
ERR_ABNORMAL_SHUTDOWN=0x804,	执行器非正常关机错误
ERR_SHUTDOWN_SAVING=0x805,	执行器关机时参数保存错误
ERR_UNKOWN=0xffff	未知错误

在线状态, 用于标识执行器是否处于连接状态 [OnlineStatus]

指令符	说明
Status_Online=0x00,	执行器在线
Status_Offline=0x01,	执行器离线

开关状态, 标识执行器的开关机状态 [SwitchStatus]

指令符	说明
ACTUATOR_SWITCH_OFF=0,	执行器已关机
ACTUATOR_SWITCH_ON=1,	执行器已开机

图表开关, 用于标识执行器图表功能的开启或关闭 [ChartSwitchStatus]

指令符	说明
CHART_SWITCH_OFF=0,	图表功能关闭, 不会产生图表数据
CHART_SWITCH_ON=1,	图表功能开启, 触发图表阈值会产生图表数据

电流环图表索引，用于标识电流图表是 IQ 值还是 ID 值 [CurrnetChart]

指令符	说明
IQ_CHART=0,	图表数据 2 通道，实际电流 IQ 曲线
ID_CHART=1,	图表数据 2 通道，实际电流 ID 曲线

归零模式，分为手动和自动两种 [HomingOperationMode]

指令符	说明
Homing_Auto=0,	自动寻找执行器最小最大位置（暂未实现）
Homing_Manual,	手动寻找执行器最小最大位置

通信方式，可通过以太网或者串口两种方式与执行器通信，初始化执行器控制器时候要指定方式，默认为以太网通信 [CommunicationType]

指令符	说明
Via_Ethernet,	以太网通信
Via_Serialport,	串口通信

操作标识，标识操作完成，可用于判断执行器控制器的指令执行状态 [OperationFlags]

指令符	说明
Recognize_Finished	执行器启动完成（如果连接的是多个执行器，会触发多次启动完成信号）
Launch_Finished	执行器关闭完成（如果连接的是多个执行器，会触发多次关闭完成信号）
Close_Finished	执行器参数保存完成（如果连接的是多个执行器，会触发多次参数保存完成信号）
Save_Params_Finished	执行器参数保存失败
Save_Params_Failed	暂未实现
Attribute_Change_Finished	

执行器模式，标识当前执行器的模式 [ActuatorMode]

指令符	说明
Mode_None	
Mode_Cur	电流模式
Mode_Vel	速度模式
Mode_Pos	位置模式
Mode_Teaching	暂未实现
Mode_Profile_Pos=6	profile 位置模式，比较于位置模式，该模式有加速减速过程
Mode_Profile_Vel	profile 速度模式，比较于速度模式，该模式有加速减速过程
Mode_Homing	归零模式

执行器属性，标识了执行器所有相关属性 [ActuatorAttribute]

指令符	说明
Cur_iq_setting	电流 IQ 值
Cur_proportional	电流比例
Cur_integral	电流积分
Cur_id_setting	电流 ID 值
Cur_minimum	预留

执行器属性，标识了执行器所有相关属性 [ActuatorAttribute]	
指令符	说明
Cur_maximum	预留
Cur_nominal	预留
Cur_output	预留
Cur_maxspeed	电流环最大速度
Actual_current	当前电流值
Vel_setting	速度设置
Vel_proportional	速度比例
Vel_integral	速度积分
Vel_output_limitation_minimum	速度环输出最小电流比例
Vel_output_limitation_maximum	速度环输出最大电流比例
Actual_velocity	速度值
Pos_setting	位置设置
Pos_proportional	位置比例
Pos_integral	位置积分
Pos_differential	位置微分
Pos_output_limitation_minimum	位置环输出最小速度比例
Pos_output_limitation_maximum	位置环输出最大速度比例
Pos_limitation_minimum	最小位置限制
Pos_limitation_maximum	最大位置限制
Homing_position	归零位置
Actual_position	当前位置
Profile_pos_max_speed	profile position 模式最大速度
Profile_pos_acc	profile position 模式加速度
Profile_pos_dec	profile position 模式减速速度
Profile_vel_max_speed	profile velocity 模式最大速度
Profile_vel_acc	profile velocity 模式加速度
Profile_vel_dec	profile velocity 模式减速速度
Chart_frequency	图像频率
Chart_threshold	图像阈值
Chart_switch	图像开关
Pos_offset	位置偏移
Voltage	电压
Pos_limitation_switch	开启或关闭位置限制
Homing_cur_maximum	归零最大电流
Homing_cur_minimum	归零最小小电流
Current_scale	物理最大电流值

执行器属性，标识了执行器所有相关属性 [ActuatorAttribute]	
指令符	说明
Velocity_scale	速度最大电流值
Filter_c_status	电流环滤波是否开启
Filter_c_value	电流环滤波值
Filter_v_status	速度环滤波是否开启
Filter_v_value	速度环滤波值
Filter_p_status	位置环滤波是否开启
Filter_p_value	位置环滤波值
Inertia	惯量
Lock_energy	堵转保护能量
Actuator_temperature	执行器温度
Inverter_temperature	逆变器温度
Actuator_protect_temperature	执行器保护温度
Actuator_recovery_temperature	执行器恢复温度
Inverter_protect_temperature	逆变器保护温度
Inverter_recovery_temperature	逆变器恢复温度
Calibration_switch	预留
Calibration_angle	预留
Actuator_switch	执行器开关机
Firmware_version	执行器固件版本
Online_status	执行器是否在线
Device_id	执行器 Id
Sn_id	执行器 SN 号
Mode_id	执行器当前模式
Error_id	错误代码
Reserve_0	预留
Reserve_1	预留
Reserve_2	预留
Reserve_3	预留
Data_cnt	属性数量
Data_chart	预留
Data_invalid	非法属性值

4. 主要类说明

4.1. 相关 API：用户与执行器进行的全部交互都在此类中实现。

static void initController(int &argc, char **argv, int nCommunicationType=Actuator::Via_Ethernet)

初始化控制器，使用控制器之前必须先初始化，通信方式可分为串口通信和以太网通信两种，默认为以太网通信

示例代码：

```
1.      int main(int argc, char *argv[])
2.      {
3.          // 初始化控制器
4.          ActuatorController::initController(argc,argv,Actuator::Via_Ethernet);
5.          ....
6.      }
```

static ActuatorController * getInstance();

获取控制器对象实例，用户只能通过此接口获取控制器对象实例，而不应该以 new 的方式获取

示例代码：

```
1.      int main(int argc, char *argv[])
2.      {
3.          // 初始化控制器
4.          ActuatorController::initController(argc,argv,Actuator::Via_Ethernet);
5.          ActuatorController * pController = ActuatorController::getInstance();
6.          ....
7.      }
```

static void processEvents();

处理控制器事件，控制器所有的信号通知以及执行器属性刷新都依赖此函数的调用，所以不应该阻塞该函数的调用

示例代码：

```
1.      ...
2.      // 执行控制器事件循环
3.      while (!bExit)
4.      {
5.          ActuatorController::processEvents();
6.      }
7.      ...
```

void autoRecognize();

识别所有可用设备，初始化完成后调用此函数，识别完成会触发触发 m_sOperationFinished 信号，操作类型为 [OperationFlags::Recognize_Finished](#)

示例代码：

```

1.     ActuatorController * pController = ActuatorController::getInstance();
2.     signal(SIGINT,processSignal);
3.     // 关联控制器的操作信号
4.     int nOperationConnection = pController->m_sOperationFinished->s_Connect([=](uint8_t nDeviceId,uint8_t operationType){
5.         switch (operationType) {
6.             case Actuator::Recognize_Finished:// 自动识别完成
7.                 if(pController->hasAvailableActuator())
8.                     {
9.                         vector<uint8_t> idArray = pController->getActuatorIdArray();
10.                        cout << "Number of connected actuators:" << idArray.size() << endl;
11.                    }
12.                break;
13.            default:
14.                break;
15.        }
16.    });

```

bool hasAvailableActuator()const;
当前控制器是否识别到可用执行器

vector<uint8_t> getActuatorIdArray()const;
获取当前控制器识别到的执行器 id 数组

void activeActuatorMode(vector<uint8_t> idArray, const Actuator::ActuatorMode nMode);
激活指定执行器的指定模式，激活成功后会触发 m_sActuatorAttrChanged 信号，
属性 id 值为 ActuatorAttribute::MODE_ID

void launchAllActuators();
启动所有已识别的执行器，每个执行器启动成功后会触发 m_sOperationFinished 信号，
操作类型为 OperationFlags::Launch_Finished，如果执行器处于开机状态，则不会触发信号

void closeAllActuators();
关闭所有已识别的执行器，每个执行器关闭成功后会触发 m_sOperationFinished 信号，
操作类型为 OperationFlags::Close_Finished

void launchActuator(uint8_t id);
启动指定 id 的执行器，执行器启动成功后会触发 m_sOperationFinished 信号，
操作类型为 OperationFlags::Launch_Finished，如果执行器处于开机状态，则不会触发信号

void closeActuator(uint8_t id);
关闭指定 id 的执行器，执行器关闭成功后会触发 m_sOperationFinished 信号，
操作类型为 OperationFlags::Close_Finished

void switchAutoRefresh(uint8_t id,bool bOpen);
开启或关闭指定 id 执行器的自动刷新功能，
自动请求设备电流、速度、位置、电压、温度、逆变器温度（默认关闭此功能）

void setAutoRefreshInterval(uint8_t id, uint32_t mSec);
设置指定 id 执行器的自动刷新时间间隔（默认时间间隔为 1s）

void setPosition(uint8_t id,double pos);
设置指定 id 执行器的的位置，范围是 -128 到 128,单位是 Revolution，为了执行效率，此指令不会触发信号

```
void setVelocity(uint8_t id,double vel);
```

设置指定 id 执行器的速度，单位是 RPM, 为了执行效率，此指令不会触发信号

```
void setCurrent(uint8_t id,double current);
```

设置指定 id 执行器的电流，单位是 A，为了执行效率，此指令不会触发信号

```
double getPosition(uint8_t id,bool bRefresh=false)const;
```

获取指定 id 执行器的当前位置，单位是 Revolution，因为请求返回存在延时，当前得到的值是上一次请求返回成功后的结果，bRefresh 如果为 true，调用此函数后会自动请求一次执行器当前位置

```
double getVelocity(uint8_t id,bool bRefresh=false)const;
```

获取指定 id 执行器的当前速度，单位是 RPM，因为请求返回存在延时，当前得到的值是上一次请求返回成功后的结果，bRefresh 如果为 true，调用此函数后会自动请求一次执行器当前速度

```
double getCurrent(uint8_t id,bool bRefresh=false)const;
```

获取指定 id 执行器的当前电流，单位是 A，因为请求返回存在延时，当前得到的值是上一次请求返回成功后的结果，bRefresh 如果为 true，调用此函数后会自动请求一次执行器当前电流

```
void setActuatorAttribute(uint8_t id,Actuator::ActuatorAttribute attrId,double value);
```

设置指定 id 执行器的指定属性 attrId 的值 value，成功后会触发 m_sActuatorAttrChanged 信号，信号的执行器 id，属性 id 和属性值对应于设置的值

```
double getActuatorAttribute(uint8_t id, Actuator::ActuatorAttribute attrId)const;
```

获取指定 id 执行器的指定属性 attrId 的值，因为请求返回存在延时，当前得到的值是上一次请求该属性返回成功后的结果

```
void saveAllParams(uint8_t id);
```

保存指定 id 执行器的当前设置参数，以便参数下次开机依然生效，保存成功后会触发 m_sOperationFinished 信号，操作类型为 [OperationFlags::Save_Params_Finished](#)，保存失败的操作类型则为 [OperationFlags::Save_Params_Failed](#)

```
void clearHomingInfo(uint8_t id);
```

清除指定 id 执行器的归零和左右位置限制等信息，清除以后，如果想开启位置限制功能，必须先设置好适当的零位和左右位置限制

```
void setHomingOperationMode(uint8_t id,uint8_t nMode);
```

目前只支持手动模式设置零位和左右位置限制，所以该函数暂未实现功能

```
void setMinPosLimit(uint8_t id);
```

将指定 id 执行器的当前位置设置执行器的最小位置限制，设置成功后会触发 m_sActuatorAttrChanged 信号，属性 id 值为 [ActuatorAttribute::POS_LIMITATION_MINIMUM](#)

```
void setMinPosLimit(uint8_t id,double posValue);
```

设置指定 id 执行器的最小位置限制，其值为 posValue，设置成功后会触发 m_sActuatorAttrChanged 信号，属性 id 值为 [ActuatorAttribute::POS_LIMITATION_MINIMUM](#)

```
void setMaxPosLimit(uint8_t id);
```

将指定 id 执行器的当前位置设置执行器的最大位置限制，设置成功后会触发 m_sActuatorAttrChanged 信号，属性 id 值为 [ActuatorAttribute::POS_LIMITATION_MAXIMUM](#)

```
void setMaxPosLimit(uint8_t id,double posValue);
```

设置指定 id 执行器的最大位置限制，其值为 posValue，设置成功后会触发 m_sActuatorAttrChanged 信号，属性 id 值为 [ActuatorAttribute::POS_LIMITATION_MAXIMUM](#)

```
void setHomingPosition(uint8_t id,double posValue);
```

设置指定 id 执行器的 posValue 为零位，设置成功后会触发 m_sActuatorAttrChanged 信号，属性 id 值为 [ActuatorAttribute::HOMING_POSITION](#)

void openChartChannel(uint8_t id,uint8_t nChannelId);

开启指定 id 执行器的指定通道为 nChannelId ([Channel_ID::channel_1](#) 到 [Channel_ID::channel_4](#)) 的图表通道，开启后如果触发了图表数据生成条件，会触发 m_sNewChartStart 信号，提示新的图表数据已经生成，然后会触发图表数据信号 m_sChartValueChange 信号

void closeChartChannel(uint8_t id, uint8_t nChannelId);

关闭指定 id 执行器的指定通道为 nChannelId ([Channel_ID::channel_1](#) 到 [Channel_ID::channel_4](#)) 的图表通道，关闭后将不会触发新的图表数据

void switchChartAllChannel(uint8_t id,bool bOn);

开启或关闭指定 id 执行器的所有图表通道

void setCurrentChartMode(uint8_t id, uint8_t mode);

设置指定 id 的电流图表数据的模式为 ID 模式或者 IQ 模式 ([CurrnetChart::ID_CHART](#)、[CurrnetChart::IQ_CHART](#)) 默认是 [CurrnetChart::ID_CHART](#)

void regainAttrbute(uint8_t id,uint8_t attrId);

请求刷新指定 id 执行器的指定属性 attrId，请求后成功返回会触发 m_sActuatorAttrChanged 信号，属性 id 值为 attrId

vector<uint16_t> getErrorHistory(uint8_t id);

获取指定 id 执行器的错误历史记录，记录为错误代码，具体含义可参考 Actuator 命名空间中的 [ErrorsDefine](#)

void reconnect(uint8_t id);

重新连接指定 id 执行器，重连成功后会触发 m_sActuatorAttrChanged 信号，属性 id 值为 [ActuatorAttribute::ONLINE_STATUS](#)，属性值为 [OnlineStatus::Status_Online](#)

void clearError(uint8_t id);

清除指定 id 执行器的错误，清除成功后会触发 m_sActuatorAttrChanged 信号，属性 id 值为 [ActuatorAttribute::ERROR_ID](#)，属性值为 [ErrorsDefine::ERR_NONE](#)

string versionString()const;

获取指定 sdk 的版本号字符串，格式为：主版本号.次版本号.发布版本号

double getMaxCurrent(uint8_t id)const;

获取指定 id 执行器的最大允许电流，单位为 A，该值仅与执行器型号有关，并且不可被修改

double getMaxVelocity(uint8_t id)const;

获取指定 id 执行器的最大允许速度，单位为 RPM，该值仅与执行器型号有关，并且不可被修改

double getMaxOutputCurrent(uint8_t id)const;

获取指定 id 执行器最大输出电流，单位为 A，该值不会大于执行器最大允许电流

bool setMaxOutputCurrent(uint8_t id,double maxOutputCurrent);

设置执行器最大输出电流，单位为 A，该值不会大于执行器最大允许电流且必须大于最小输出电流，如果设置的电流值有效，返回 true，否则返回 false，设置不会影响执行器

double getMinOutputCurrent(uint8_t id)const;

获取执行器最小输出电流，单位为 A，该值不会小于执行器最大允许电流的负值

bool setMinOutputCurrent(uint8_t id,double minOutputCurrent);

设置执行器最小输出电流，单位为 A，该值不会小于执行器最大允许电流的负值且必须小于最大输出电流，如果设置的电流值有效，返回 true，否则返回 false


```
double getMaxOutputVelocity(uint8_t id)const;
```

获取执行器最大输出速度，单位为 RPM，该值不会大于执行器最大允许速度

```
bool setMaxOutputVelocity(uint8_t id,double maxOutputVelocity);
```

设置执行器最大输出速度，单位为 RPM，该值不会大于执行器最大允许速度且必须大于最小输出速度，如果设置的速度值有效，返回 true，否则返回 false

```
double getMinOutputVelocity(uint8_t id)const;
```

获取执行器最小输出速度，单位为 RPM，该值不会小于执行器最大允许速度的负值

```
bool setMinOutputVelocity(uint8_t id,double minOutputVelocity);
```

设置执行器最最小输出速度，单位为 RPM，该值不会小于执行器最大允许速度的负值且必须小于最大输出速度，如果设置的速度值有效，返回 true，否则返回 false

```
void activeActuatorMode(uint8_t id, const Actuator::ActuatorMode nMode);
```

激活单个执行器的指定模式

4.2. 信号

```
CSignal<uint8_t,uint8_t> m_sOperationFinished;
```

操作完成信号，第一个 uint8_t 代表执行器 id，如果是 0 不代表特定执行器，第二个 uint8_t 代表操作类型

示例代码：

```
1.      ...
2.      // 关联控制器的操作信号
3.      int nOperationConnection = pController->m_sOperationFinished.s_Connect([&](uint8_t nDeviceId,uint8_t operationType){
4.          switch (operationType) {
5.              case Actuator::Recognize_Finished:// 自动识别完成
6.                  if(pController->hasAvailableActuator())
7.                  {
8.                      vector<uint8_t> idArray = pController->getActuatorIdArray();
9.                      cout << "Number of connected actuators:" << idArray.size() << endl;
10.                     foreach (uint8_t id, idArray) {
11.                         if(pController->getActuatorAttribute(id,Actuator::ACTUATOR_SWITCH)!=Actuator::ACTUATOR_SWITCH_OFF)
12.                         {
13.                             ++ nLaunchedActuatorCnt;
14.                             if(nLaunchedActuatorCnt == pController->getActuatorIdArray().size())// 所有执行器都已启动完成
15.                             {
16.                                 cout << "All actuators have launched!" << endl;
17.                             }
18.                         }
19.                     }
20.
21.                     break;
22.                 }
23.                 break;
24.                 case Actuator::Launch_Finished:
25.                     if(++nLaunchedActuatorCnt == pController->getActuatorIdArray().size())// 所有执行器都已启动完成
26.                     {
27.                         cout << "All actuators have launched!" << endl;
28.                     }
29.                     break;
30.                 default:
31.                     break;
32.             }
33.         });
34.      ...
```

CSignal<uint8_t,uint8_t,double> m_sRequestBack;

请求返回信号，保留信号，暂未实现

CSignal<uint8_t,uint16_t,std::string> m_sError;

错误信号：uint8_t 代表执行器 id，如果是 0 不代表特定执行器，uint16_t 代表错误代码，std::string 代表错误信息字符串

示例代码：

```
1.      ...
2.      // 关联错误信号
3.      int nErrorConnection = pController->m_sError.s_Connect([=])(uint8_t nDeviceld,uint16_t nErrorType,string errorInfo){
4.          if(nDeviceld==0)
5.          {
6.              cout << "Error: " << (int)nErrorType << " " << errorInfo << endl;
7.          }
8.          else
9.          {
10.             cout << "Actuator " << (int)nDeviceld << " " <<"error" << (int)nErrorType << " " << errorInfo << endl;
11.          }
12.      });
13.      ...
```

CSignal<uint8_t,uint8_t,double> m_sActuatorAttrChanged;

执行器属性变化信号：第一个 uint8_t 代表执行器 id，第二个 uint8_t 代表执行器属性 id，double 代表执行器该属性的值

示例代码：

```
1.      ...
2.      // 关联控制器控制的执行器属性变化信号
3.      int nAttrConnection = pController->m_sActuatorAttrChanged.s_Connect([=])(uint8_t nDeviceld,uint8_t nAttrId,double value){
4.          cout << "Actuator ID: " << (int)nDeviceld << endl;
5.          cout << "atribute ID: " << (int)nAttrId << endl;
6.          cout << "atribute value: " << value << endl;
7.          cout << "-----" << endl;
8.      });
9.      ...
```

CSignal<> m_sNewChartStart;

图表新周期开始信号，当执行器触发新的图表数据时，会触发此信号，代表会开始发送新周期的图表数据

CSignal<uint8_t,double> m_sChartValueChange;

图表数据信号，uint8_t 代表图表通道 id，double 代表图表数据，一个周期会有 200 个数据点

5. 版本历史

serialPort_Ethernet2CAN_sdk_2.0.0

合并串口和以太网通信；

包含 qt 模块头文件，不再必须安装 qt 开发环境

serialPort_Ethernet2CAN_sdk_3.0.0

去除 sdk 对 qt 的模块依赖，改用支持 c++11 标准的 aiso

控制器的信号不再在栈上分配，以免出现析构问题

增加 11 个新的 API：

```
double getMaxCurrent(uint8_t id)const
double getMaxVelocity(uint8_t id)const
double getMaxOutputCurrent(uint8_t id)const
bool setMaxOutputCurrent(uint8_t id,double maxOutputCurrent)
double getMinOutputCurrent(uint8_t id)const
bool setMinOutputCurrent(uint8_t id,double minOutputCurrent)
double getMaxOutputVelocity(uint8_t id)const
bool setMaxOutputVelocity(uint8_t id,double maxOutputVelocity)
double getMinOutputVelocity(uint8_t id)const
bool setMinOutputVelocity(uint8_t id,double minOutputVelocity)
void activeActuatorMode(uint8_t id, const Actuator::ActuatorMode nMode)
```