编号：

智能驾驶开发者平台 API

用户手册

比亚迪汽车工业有限公司

2018 年 09 月 28 日

智能驾驶开发者平台 API 用户手册

版本号	修改人	修改内容	生效日期
v0.1	何敏政	1） 去掉“目标方向盘角速度设置 API” 2） 去掉“获取多媒体触摸坐标信息 API” 3） 去掉“获取多媒体分辨率信息 API” 4） 去掉“获取多媒体动作信息 API” 5） 去掉“获取多媒体移动信息 API” 6） 去掉“获取障碍物距离信息 API” 7） 去掉“获取驻车辅助系统电源开关状态 API”	2018.08.23
v0.2	高上添	修改 3.3 章节“API 调用逻辑说明”	2018.09.28
v0.3	高上添	增加第 5 章节“API 例程说明”	2018.11.22
v0.4	何敏政	修改 4.4.12 章节的 API 函数名称	2018.12.07
V0.5	高上添	增加了 4.3.17 章节的横向控制 API 接口	2019.02.12

目 录

1 编写目的

比亚迪汽车工业有限公司为“自动驾驶领域”的广大开发者特别打造了一款车型配置 —— 秦 Pro-EV 开放版，这是一款全面支持线控的车型配置，具备了横向线控、纵向线控、人机交互线控等能力。在该车型配置上已经预留了 13 个摄像头、6 个毫米波雷达、以及 1 个激光雷达的线束，还包含部分的支架。开发者只需接上自己摄像头、毫米波雷达、以及激光雷达等传感器，再将自己的“自动驾驶系统 ECU”与车后备箱内的“安全网关”连接，就可以对整车进行控制。

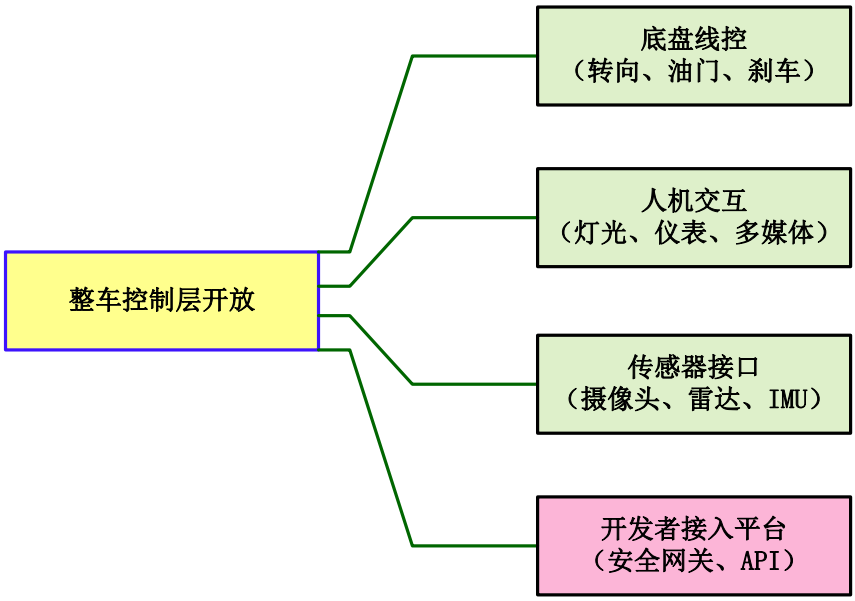
目前“安全网关”对外开放的接口形式为 CAN，开发者的“自动驾驶系统 ECU” 可以通过“安全网关”获取整车的状态信息、也可以通过“安全网关”向整车上的各个执行部件发送相关控制指令。为方便开发者对整车进行控制，比亚迪汽车工业有限公司给开发者提供了开发者平台 API 库，开发者的软件程序只需要调用特定的 API 接口函数就可以对整车进行相应的控制、以及从整车上获取相应的反馈信息，而不用关注繁琐的 CAN 报文数据的解析、以及 CAN 发送报文数据的封装打包。

本文档是开发者平台 API 库的用户手册，将向开发者介绍：

- ✧ 比亚迪“开发者平台 API 库”如何使用？
- ✧ 为了实现对整车精准、有效的控制，有哪些步骤和流程需要遵循？
- ✧ 开发者实施某种特定控制行为以后，应当如何从整车去采集相应的反馈信号？

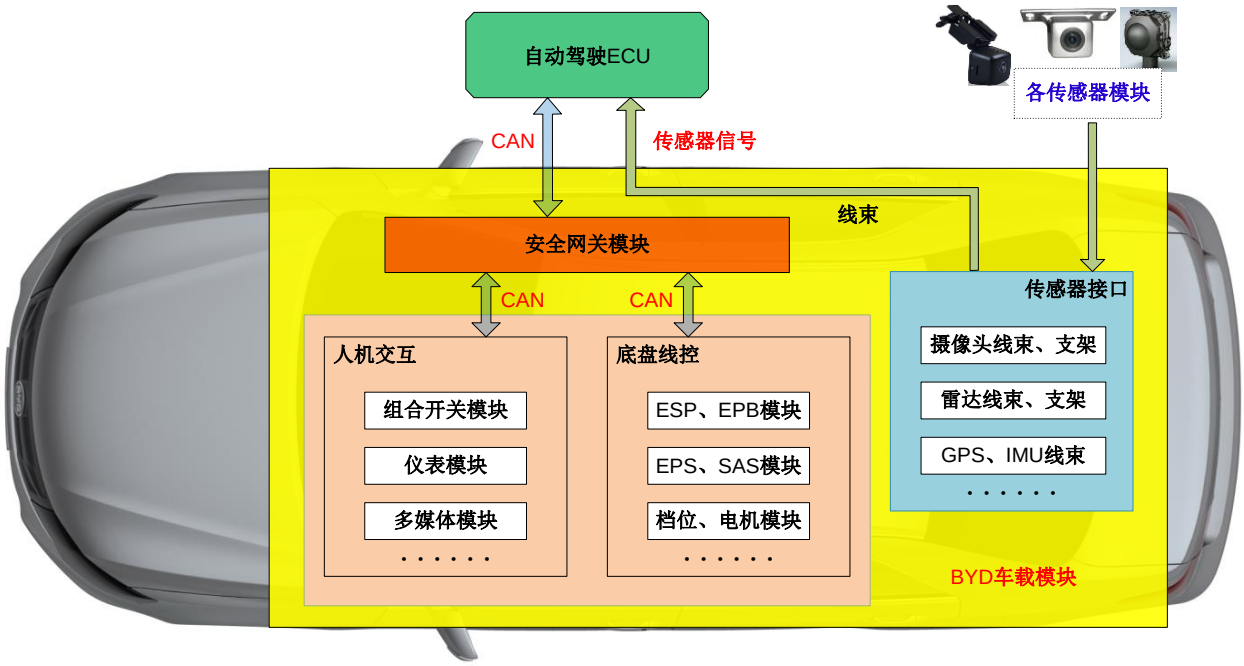
2 开发者平台 API

比亚迪汽车工业有限公司提供的开发者接入平台（包括“安全网关”和“开发者平台 API”）属于比亚迪 DiLink 系统“整车控制层开放”之中的一项重要内容：



如上图，底盘线控、人机交互这 2 项开放内容都需要通过“开发者接入平台”才能使用！

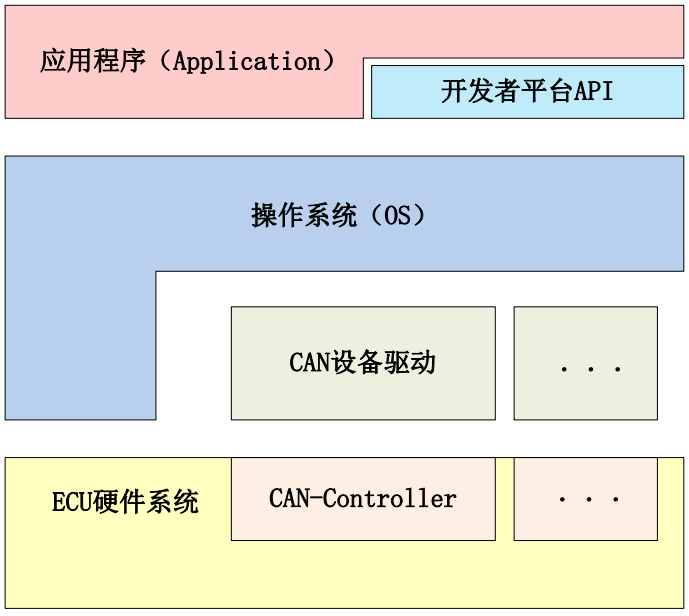
而安全网关是开发者的“自动驾驶 ECU”与比亚迪车进行数据交互的唯一硬件接口：



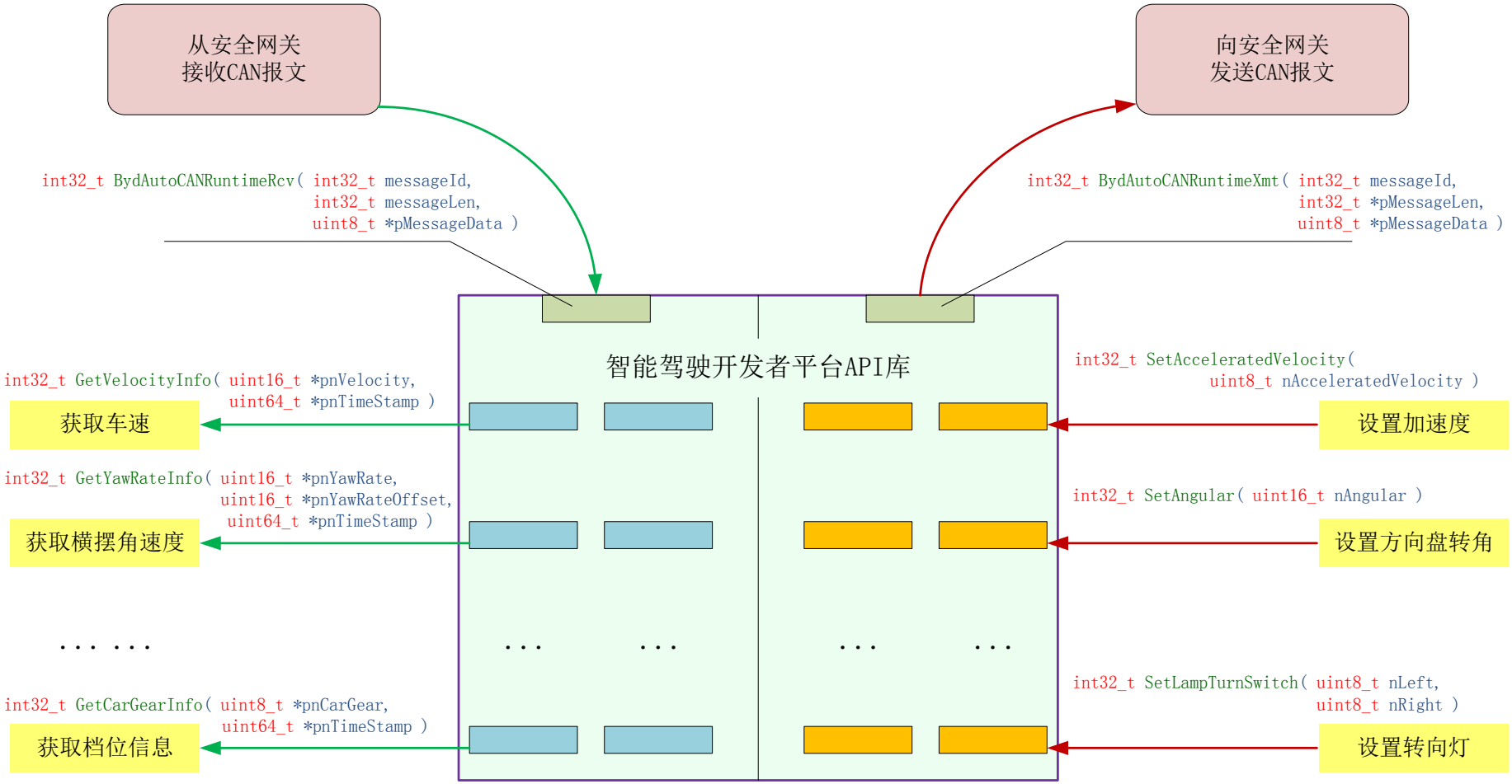
现阶段的安全网关支持的“通讯接口”只有 CAN 接口，后续可能会增加以太网接口（暂不支持）。

比亚迪汽车工业有限公司为了使“自动驾驶领域”的广大开发者无需关注繁琐的 CAN 报文数据的解析和封装打包等操作，向开发者提供 API 库：

广大自动驾驶开发者使用的硬件系统是存在差异性的，具体而言：有的开发者可能使用“工控机”进行自动驾驶的开发，有的开发者则采用嵌入式系统的方案。硬件系统的差异导致 CAN 的设备驱动也是各有差异的，所以比亚迪汽车工业有限公司提供的开发者平台 API 采取了“与 CAN 设备驱动分离”的实现方式：



开发者的软件程序负责对 CAN 设备驱动进行初始化、接收从安全网关传输过来的 CAN 报文以及向安全网关发送相关控制操作的 CAN 报文：



开发者的应用程序从安全网关接收了 CAN 报文以后，通过调用以下函数：

```
int32_t BydAutoCANRuntimeRcv( int32_t messageId,
                              int32_t messageLen,
                              uint8_t *pMessageData )
```

将报文 ID、报文长度、以及报文数据等输入到开发者平台 API 库之中；

当开发者的应用程序需要获取当前的整车相关状态的时候，就可以直接调用 API 库之中的特定功能的接口函数，例如：

- 获取车速 ——

```
int32_t GetVelocityInfo( uint16_t *pnVelocity,
                         uint64_t *pnTimeStamp )
```

- 获取横摆角速度 ——

```
int32_t GetYawRateInfo( uint16_t *pnYawRate,
                        uint16_t *pnYawRateOffset,
                        uint64_t *pnTimeStamp )
```

- 获取档位信息 ——

```
int32_t GetCarGearInfo( uint8_t *pnCarGear,
                        uint64_t *pnTimeStamp )
```

同样，开发者的应用程序可以在任意时刻调用 API 库之中的特定功能的接口函数，用来实现相应的控制操作，例如：

- 设置加速度 ——

```
int32_t SetAcceleratedVelocity( uint8_t nAcceleratedVelocity )
```

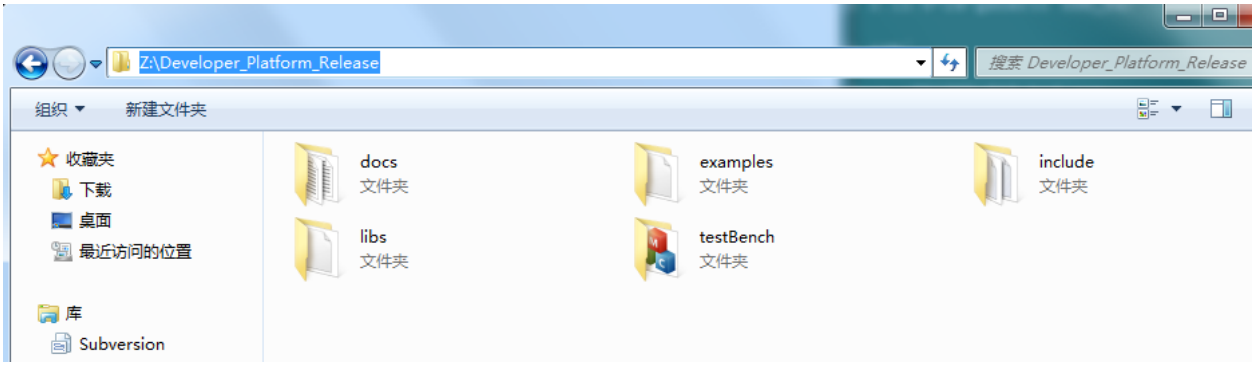
开发者的应用程序需要定时（例如 10ms 时间间隔）向安全网关发送针对各执行机构的 CAN 控制报文。在每次执行 CAN 报文发送操作之前需要调用以下函数 ——

```
int32_t BydAutoCANRuntimeXmt( int32_t messageId,
                              int32_t *pMessageLen,
                              uint8_t *pMessageData )
```

实现的功能：为特定 ID 的发送报文进行报文数据的打包、以及报文长度的确定。

3 API 使用规则

比亚迪汽车工业有限公司给广大开发者提供的 API 库以 SDK 的形式存在：



如上图所示，SDK 开发包中包含了 5 个子目录 ——

1） docs 子目录下放置的是《智能驾驶开发者平台 API 用户手册》，即本文档；

2） examples 子目录下放置的是“开发者平台 API 使用例程”；

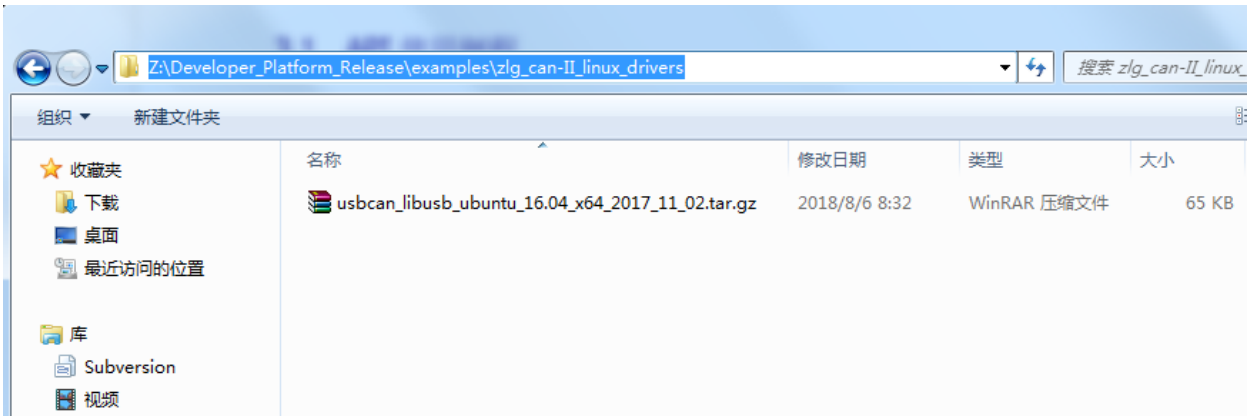
3） include 子目录下放置的是“API 库引用的头文件”；

4） libs 子目录下放置的是“API 库”；

5） testBench 子目录下放置的是“ACU 模拟平台（Windows 版本）”。

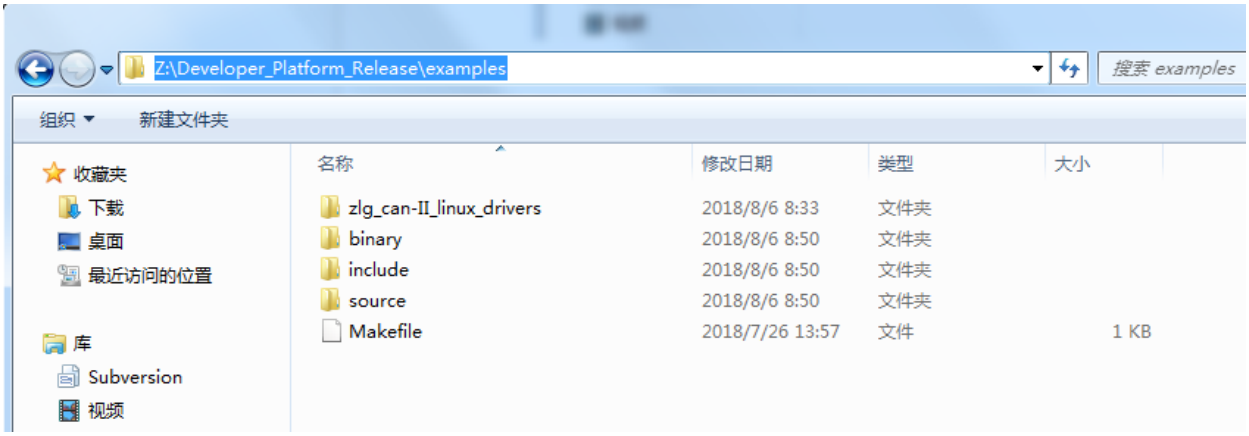
3.1 API 使用例程

第一阶段提供的“API 使用例程”基于 Ubuntu Linux 系统（16.04 版本）开发的，使用 ZLG_CAN-II 板卡测试通过，ZLG_CAN-II 板卡驱动位于“\examples\zlg_can-II_linux_drivers”目录下：

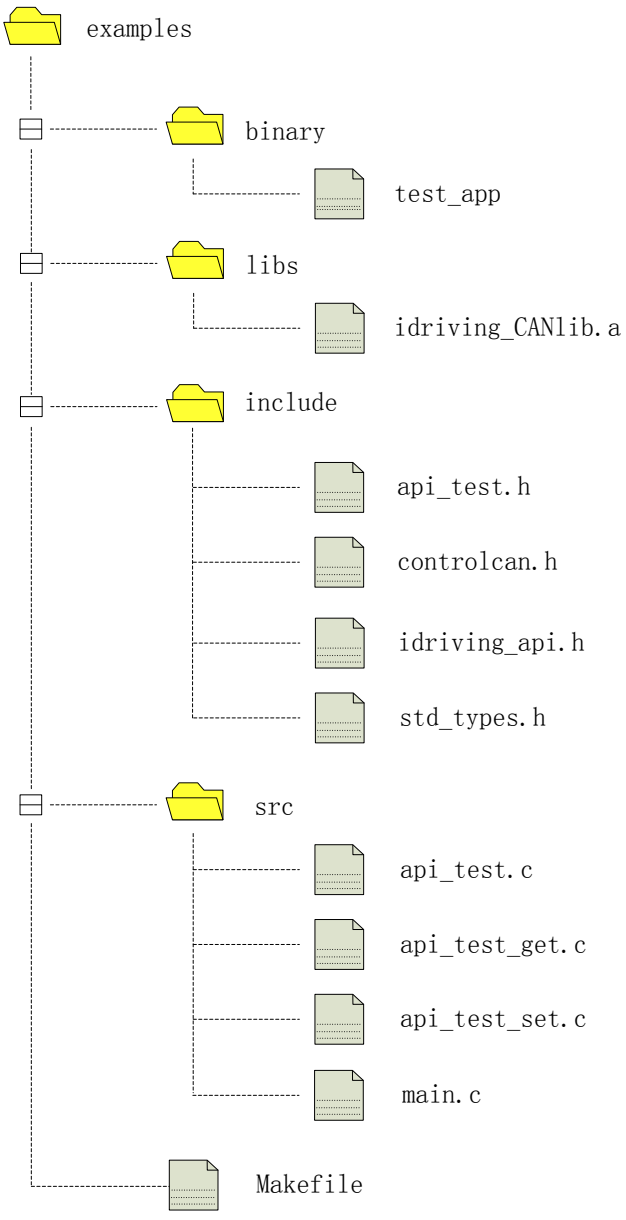


将该压缩包进行解压，将其中的 libusbcan.so 和 libusbcan.so.1 文件复制到 Ubuntu 16.04 系统的/lib 目录下，即可完成 ZLG_CAN-II 板卡驱动的安装。

接下来就可以对“API 使用例程”进行编译、并运行。



如图所示，“API 使用例程”包含 binary、include、source 等几个目录：



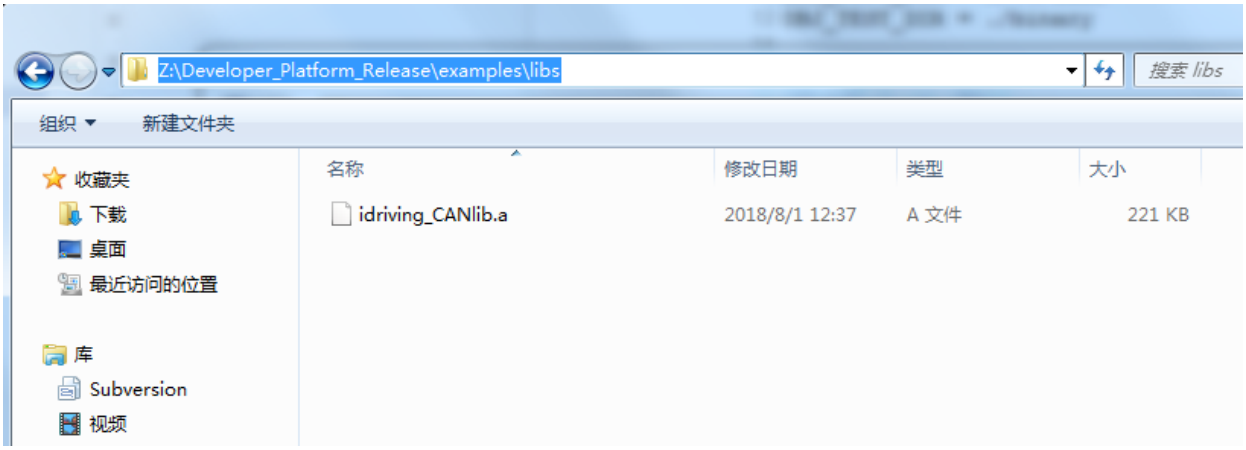
其中，binary 目录下“test_app”为编译生成的可执行文件，可以直接通过终端运行！

3.2 API 库的调用

广大自动驾驶的开发者请参照“API 使用例程”的方式使用 API 库文件：

```
# (c) BYD Auto Ltd.

#########################################################
#                                                       #
# AutoMatic Driving System API Library Build Targets    #
#                                                       #
#########################################################

INC_TEST_DIR = -I./include
INC_TEST_DIR += -I./../include
SRC_TEST_DIR = ./source
LIB_TEST_DIR = ./../libs
OBJ_TEST_DIR = ./binary

CC = gcc
CFLAGS = -g -Wall
RM = rm

TEST_SRCS = $(foreach dir, $(SRC_TEST_DIR), $(wildcard $(dir)/*.c))
TEST_LIBS = $(foreach dir, $(LIB_TEST_DIR), $(wildcard $(dir)/*.a))

APP_NAME = $(OBJ_TEST_DIR)/test_app

$(APP_NAME):$(TEST_SRCS)
    $(CC) $(CFLAGS) $^ $(TEST_LIBS) -o $@ $(INC_TEST_DIR) -lpthread -lusbcan

```

现阶段提供的 API 库文件为 *.a 格式：



API 库文件中包含的接口函数请查看对应的头文件：

“\examples\include\idriving_api.h”

比亚迪汽车工业有限公司建议广大自动驾驶的开发者在其 Ubuntu Linux 软件程序中创建一个线程，用于持续读取来自 CAN 通讯板卡的报文，例如：

```
void* can_rcvThread( void *data )
{
    CAN_DEV_INFO *pCAN_Info = (CAN_DEV_INFO *)data;

    VCI_CAN_OBJ can[RX_BUFF_SIZE]; // buffer
    int cnt, loop;                 // current received

    while (gTestFlag) {
        // msleep(0x01);

        cnt = VCI_Receive(pCAN_Info->devType, pCAN_Info->devIndex,
                          pCAN_Info->channelNum, can, RX_BUFF_SIZE, RX_WAIT_TIME);
        if (!cnt)
            continue;

        for (loop = 0; loop < cnt; loop++) {
            // CAN Receive Message Analysis
            BydAutoCANRuntimeRcv(can[loop]->ID,
                                 can[loop]->DataLen,
                                 can[loop]->Data);
        }
    }
    // printf("can_rcvThread End.\n");
    pthread_exit(0);

    return 0;
}
```

在上述示例代码之中，“VCI_Receive( )”是 ZLG_CAN-II 板卡驱动中提供的接口函数，它用于读取 ZLG_CAN-II 板卡接收的 CAN 报文，该函数的返回值 cnt 表示调用该接口函数读到的 CAN 报文个数。

“BydAutoCANRuntimeRcv ( )”是开发者平台 API 库中提供的接口函数，对于从 CAN 通讯板卡接收的每一个 CAN 报文都需要调用一次该接口函数，函数声明如下：

```
/** ---------------------------------------------------------------
 *  \功能：  CAN报文接收后处理
 *
 *  \参数：  messageId     [输入]          报文ID
 *  \参数：  messageLen    [输入]          报文长度
 *  \参数：  pMessageData  [输入]          报文数据
 *
 *  \返回值：  设置成功后返回 ERR_BYD_AUTO_OK，否则返回相应的错误状态
 */
int32_t BydAutoCANRuntimeRcv( uint32_t messageId,
                              uint32_t messageLen,
                              uint8_t *pMessageData );
```

比亚迪汽车工业有限公司建议广大自动驾驶的开发者在其 Ubuntu Linux 软件程序中还创建一个专门用于向 CAN 通讯板卡发送 CAN 报文的线程（或者定时服务），例如：

```
void* can_xmtThread( void *data )
{
    CAN_DEV_INFO *pCAN_Info = (CAN_DEV_INFO *)data;
    VCI_CAN_OBJ canObj;
    PVCI_CAN_OBJ pCAN_Obj = &canObj;
    uint32_t MessageId = BYD_AUTO_IDRIVING_CMD_0;

    while (gTestFlag) {
        msleep(10);

        // Developer API test
        BydAutoCANRuntimeXmt(MessageId,
                             (uint32_t *)&pCAN_Obj->DataLen,
                             pCAN_Obj->Data);

        /* Transmit the can message */
        VCI_Transmit(pCAN_Info->devType, pCAN_Info->devIndex,
                     pCAN_Info->channelNum, pCAN_Obj, 1);

    }
    // printf("can_xmtThread End.\n");
    pthread_exit(0);

    return 0;
}
```

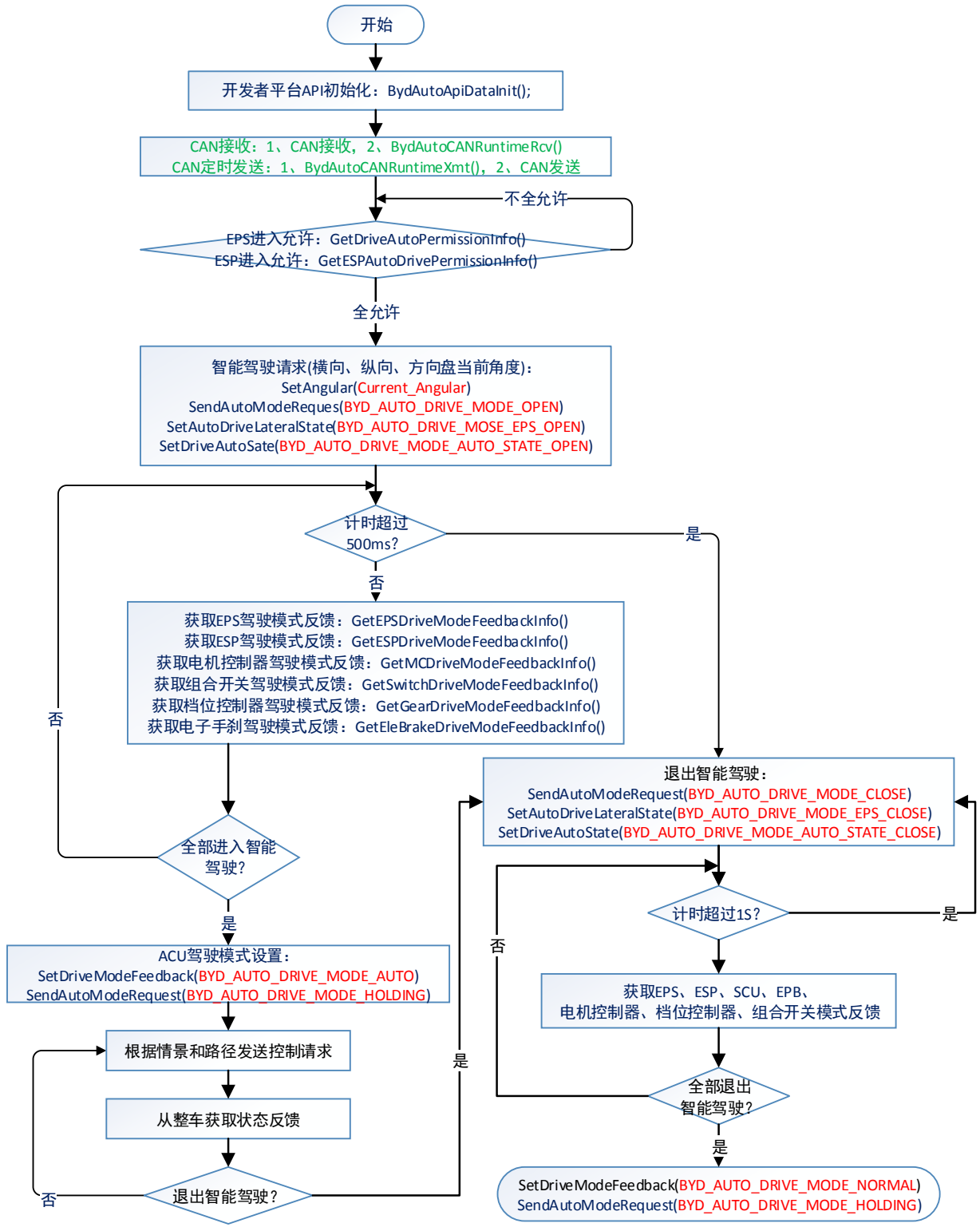
在上述示例代码之中，“VCI_Transmit ( )”是 ZLG_CAN-II 板卡驱动中提供的接口函数，它用于通过该板卡对外发送 CAN 报文，函数的最后一个参数为 1 表示发送每次发送 1 帧 CAN 报文。

“BydAutoCANRuntimeXmt ( )”是开发者平台 API 库中提供的接口函数，它用于打包每次定时（例如每隔 10ms）发送的 CAN 报文数据，函数声明如下：

```
/** ---------------------------------------------------------------
 *  \功能：  CAN报文发送前处理
 *
 *  \参数：  messageId     [输入]          报文ID
 *  \参数：  pMessageLen   [输入 & 输出]   报文长度
 *  \参数：  pMessageData  [输入 & 输出]   报文数据
 *
 *  \返回值：  设置成功后返回 ERR_BYD_AUTO_OK，否则返回相应的错误状态
 */
int32_t BydAutoCANRuntimeXmt( uint32_t messageId,
                              uint32_t *pMessageLen,
                              uint8_t  *pMessageData );
```

3.3 API 调用逻辑说明

广大自动驾驶的开发者在使用 API 进行应用开发的时候，需要注意：某些 API 的调用顺序是与整车上相关执行机构的工作逻辑相关联的。具体描述如下：



4 API 接口说明

比亚迪汽车工业有限公司为广大自动驾驶开发者提供的 API 库中包含了以下 API 接口：

API 库的初始化		
序号	功能描述	接口函数（API）
1	API 初始化函数	int32_t BydAutoApiDataInit( );

API 库中循环调用的接口		
序号	功能描述	接口函数（API）
1	CAN 接收报文实时处理	int32_t BydAutoCANRuntimeRcv(uint32_t messageId, uint32_t messageLen, uint8_t *pMessageData);
2	CAN 发送报文实时生成	int32_t BydAutoCANRuntimeXmt(uint32_t messageId, uint32_t *pMessageLen, uint8_t *pMessageData);

API 库中设置类接口		
序号	功能描述	接口函数（API）
1	向各功能模块发送驾驶模式请求	int32_t SendAutoDriveModeRequest(uint8_t nModeRequest);
2	设置灯光 AUTO 档开关	int32_t SetLampAutoSwitch(uint8_t nAutoSwitch);
3	转向灯设置	int32_t SetLampTurnSwitch(uint8_t nLeft, uint8_t nRight);
4	制动灯设置	int32_t SetLampBrakeSwitch(uint8_t nBrakeSwitch);
5	目标方向盘角度设置	int32_t SetAngular(uint16_t nAngular);

6	目标 EPB 状态设置	int32_t SetEpbState(uint8_t nEpbState);
7	目标雨刮档位设置	int32_t SetRainWiper(uint8_t nRainWiper);
8	目标档位设置	int32_t SetCarGear(uint8_t nCarGear);
9	驾驶模式反馈设置	int32_t SetDriveModeFeedback(uint8_t nDriveModeFeedback);
10	目标加速度设置	int32_t SetAcceleratedVelocity(uint8_t nAcceleratedVelocity);
11	智能驾驶纵向控制状态设置	int32_t SetDriveAutoState(uint8_t nDriveAutoState);
12	智能驾驶 API 滚动计数及控制发送时序	int32_t SendAliveCount(uint8_t nCount);
13	紧急告警灯设置	int32_t SetLampEmergencyWarningSwitch(uint8_t nSwitch);
14	外围灯光信号设置	int32_t SetLampOutSideSwitch(uint8_t nArea, uint8_t nSwitch);
15	喇叭设置	int32_t SetHornSwitch(uint8_t nSwitch);
16	多媒体显示设置	int32_t SetMultimediaDisplaySwitch(uint8_t nSwitch);

API 库中获取类接口		
序号	功能描述	接口函数（API）
1	获取车速信息	int32_t GetVelocityInfo(uint16_t *pnVelocity, uint64_t *pnTimeStamp);
2	获取车轮实时轮速信息	int32_t GetWheelSpeedInfo(uint16_t *pnFL, uint16_t *pnFR, uint16_t *pnRL, uint16_t *pnRR, uint64_t *pnTimeStamp);
3	获取车辆灯光信息	int32_t GetLampInfo(uint8_t *pnSmall, uint8_t *pnNear, uint8_t *pnFar, uint64_t *pnTimeStamp);

4	获取车辆转向灯信息	int32_t GetLampTurnInfo(uint8_t *pnLeft, uint8_t *pnRight, uint64_t *pnTimeStamp);
5	获取车辆雾灯信息	int32_t GetLampFogInfo(uint8_t *pnFront, uint8_t *pnRear, uint64_t *pnTimeStamp);
6	获取雨刮档位信息	int32_t GetRainWiperInfo(uint8_t *pnRainWiper, uint64_t *pnTimeStamp);
7	获取组合开关智能驾驶模式反馈信息	int32_t GetSwitchDriveModeFeedbackInfo(uint8_t *pnSwitchDriveModeFeedback, uint64_t *pnTimeStamp);
8	获取灯光 AUTO 档信息	int32_t GetLampAutoInfo(uint8_t *pnLampAuto, uint64_t *pnTimeStamp);
9	获取车门状态信息	int32_t GetDoorInfo(uint8_t *pnFL, uint8_t *pnFR, uint8_t *pnRL, uint8_t *pnRR, uint8_t *pnTrunk, uint64_t *pnTimeStamp);
10	获取车轮行驶方向信息	int32_t GetWheelDriveDirectionInfo(uint8_t *pnFL, uint8_t *pnFR, uint8_t *pnRL, uint8_t *pnRR, uint64_t *pnTimeStamp);
11	获取 ESP 当前驾驶模式信息	int32_t GetESPDriveModeFeedbackInfo(uint8_t *pnESPDriveModeFeedback, uint64_t *pnTimeStamp);
12	获取 ESP 进入智能驾驶允许信息	int32_t GetESPAutoDrivePermissionInfo (uint8_t *pnAutoDrivePermission, uint64_t *pnTimeStamp);
13	获取 ESP 滚动计数信息	int32_t GetESPAliveInfo(uint8_t *pnAlive, uint64_t *pnTimeStamp);

14	获取 ESP 模块故障信息 （尚未实现）	int32_t GetESPErrInfo(uint8_t *pnTractionControl, uint8_t *pnBodyDynamic, uint64_t *pnTimeStamp);
15	获取横摆角速度信息	int32_t GetYawRateInfo(uint16_t *pnYawRate, uint16_t *pnYawRateOffset, uint64_t *pnTimeStamp);
16	获取车辆纵向加速度	int32_t GetAccelerationXInfo(uint16_t *pnAX, uint16_t *pnAXOffset, uint64_t *pnTimeStamp);
17	获取车辆横向加速度	int32_t GetAccelerationYInfo(uint16_t *pnAY, uint16_t *pnAYOffset, uint64_t *pnTimeStamp);
18	获取轮速脉冲信息	int32_t GetWheelPulseCounterInfo(uint16_t *pnFL, uint16_t *pnFR, uint16_t *pnRL, uint16_t *pnRR, uint64_t *pnTimeStamp);
19	获取油门深度信息	int32_t GetAccelerateDeepnessInfo(uint8_t *pnAccelerateDeepness, uint64_t *pnTimeStamp);
20	获取制动深度信息	int32_t GetBrakeDeepnessInfo(uint8_t *pnBrakeDeepness, uint64_t *pnTimeStamp);
21	获取制动踏板状态信息	int32_t GetBrakePedalSignalInfo(uint8_t *pnBrakePedalSignal, uint64_t *pnTimeStamp);
22	获取档位信息	int32_t GetCarGearInfo(uint8_t *pnCarGear, uint64_t *pnTimeStamp);
23	获取档位驾驶模式反馈	int32_t GetGearDriveModeFeedbackInfo(uint8_t* pnGearDriveModeFeedback, uint64_t* pnTimeStamp);

24	获取电子手刹状态信息	int32_t GetEleBrakeInfo(uint8_t *pnEleBrake, uint64_t *pnTimeStamp);
25	获取电子手刹驾驶模式反馈信息	int32_t GetEleBrakeDriveModeFeedbackInfo(uint8_t* pnEleBrakeDriveModeFeedback, uint64_t* pnTimeStamp);
26	获取方向盘角度信息	int32_t GetAngularInfo(uint16_t *pnAngular, uint64_t *pnTimeStamp);
27	获取 EPS 当前驾驶模式信息	int32_t GetEPSDriveModeFeedbackInfo(uint8_t *pnEPSDriveModeFeedback, uint64_t *pnTimeStamp);
28	获取 EPS 进入智能驾驶允许信息	int32_t GetDriveAutoPermissionInfo(uint8_t *pnDriveAutoPermission, uint64_t *pnTimeStamp);
29	获取方向盘旋转速度信息	int32_t GetRotationSpeedInfo(uint8_t *pnRotationSpeed, uint64_t *pnTimeStamp);
30	获取电机扭矩信息	int32_t GetMotorTorqueInfo(uint16_t *pnMotorTorque, uint64_t *pnTimeStamp);
31	获取电机控制器驾驶模式反馈信息	int32_t GetMCDriveModeFeedbackInfo(uint8_t *pnMCDriveModeFeedback, uint64_t *pnTimeStamp);
32	获取主缸压力值信息 （尚未实现）	int32_t GetMCPressureInfo(uint16_t *pnPressure, uint8_t *pnPressureState, uint16_t *pnPressureOffset, uint8_t *pnPressureOffsetState, uint64_t *pnTimeStamp);
33	获取智能驾驶开关信息	int32_t GetAutoDriveKeyInfo(uint8_t *pnAutoDriveKey, uint64_t *pnTimeStamp);
34	获取智能驾驶紧急退出按键信息	int32_t GetEmergencyExitKeyInfo(uint8_t *pnEmergencyExitKey, uint64_t *pnTimeStamp);

4.1 初始化接口

4.1.1 API 初始化函数

API 名称	int32_t BydAutoApiDataInit( );		
接口描述	开发者平台 API 库的初始化		
参数说明	输入参数	void（空）	
	输出参数	void（空）	
返回值	ERR_BYD_AUTO_OK （0）		初始化成功
	ERR_BYD_AUTO_MEM_MALLOC （-2）		内存申请错误

4.2 循环调用接口

4.2.1 CAN 接收报文实时处理

API 名称	int32_t BydAutoCANRuntimeRcv(uint32_t messageId, uint32_t messageLen, uint8_t *pMessageData);		
接口描述	每次接收到 1 个 CAN 报文，都需要调用 1 次该函数		
参数说明	输入参数	messageId（报文 ID 号）	
		messageLen（报文数据长度）	
		pMessageData（指向 CAN 报文数据的指针）	
	输出参数	无	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，不属于来自安全网关的报文 ID，或者报文长度不符合要求）

4.2.2 CAN 发送报文实时生成

API 名称	int32_t BydAutoCANRuntimeXmt(uint32_t messageId, uint32_t *pMessageLen, uint8_t *pMessageData);		
接口描述	每次发送自动驾驶相关控制报文之前，都要调用此函数		
参数说明	输入参数	messageId（报文 ID 号）	
		pMessageLen（报文数据长度）	
		pMessageData（指向 CAN 报文数据的指针）	
	输出参数	pMessageLen（报文数据长度）	
		pMessageData（指向 CAN 报文数据）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，不属于来自安全网关的报文 ID）

4.3 智能驾驶设置类接口

4.3.1 向各功能模块发送驾驶模式请求

API 名称	int32_t SendAutoDriveModeRequest(uint8_t nModeRequest);		
接口描述	向各功能模块发送驾驶模式请求		
参数说明	输入参数	nModeRequest（模式请求）：	
		BYD_AUTO_DRIVE_MODE_HOLDING (0) BYD_AUTO_DRIVE_MODE_OPEN (1) BYD_AUTO_DRIVE_MODE_CLOSE (2)	保持 开启 关闭
	输出参数	无	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，无法识别的模式请求）

4.3.2 设置灯光 AUTO 档开关

API 名称	int32_t SetLampAutoSwitch(uint8_t nAutoSwitch);		
接口描述	设置灯光 AUTO 档开关		
参数说明	输入参数	nAutoSwitch（AUTO 档开关）：	
		BYD_AUTO_LAMP_AUTO_INVALID (0) BYD_AUTO_LAMP_AUTO_ON (1) BYD_AUTO_LAMP_AUTO_OFF (2)	无效 打开 关闭
	输出参数	无	
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入参数不属于上述参数范围）	

4.3.3 转向灯设置

API 名称	int32_t SetLampTurnSwitch(uint8_t nLeft, uint8_t nRight);		
接口描述	设置转向灯开关		
参数说明	输入参数	nLeft（左转向灯开关）：	
		BYD_AUTO_LAMP_ON (1) BYD_AUTO_LAMP_OFF (0)	打开 关闭
		nRight（右转向灯开关）：	
		BYD_AUTO_LAMP_ON (1) BYD_AUTO_LAMP_OFF (0)	打开 关闭
	输出参数	无	
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入参数不属于上述参数范围）	

4.3.4 制动灯设置

API 名称	int32_t SetLampBrakeSwitch(uint8_t nBrakeSwitch);		
接口描述	设置制动灯开关		
参数说明	输入参数	nBrakeSwitch（制动灯开关）：	
		BYD_AUTO_LAMP_ON (1) BYD_AUTO_LAMP_OFF (0)	打开 关闭
	输出参数	无	
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入参数不属于上述参数范围）	

4.3.5 目标方向盘角度设置

API 名称	int32_t SetAngular(uint16_t nAngular);	
接口描述	设置目标方向盘角度	
参数说明	输入参数	nAngular 与“目标方向盘角度”之间的计算公式： nAngular = (780 + 目标方向盘角度) ×10； 其中，目标方向盘角度可设置范围：[-500.0, +500.0]度 对应 nAngular 的数值范围：[0x0AF0, 0x3200]
	输出参数	无
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入参数不属于上述参数范围）

4.3.6 目标 EPB 状态设置

API 名称	int32_t SetEpbState(uint8_t nEpbState);		
接口描述	设置目标 EPB 状态		
参数说明	输入参数	nEpbState（目标 EPB 状态）：	
		BYD_AUTO_EPB_SET_INVALID （0） BYD_AUTO_EPB_SET_APPLYING （1） BYD_AUTO_EPB_SET_RELEASED （2）	EPB 无效 EPB 拉起 EPB 释放
	输出参数	无	
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入参数不属于上述参数范围）	

4.3.7 目标雨刮档位设置

API 名称	int32_t SetRainWiper(uint8_t nRainWiper);		
接口描述	设置目标雨刮档位		
参数说明	输入参数	nRainWiper（目标雨刮档位）：	
		BYD_AUTO_RAIN_WIPER_OFF （1） BYD_AUTO_RAIN_WIPER_SHORT_PRESS （2） BYD_AUTO_RAIN_WIPER_LONG_PRESS （3） BYD_AUTO_RAIN_WIPER_INTERVAL_1 （4） BYD_AUTO_RAIN_WIPER_INTERVAL_2 （5） BYD_AUTO_RAIN_WIPER_INTERVAL_3 （6） BYD_AUTO_RAIN_WIPER_INTERVAL_4 （7） BYD_AUTO_RAIN_WIPER_SLOW （8） BYD_AUTO_RAIN_WIPER_QUICK （9）	OFF 档 点刮短按 点刮长按 间隙 1 档 间隙 2 档 间隙 3 档 间隙 4 档 慢刮 快刮
	输出参数	无	
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入参数不属于上述参数范围）	

4.3.8 目标档位设置

API 名称	int32_t SetCarGear(uint8_t nCarGear);		
接口描述	设置目标档位		
参数说明	输入参数	nCarGear（目标档位）：	
		BYD_AUTO_CAR_GEAR_INVALID （0） BYD_AUTO_CAR_GEAR_P （1） BYD_AUTO_CAR_GEAR_R （2） BYD_AUTO_CAR_GEAR_N （3） BYD_AUTO_CAR_GEAR_D （4）	无效 P 档 R 档 N 档 D 档
	输出参数	无	
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入参数不属于上述参数范围）	

4.3.9 驾驶模式反馈设置

API 名称	int32_t SetDriveModeFeedback(uint8_t nDriveModeFeedback);		
接口描述	设置驾驶模式反馈（自动驾驶 ECU 判断所有执行机构是否都进入了自动驾驶模式，由自动驾驶 ECU 对外广播）		
参数说明	输入参数	nDriveModeFeedback（驾驶模式反馈）：	
		BYD_AUTO_DRIVE_MODE_AUTO_ABNORMAL_1 （0） BYD_AUTO_DRIVE_MODE_AUTO （1） BYD_AUTO_DRIVE_MODE_NORMAL （2） BYD_AUTO_DRIVE_MODE_AUTO_ABNORMAL_2 （3）	异常 1 自动驾驶模式 正常驾驶模式 异常 2（故障）
	输出参数	无	
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入参数不属于上述参数范围）	

4.3.10 目标加速度设置

API 名称	int32_t SetAcceleratedVelocity(uint8_t nAcceleratedVelocity);		
接口描述	设置目标加速度		
参数说明	输入参数	nAcceleratedVelocity 与“目标加速度”之间的计算公式： nAcceleratedVelocity = （目标加速度 + 5）× 20； 其中，目标加速度可设置范围：[-5.00，+5.00] m/s^2 对应 nAcceleratedVelocity 的数值范围：[0x0，0xC8]	
	输出参数	无	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入参数不属于上述参数范围）

4.3.11 智能驾驶纵向控制状态设置

API 名称	int32_t SetDriveAutoState(uint8_t nDriveAutoState);		
接口描述	设置纵向控制的状态（向 ESP 系统发送请求）		
参数说明	输入参数	nDriveAutoState（纵向控制状态）：	
		BYD_AUTO_DRIVE_MODE_AUTO_STATE_CLOSE （0） BYD_AUTO_DRIVE_MODE_AUTO_STATE_OPEN （3）	纵向控制关闭 纵向控制开启
	输出参数	无	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入参数不属于上述参数范围）

4.3.12 智能驾驶 API 滚动计数及控制发送时序

API 名称	int32_t SendAliveCount(uint8_t nCount);		
接口描述	设置循环滚动计数器		
参数说明	输入参数	nCount（滚动计数器）：	
		该计数器为循环滚动计数器，它需要定时（20ms）增长，初始值为 0，当数值增加到 15 以后下次重新从 0 开始增长。	
	输出参数	无	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入参数不属于上述参数范围）

4.3.13 紧急告警灯设置

API 名称	int32_t SetLampEmergencyWarningSwitch(uint8_t nSwitch);		
接口描述	设置紧急告警灯开关		
参数说明	输入参数	nSwitch（紧急告警灯开关）：	
		BYD_AUTO_LAMP_ON （1） BYD_AUTO_LAMP_OFF （0）	打开 关闭
	输出参数	无	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入参数不属于上述参数范围）

4.3.14 外围灯光信号设置

API 名称	int32_t SetLampOutSideSwitch(uint8_t nArea, uint8_t nSwitch);			
接口描述	设置外围灯光信号的开关			
参数说明	输入参数	nArea（灯光位置 / 类型）:		
		BYD_AUTO_LAMP_SMALL （0） BYD_AUTO_LAMP_LOW （1） BYD_AUTO_LAMP_HIGH （2） BYD_AUTO_LAMP_FRONT_FOG （3） BYD_AUTO_LAMP_REAR_FOG （4）	小灯 近光灯 远光灯 前雾灯 后雾灯
		nSwitch（灯光信号的开关）		
		BYD_AUTO_LAMP_ON （1） BYD_AUTO_LAMP_OFF （0）	打开 关闭
	输出参数	无		
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入参数不属于上述参数范围）	

4.3.15 喇叭设置

API 名称	int32_t SetHornSwitch(uint8_t nSwitch);			
接口描述	设置喇叭信号的开关			
参数说明	输入参数	nSwitch（喇叭信号的开关）:		
		BYD_AUTO_HORN_ON （1） BYD_AUTO_HORN_OFF （0）	开启 关闭	
	输出参数	无		
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入参数不属于上述参数范围）	

4.3.16 多媒体显示设置

API 名称	int32_t SetMultimediaDisplaySwitch(uint8_t nSwitch);		
接口描述	设置多媒体显示的开关		
参数说明	输入参数	nSwitch（多媒体显示的开关）：	
		BYD_AUTO_MULTIMEDIA_DISPLAY_ON （1） BYD_AUTO_MULTIMEDIA_DISPLAY_OFF （2）	开启 关闭
	输出参数	无	
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入参数不属于上述参数范围）	

4.3.17 智能驾驶横向控制状态设置

API 名称	int32_t SetAutoDriveLateralState(uint8_t nAutoDriveLateralState);		
接口描述	设置智能驾驶横向控制状态		
参数说明	输入参数	nAutoDriveLateralState（智能驾驶横向控制状态）：	
		BYD_AUTO_DRIVE_MODE_EPS_CLOSE （0） BYD_AUTO_DRIVE_MOSE_EPS_OPEN （1）	关闭 开启
	输出参数	无	
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入参数不属于上述参数范围）	

4.4 智能驾驶获取类接口

4.4.1 获取车速信息

API 名称	int32_t GetVelocityInfo(uint16_t *pnVelocity, uint64_t *pnTimeStamp);		
接口描述	获取车辆实时的速度信息		
参数说明	输入参数	pnVelocity（指向车速信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	pnVelocity 指向的数据与实际车速的计算公式： 实际车速 = (pnVelocity 指向的数据 ×0.06875) km / h 其中，pnVelocity 指向的数值范围：[0x0，0x0FFE] 对应实际车速的数值范围：[0，281.4625] km / h	
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.2 获取车轮实时轮速信息.

API 名称	int32_t GetWheelSpeedInfo(uint16_t *pnFL, uint16_t *pnFR, uint16_t *pnRL, uint16_t *pnRR, uint64_t *pnTimeStamp);	
接口描述	获取实时的车轮轮速信息	
参数说明	输入参数	pnFL（指向左前轮速的指针）;
		pnFR（指向右前轮速的指针）;
		pnRL（指向左后轮速的指针）;
		pnRR（指向右后轮速的指针）;
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）

	输出参数	pnFL、pnFR、pnRL、pnRR 指向的数据与实际轮速的计算公式： 实际轮速 = （指针指向的数据 ×0.06875）km / h 其中，指针指向的数值范围：[0x0，0x0FFE] 对应实际轮速的数值范围：[0，281.4625] km / h	
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.3 获取车辆灯光信息

API 名称	int32_t GetLampInfo(uint8_t *pnSmall, uint8_t *pnNear, uint8_t *pnFar, uint64_t *pnTimeStamp);		
接口描述	获取车辆灯光信息		
参数说明	输入参数	pnSmall（指向小灯状态信息的指针）;	
		pnNear（指向近光灯状态信息的指针）;	
		pnFar（指向远光灯状态信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	小灯、近光灯、远光灯状态：	
		BYD_AUTO_LAMP_OFF （0）	关闭
		BYD_AUTO_LAMP_ON （1）	打开
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.4 获取车辆转向灯信息

API 名称	int32_t GetLampTurnInfo(uint8_t *pnLeft, uint8_t *pnRight, uint64_t *pnTimeStamp);		
接口描述	获取车辆转向灯信息		
参数说明	输入参数	pnLeft（指向左转向灯状态信息的指针）;	
		pnRight（指向右转向灯状态信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	左转向灯、右转向灯状态：	
		BYD_AUTO_LAMP_OFF （0）	关闭
		BYD_AUTO_LAMP_ON （1）	打开
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入指针为空）	
	ERR_BYD_AUTO_READ （-6）	函数调用失败，读取数据失败	

4.4.5 获取车辆雾灯信息

API 名称	int32_t GetLampFogInfo(uint8_t *pnFront, uint8_t *pnRear, uint64_t *pnTimeStamp);		
接口描述	获取车辆雾灯信息		
参数说明	输入参数	pnFront（指向前雾灯状态信息的指针）;	
		pnRear（指向后雾灯状态信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	前雾灯、后雾灯状态：	
		BYD_AUTO_LAMP_OFF （0）	关闭
		BYD_AUTO_LAMP_ON （1）	打开
		pnTimeStamp（指向读取到的时间计数标记）	

返回值	ERR_BYD_AUTO_OK （0）	函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）	函数调用失败，读取数据失败

4.4.6 获取雨刮档位信息

API 名称	int32_t GetRainWiperInfo(uint8_t *pnRainWiper, uint64_t *pnTimeStamp);		
接口描述	获取雨刮档位信息		
参数说明	输入参数	pnRainWiper（指向雨刮档位信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	雨刮档位状态：	
		BYD_AUTO_RAIN_WIPER_INVALID （0）	无效
		BYD_AUTO_RAIN_WIPER_OFF （1）	OFF 档
		BYD_AUTO_RAIN_WIPER_SHORT_PRESS （2）	点刮短按
		BYD_AUTO_RAIN_WIPER_LONG_PRESS （3）	点刮长按
		BYD_AUTO_RAIN_WIPER_INTERVAL_1 （4）	间隙 1 档
		BYD_AUTO_RAIN_WIPER_INTERVAL_2 （5）	间隙 2 档
		BYD_AUTO_RAIN_WIPER_INTERVAL_3 （6）	间隙 3 档
		BYD_AUTO_RAIN_WIPER_INTERVAL_4 （7）	间隙 4 档
		BYD_AUTO_RAIN_WIPER_SLOW （8）	慢刮
		BYD_AUTO_RAIN_WIPER_QUICK （9）	块刮
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入指针为空）	
	ERR_BYD_AUTO_READ （-6）	函数调用失败，读取数据失败	

4.4.7 获取组合开关智能驾驶模式反馈信息

API 名称	int32_t GetSwitchDriveModeFeedbackInfo(uint8_t *pnSwitchDriveModeFeedback, uint64_t *pnTimeStamp);		
接口描述	获取组合开关智能驾驶模式反馈信息		
参数说明	输入参数	pnSwitchDriveModeFeedback（指向驾驶模式反馈信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	驾驶模式反馈信息：	
		BYD_AUTO_DRIVE_MODE_SWITCH_FEEDBACK_INVALID （0）	无效
		BYD_AUTO_DRIVE_MODE_SWITCH_FEEDBACK_NORMAL （1）	正常驾驶
		BYD_AUTO_DRIVE_MODE_SWITCH_FEEDBACK_AUTO （2）	自动驾驶
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数 （例如，输入指针为空）	
	ERR_BYD_AUTO_READ （-6）	函数调用失败，读取数据失败	

4.4.8 获取灯光 AUTO 档信息

API 名称	int32_t GetLampAutoInfo(uint8_t *pnLampAuto, uint64_t *pnTimeStamp);		
接口描述	获取灯光 AUTO 档信息		
参数说明	输入参数	pnLampAuto（指向灯光 AUTO 档状态信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	灯光 AUTO 档状态：	
		BYD_AUTO_LAMP_AUTO_INVALID （0）	无效
		BYD_AUTO_LAMP_AUTO_ON （1）	AUTO 档
		BYD_AUTO_LAMP_AUTO_OFF （2）	非 AUTO 档
		pnTimeStamp（指向读取到的时间计数标记）	

返回值	ERR_BYD_AUTO_OK （0）	函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）	函数调用失败，读取数据失败

4.4.9 获取车门状态信息

API 名称	int32_t GetDoorInfo(uint8_t *pnFL, uint8_t *pnFR, uint8_t *pnRL, uint8_t *pnRR, uint8_t *pnTrunk, uint64_t *pnTimeStamp);		
接口描述	获取车门状态信息		
参数说明	输入参数	pnFL（指向左前车门状态信息的指针）;	
		pnFR（指向右前车门状态信息的指针）;	
		pnRL（指向左后车门状态信息的指针）;	
		pnRR（指向右后车门状态信息的指针）;	
		pnTrunk（指向行李箱状态信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	左前、右前、左后、右后车门状态：	
		BYD_AUTO_DOOR_OFF （0）	关闭
		BYD_AUTO_DOOR_ON （1）	打开
		行李箱状态：	
		BYD_AUTO_TRUNK_OFF （0）	关闭
		BYD_AUTO_TRUNK_ON （1）	打开
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.10 获取车轮行驶方向信息

API 名称	int32_t GetWheelDriveDirectionInfo(uint8_t *pnFL, uint8_t *pnFR, uint8_t *pnRL, uint8_t *pnRR, uint64_t *pnTimeStamp);		
接口描述	获取车轮行驶方向信息		
参数说明	输入参数	pnFL（指向左前车轮行驶方向信息的指针）;	
		pnFR（指向右前车轮行驶方向信息的指针）;	
		pnRL（指向左后车轮行驶方向信息的指针）;	
		pnRR（指向右后车轮行驶方向信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	左前、右前、左后、右后车轮行驶方向：	
		BYD_AUTO_DRIVE_DIRECTION_UNDEFINE （0）	无效
		BYD_AUTO_DRIVE_DIRECTION_FORWARD （1）	向前
		BYD_AUTO_DRIVE_DIRECTION_BACKWARD （2）	向后
		BYD_AUTO_DRIVE_DIRECTION_STOP （3）	停止
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入指针为空）	
	ERR_BYD_AUTO_READ （-6）	函数调用失败，读取数据失败	

4.4.11 获取 ESP 当前驾驶模式信息

API 名称	int32_t GetESPDriveModeFeedbackInfo(uint8_t *pnESPDriveModeFeedback, uint64_t *pnTimeStamp);	
接口描述	获取 ESP 当前驾驶模式信息	
参数说明	输入参数	pnESPDriveModeFeedback（指向 ESP 驾驶模式信息的指针）;
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）

	输出参数	ESP 当前驾驶模式信息：	
		BYD_AUTO_DRIVE_MODE_ESP_FEEDBACK_CLOSE （0）	正常驾驶
		BYD_AUTO_DRIVE_MODE_ESP_FEEDBACK_OPEN （1）	自动驾驶
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.12 获取 ESP 进入智能驾驶允许信息

API 名称	int32_t GetESPAutoDrivePermissionInfo (uint8_t *pnAutoDrivePermission, uint64_t *pnTimeStamp);		
接口描述	获取 ESP 控制激活状态信息		
参数说明	输入参数	pnAutoDrivePermission（指向 ESP 进入允许信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	ESP 进入智能驾驶允许信息状态：	
		BYD_AUTO_DRIVE_ESP_NOT_PERMISSION （0）	ESP 不允许进入
		BYD_AUTO_DRIVE_ESP_PERMISSION_OK （1）	ESP 允许进入
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.13 获取 ESP 滚动计数信息

API 名称	int32_t GetESPAliveInfo(uint8_t *pnAlive, uint64_t *pnTimeStamp);		
接口描述	获取 ESP 滚动计数信息		
参数说明	输入参数	pnAlive（指向 ESP 滚动计数信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	ESP 滚动计数信息： 数值范围 [0，15]	
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.14 获取 ESP 模块故障信息（尚未实现）

API 名称	int32_t GetESPErrInfo(uint8_t *pnTractionControl, uint8_t *pnBodyDynamic, uint64_t *pnTimeStamp);
接口描述	获取 ESP 滚动计数信息 （暂未实现）

4.4.15 获取横摆角速度信息

API 名称	int32_t GetYawRateInfo(uint16_t *pnYawRate, uint16_t *pnYawRateOffset, uint64_t *pnTimeStamp);		
接口描述	获取车辆横摆角速度信息		
参数说明	输入参数	pnYawRate（指向横摆角速度信息的指针）;	
		pnYawRateOffset（指向横摆角速度偏移量信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	实际横摆角速度与 pnYawRate 指向的值之间计算公式： 横摆角速度 =（pnYawRate 指向的值 × 0.002132603）－ 2.0943 rad/s pnYawRate 指向的值范围：[0x0，0x07AC] 对应横摆角速度的范围：[-2.0943，2.0943] rad/s	
		实际横摆角速度偏移量与 pnYawRateOffset 指向的值之间计算公式： 横摆角速度偏移量 =（pnYawRateOffset 指向的值 × 0.002132603） － 0.13 rad/s pnYawRateOffset 指向的值范围：[0x0，0x79] 对应横摆角速度偏移量的范围：[-0.13,0.13]	
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.16 获取车辆纵向加速度

API 名称	int32_t GetAccelerationXInfo(uint16_t *pnAX, uint16_t *pnAXOffset, uint64_t *pnTimeStamp);
接口描述	获取车辆纵向加速度信息

参数说明	输入参数	pnAX（指向纵向加速度信息的指针）;
		pnAXOffset（指向纵向加速度偏移量信息的指针）;
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）
	输出参数	实际车辆纵向加速度与 pnAX 指向的数值之间计算公式： 车辆纵向加速度 =（pnAX 指向的数值 × 0.027126736） — 21.593 m/s^2 pnAX 指向的数值范围：[0x0，0x638] 对应车辆纵向加速度范围： [-21.593，21.593] m/s^2
		实际车辆纵向加速度偏移量与 pnAXOffset 指向的数值之间计算公式： 车辆纵向加速度偏移量 = （pnAXOffset 指向的数值 × 0.027126736）— 21.593 m/s^2 pnAXOffset 指向的数值范围：[0x0，0x638] 对应车辆纵向加速度偏移量范围：[-21.593，21.593] m/s^2
		pnTimeStamp（指向读取到的时间计数标记）
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）	函数调用失败，读取数据失败

4.4.17 获取车辆横向加速度

API 名称	int32_t GetAccelerationYInfo(uint16_t *pnAY, uint16_t *pnAYOffset, uint64_t *pnTimeStamp);	
接口描述	获取车辆横向加速度信息	
参数说明	输入参数	pnAY（指向车辆横向加速度信息的指针）;
		pnAYOffset（指向车辆横向加速度偏移量信息的指针）;
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）

	输出参数	实际车辆横向加速度与 pnAY 指向数值之间的计算公式： 车辆横向加速度 =（pnAY 指向数值 × 0.027126736） — 21.593 m/s^2 pnAY 指向数值的范围：[0x0，0x638] 对应车辆横向加速度范围：[-21.593，21.593] m/s^2
		实际车辆横向加速度偏移量与 pnAYOffset 指向数值之间的计算公式： 车辆横向加速度偏移量 = （pnAYOffset 指向数值 × 0.027126736）— 21.593 m/s^2 pnAYOffset 指向数值的范围：[0x0，0x638] 对应车辆横向加速度偏移量的范围：[-21.593，21.593] m/s^2
		pnTimeStamp（指向读取到的时间计数标记）
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）	函数调用失败，读取数据失败

4.4.18 获取轮速脉冲信息

API 名称	int32_t GetWheelPulseCounterInfo(uint16_t *pnFL, uint16_t *pnFR, uint16_t *pnRL, uint16_t *pnRR, uint64_t *pnTimeStamp);	
接口描述	获取获取轮速脉冲计数信息	
参数说明	输入参数	pnFL（指向左前轮速脉冲计数信息的指针）;
		pnFR（指向右前轮速脉冲计数信息的指针）;
		pnRL（指向左后轮速脉冲计数信息的指针）;
		pnRR（指向右后轮速脉冲计数信息的指针）;

		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	左前、右前、左后、右后轮速脉冲计数与指针指向数值之间计算公式： 轮速脉冲计数 = 上述指针指向数值 × 1； pnFL、pnFR、pnRL、pnRR 指向数值范围：[0，0xFE] 对应轮速脉冲计数的范围：[0，1022]	
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.19 获取油门深度信息

API 名称	int32_t GetAccelerateDeepnessInfo(uint8_t *pnAccelerateDeepness, uint64_t *pnTimeStamp);		
接口描述	获取油门深度信息		
参数说明	输入参数	pnAccelerateDeepness（指向油门深度信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	实际油门深度与 pnAccelerateDeepness 指向数值之间的计算公式： 油门深度 = pnAccelerateDeepness 指向数值 × 1% pnAccelerateDeepness 指向数值的范围：[0x0，0x64] 对应油门深度的数值范围：[0%，100%]	
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.20 获取制动深度信息

API 名称	int32_t GetBrakeDeepnessInfo(uint8_t *pnBrakeDeepness, uint64_t *pnTimeStamp);		
接口描述	获取制动深度信息		
参数说明	输入参数	pnBrakeDeepness（指向制动深度信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	实际制动深度与 pnBrakeDeepness 指向数值之间的计算公式： 制动深度 = pnBrakeDeepness 指向数值 × 1% pnBrakeDeepness 指向数值的范围：[0x0，0x64] 对应制动深度的数值范围：[0%，100%]	
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入指针为空）	
	ERR_BYD_AUTO_READ （-6）	函数调用失败，读取数据失败	

4.4.21 获取制动踏板状态信息

API 名称	int32_t GetBrakePedalSignalInfo(uint8_t *pnBrakePedalSignal, uint64_t *pnTimeStamp);		
接口描述	获取制动踏板状态信息		
参数说明	输入参数	pnBrakePedalSignal（指向制动踏板状态信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	制动踏板状态：	
		BYD_AUTO_BRAKE_PEDAL_SIGNAL_NO_PRESSED （0）	未踩下
		BYD_AUTO_BRAKE_PEDAL_SIGNAL_PRESSED （1）	踩下
		BYD_AUTO_BRAKE_PEDAL_SIGNAL_ERR （3）	信号错误
		pnTimeStamp（指向读取到的时间计数标记）	

返回值	ERR_BYD_AUTO_OK （0）	函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）	函数调用失败，读取数据失败

4.4.22 获取档位信息

API 名称	int32_t GetCarGearInfo(uint8_t *pnCarGear, uint64_t *pnTimeStamp);		
接口描述	获取档位信息		
参数说明	输入参数	pnCarGear（指向档位信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	档位信息：	
		BYD_AUTO_CAR_GEAR_INVALID （0）	无效
		BYD_AUTO_CAR_GEAR_P （1）	P 档
		BYD_AUTO_CAR_GEAR_R （2）	R 档
		BYD_AUTO_CAR_GEAR_N （3）	N 档
		BYD_AUTO_CAR_GEAR_D （4）	D 档
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.23 获取档位驾驶模式反馈

API 名称	int32_t GetGearDriveModeFeedbackInfo(uint8_t* pnGearDriveModeFeedback, uint64_t* pnTimeStamp);

接口描述	获取档位驾驶模式反馈信息		
参数说明	输入参数	pnGearDriveModeFeedback（指向档位驾驶模式反馈信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	档位驾驶模式反馈：	
		BYD_AUTO_DRIVE_MODE_GEAR_FEEDBACK_NORMAL （0）	正常驾驶
		BYD_AUTO_DRIVE_MODE_GEAR_FEEDBACK_AUTO （1）	自动驾驶
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入指针为空）	
	ERR_BYD_AUTO_READ （-6）	函数调用失败，读取数据失败	

4.4.24 获取电子手刹状态信息

API 名称	int32_t GetEleBrakeInfo(uint8_t *pnEleBrake, uint64_t *pnTimeStamp);		
接口描述	获取电子手刹（EPB）状态信息		
参数说明	输入参数	pnEleBrake（指向电子手刹状态信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	电子手刹状态信息：	
		BYD_AUTO_EPB_GET_RELEASING （0）	正在释放
		BYD_AUTO_EPB_GET_RELEASED （1）	释放
		BYD_AUTO_EPB_GET_APPLYING （2）	正在拉起
		BYD_AUTO_EPB_GET_APPLIED （3）	拉起
		BYD_AUTO_EPB_GET_BEAKE_FAULT （4）	信息错误

		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.25 获取电子手刹驾驶模式反馈信息

API 名称	int32_t GetEleBrakeDriveModeFeedbackInfo(uint8_t* pnEleBrakeDriveModeFeedback, uint64_t* pnTimeStamp);		
接口描述	获取电子手刹（EPB）驾驶模式反馈信息		
参数说明	输入参数	pnEleBrakeDriveModeFeedback（指向 EPB 驾驶模式反馈信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	电子手刹（EPB）驾驶模式反馈信息：	
		BYD_AUTO_DRIVE_MODE_ELE_BRAKE_FEEDBACK_INVALID （0）	无效
		BYD_AUTO_DRIVE_MODE_ELE_BRAKE_FEEDBACK_NORMAL （1）	正常驾驶
		BYD_AUTO_DRIVE_MODE_ELE_BRAKE_FEEDBACK_AUTO （2）	自动驾驶
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.26 获取方向盘角度信息

API 名称	int32_t GetAngularInfo(uint16_t *pnAngular, uint64_t *pnTimeStamp);		
接口描述	获取方向盘角度信息		
参数说明	输入参数	pnAngular（指向方向盘角度信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	实际方向盘角度与 pnAngular 指向的数值之间计算公式： 方向盘角度 = （int16_t）（pnAngular 指向的数值）× 0.1 pnAngular 指向的数值范围：[0xE188，0x1E77] 对应方向盘角度数值范围：[-780.0，799.9]度	
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.27 获取 EPS 当前驾驶模式信息

API 名称	int32_t GetEPSDriveModeFeedbackInfo(uint8_t *pnEPSDriveModeFeedback, uint64_t *pnTimeStamp);		
接口描述	获取 EPS 驾驶模式信息		
参数说明	输入参数	pnEPSDriveModeFeedback（指向 EPS 驾驶模式信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	EPS 驾驶模式信息：	
		BYD_AUTO_DRIVE_MODE_EPS_FEEDBACK_INVALID （0）	无效
		BYD_AUTO_DRIVE_MODE_EPS_FEEDBACK_NORMAL （1）	正常驾驶

		BYD_AUTO_DRIVE_MODE_EPS_FEEDBACK_AUTO （2）	自动驾驶
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.28 获取 EPS 进入智能驾驶允许信息

API 名称	int32_t GetDriveAutoPermissionInfo(uint8_t *pnDriveAutoPermission, uint64_t *pnTimeStamp);		
接口描述	获取 EPS 进入智能驾驶模式允许信息		
参数说明	输入参数	pnDriveAutoPermission（指向 EPS 进入智能驾驶模式允许信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	EPS 进入智能驾驶模式允许信息：	
		BYD_AUTO_EPS_PERMISSION_INVALID （0）	无效
		BYD_AUTO_EPS_PERMISSION_OK （1）	允许进入
		BYD_AUTO_EPS_NOT_PERMISSION （2）	不允许进入
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.29 获取方向盘旋转速度信息

API 名称	int32_t GetRotationSpeedInfo(uint8_t *pnRotationSpeed, uint64_t *pnTimeStamp);
接口描述	获取方向盘旋转速度信息

参数说明	输入参数	pnRotationSpeed（指向方向盘旋转速度信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	实际方向盘旋转速度与 pnRotationSpeed 指向数值之间的计算公式： 方向盘旋转速度 =（pnRotationSpeed 指向数值 × 4）° /s pnRotationSpeed 指向数值的范围：[0，0xFE] 对应方向盘旋转速度的数值范围：[0，1016] ° /s	
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.30 获取电机扭矩信息

API 名称	int32_t GetMotorTorqueInfo(uint16_t *pnMotorTorque, uint64_t *pnTimeStamp);	
接口描述	获取电机扭矩信息	
参数说明	输入参数	pnMotorTorque（指向电机扭矩信息的指针）;
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）
	输出参数	实际电机扭矩与 pnMotorTorque 指向数值之间的计算公式： 电机扭矩 =（pnMotorTorque 指向数值 － 12000）NM pnMotorTorque 指向数值的范围：[0x2DAA，0x3016] 对应电机扭矩的数值范围：[-310，310] NM

		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.31 获取电机控制器驾驶模式反馈信息

API 名称	int32_t GetMCDriveModeFeedbackInfo(uint8_t *pnMCDriveModeFeedback, uint64_t *pnTimeStamp);		
接口描述	获取电机控制器驾驶模式反馈信息		
参数说明	输入参数	pnMCDriveModeFeedback（指向电机控制器驾驶模式反馈信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	电机控制器驾驶模式反馈信息：	
		BYD_AUTO_DRIVE_MODE_MC_FEEDBACK_NORMAL （0）	正常驾驶
		BYD_AUTO_DRIVE_MODE_MC_FEEDBACK_AUTO （1）	自动驾驶
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）		函数调用成功
	ERR_BYD_AUTO_INVALID_PARAM （-3）		函数调用失败，无效参数（例如，输入指针为空）
	ERR_BYD_AUTO_READ （-6）		函数调用失败，读取数据失败

4.4.32 获取主缸压力值信息（尚未实现）

API 名称	int32_t GetMCPressureInfo(uint16_t *pnPressure, uint8_t *pnPressureState, uint16_t *pnPressureOffset, uint8_t *pnPressureOffsetState, uint64_t *pnTimeStamp);

接口描述	获取主缸压力值信息 （暂未实现）

4.4.33 获取智能驾驶开关信息

API 名称	int32_t GetAutoDriveKeyInfo(uint8_t *pnAutoDriveKey, uint64_t *pnTimeStamp);		
接口描述	获取智能驾驶开关信息		
参数说明	输入参数	pnAutoDriveKey（指向智能驾驶开关信息的指针）;	
		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	智能驾驶开关信息：	
		BYD_AUTO_MEDIUM_KEY_INVALID （0）	无效
		BYD_AUTO_MEDIUM_KEY_DRIVE_AUTO_SWITCH （5）	开关按下
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数 （例如，输入指针为空）	
	ERR_BYD_AUTO_READ （-6）	函数调用失败，读取数据失败	

4.4.34 获取智能驾驶紧急退出按键信息

API 名称	int32_t GetEmergencyExitKeyInfo(uint8_t *pnEmergencyExitKey, uint64_t *pnTimeStamp);	
接口描述	获取智能驾驶紧急退出按键信息	
参数说明	输入参数	pnEmergencyExitKey（指向智能驾驶紧急退出按键信息的指针）;

		pnTimeStamp（指向时间计数标记的指针，用于判断时间戳是否有更新）	
	输出参数	智能驾驶紧急退出按键信息：	
		BYD_AUTO_MEDIUM_KEY_INVALID （0）	无效
		BYD_AUTO_MEDIUM_KEY_STOP_SWITCH （2）	按下紧急退出键
		pnTimeStamp（指向读取到的时间计数标记）	
返回值	ERR_BYD_AUTO_OK （0）	函数调用成功	
	ERR_BYD_AUTO_INVALID_PARAM （-3）	函数调用失败，无效参数（例如，输入指针为空）	
	ERR_BYD_AUTO_READ （-6）	函数调用失败，读取数据失败	

5 API 例程说明

比亚迪汽车工业有限公司为广大自动驾驶开发者提供的 SDK 软件包之中包含了 API 例程，通过该例程演示 API 的具体使用细节。

说明：此 API 例程已经在 Ubuntu 16.04 系统的 x64 电脑上面测试通过。

5.1 文件说明

首先将 SDK 软件放置在 LINUX 用户相关目录下，然后通过 cd 命令进入 Developer_Platform_Release 目录下，如下图所示：

```
gaoshangtian@level3-HP:/$ cd home/gaoshangtian/Developer_Platform_Release/
gaoshangtian@level3-HP:~/Developer_Platform_Release$ ls
docs  examples  include  libs  testBench
gaoshangtian@level3-HP:~/Developer_Platform_Release$
```

其中 Developer_Platform_Release/libs/ldriving_CANlib.a 为 LINUX 下自动驾驶开发者进行开发所需要的库文件，如下图所示：

```
gaoshangtian@level3-HP:~/Developer_Platform_Release/libs$ ls
idriving_CANlib.a
```

而 Developer_Platform_Release/include 包含调用库函数所需要的头文件,如下图所示：

```
gaoshangtian@level3-HP:~/Developer_Platform_Release/include$ ls
idriving_api.h  std_types.h
```

Developer_Platform_Release/testBench/智能驾驶-秦 Pro 模拟平台.exe（运行在 windows 平台）是用于模拟秦 Pro 整车的，其中 ControlCAN.dll 和 kerneldlls 下相关 dll 文件为此应用所需的动态链接库，本案例可以通过 2 个 CAN 盒与它进行 PC 端测试，也可与秦 Pro 整车进行通信操作，位置如下图红色框图所示：

```
gaoshangtian@level3-HP:~/Developer_Platform_Release/testBench$ ls
ControlCAN.dll  kerneldlls  智能驾驶-ACU模拟平台.exe  智能驾驶-秦Pro模拟平台.exe
```

Developer_Platform_Release/testBench/智能驾驶-ACU 模拟平台.exe（运行在 windows 平台）是基于自动驾驶 API 开发的应用软件，它主要为了验证 API 的正确性和可靠性，目前可通过它在 PC 端运行并与秦 Pro 整车进行 CAN 通信来实现整车自动驾驶所需信息的获取和控制，其位置如下图红色框图所示：

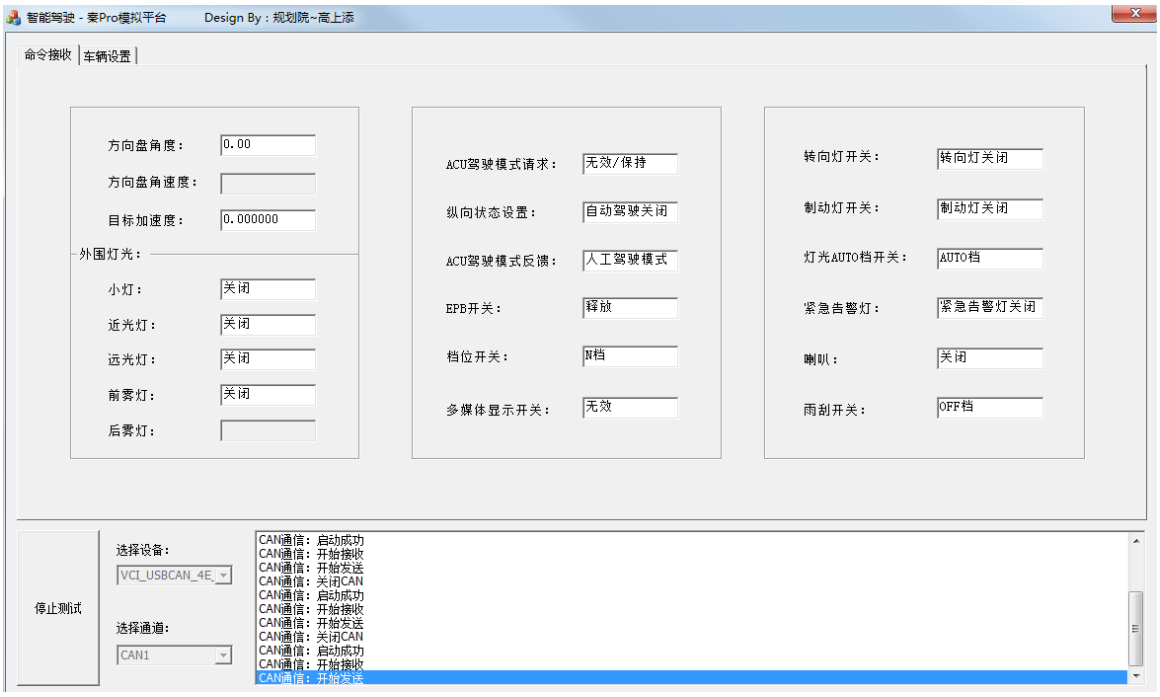
```
gaoshangtian@level3-HP:~/Developer_Platform_Release/testBench$ ls
ControlCAN.dll  kerneldlls  智能驾驶-ACU模拟平台.exe  智能驾驶-秦Pro模拟平台.exe
```

5.2 工具使用

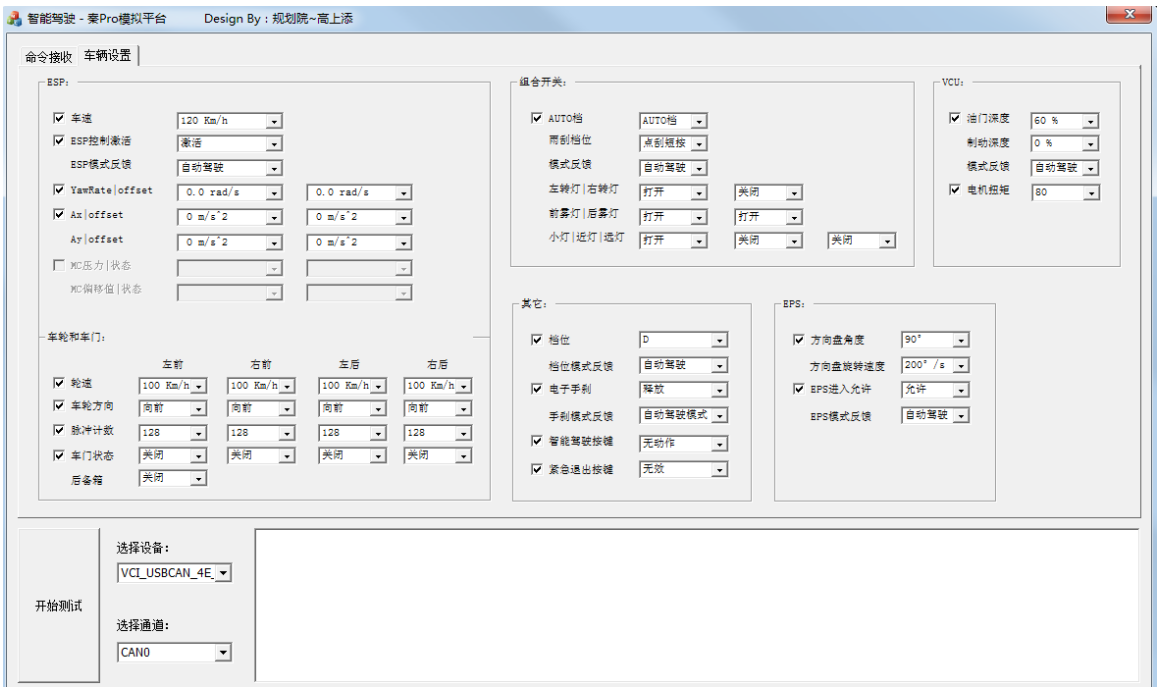
5.2.1 智能驾驶-秦 Pro 模拟平台.exe

将 testBench 整个文件夹放在 windows 平台下，进入文件夹后，双击“智能驾驶-秦 Pro 模拟平台.exe”，首先出现如下界面，默认是先显示“命令接收”（可以通过左上角的 tab 控件对“命令接收”和“车辆设置”

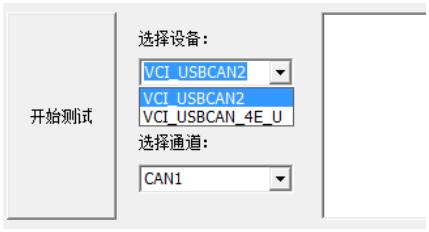
进行切换）界面。此界面可以显示 LINUX 平台下的 test_app 应用发出的设置命令信息，也可以显示“智能驾驶-ACU 模拟平台.exe”工具的“命令设置”的设置值（包括方向盘、加速度、组合开关等等）。

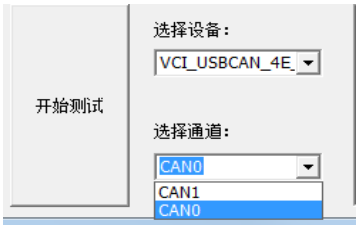


切换至“车辆设置“，其界面如下，可以在此页面分别对 ESP 等相关参数、车门、车轮、组合开关、VCU、EPS 等等相关执行器与自动驾驶相关的参数进行设置，以此来模拟秦 Pro 的整车自动驾驶相关数据。值得注意的是，这下面设置的参数有限，只是为了验证协议的正确，其中带“✓”控件是用来控制右侧选项框使能，这带钩情况下，可以进行设置，如果去掉钩，则保持当前的设置值，并且不能操作右侧控件选项。

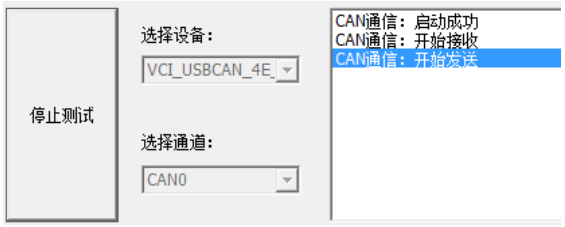


CAN 设备连接：本工具是设置了 2 个 CAN 设备选择（VCI_USBCAN2 和 VCI_USBCAN_4E_U）和 2 个通道选择，在上图的左下角所示（详细如下所示），当然可以根据需求，后续增加相关的设备和通道选择。



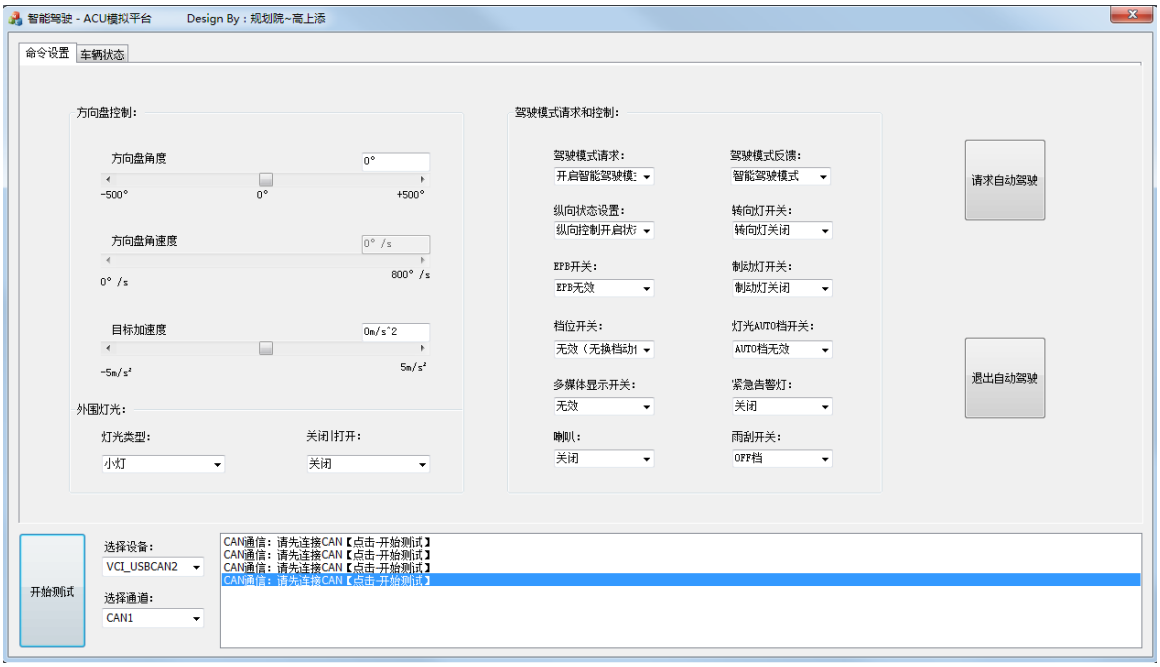


在选择完相关的设备和通道后，点击“开始测试”，连接成功则显示如下，如果显示“CAN 通信：打开设备失败!”，先检查是否有其他 CAN 通信软件连接到 CAN 设备，有则关闭其他软件，如果没有可以点击如下的“停止测试”按钮退出，然后重新点击“开始测试”。

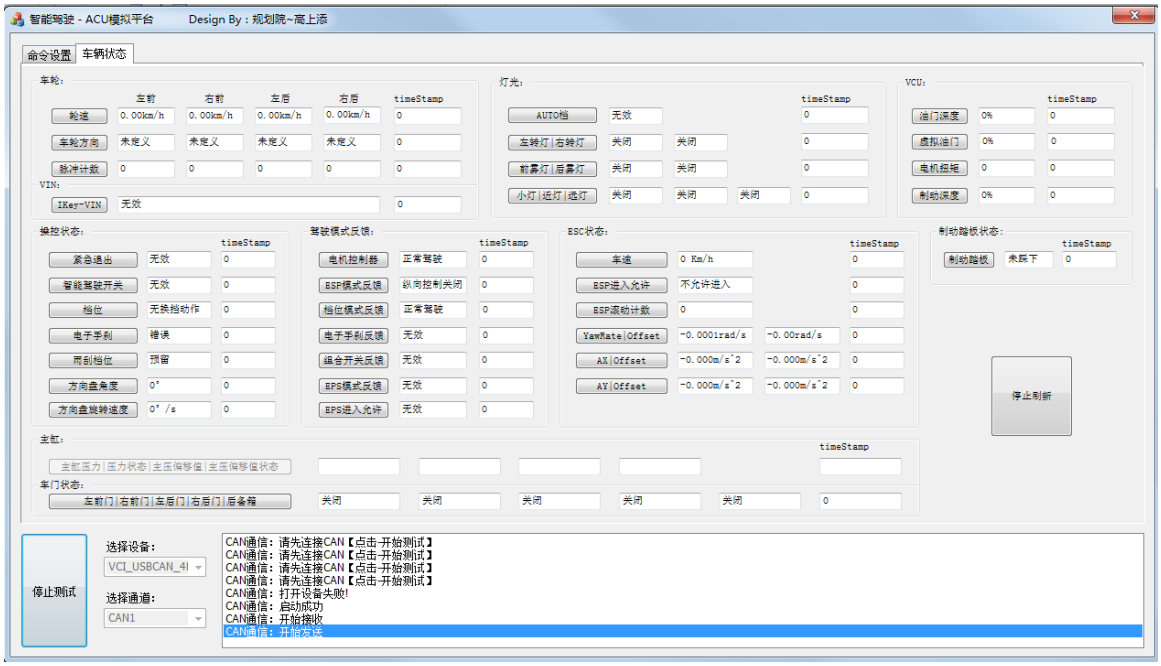


5.2.2 智能驾驶-ACU 模拟平台.exe

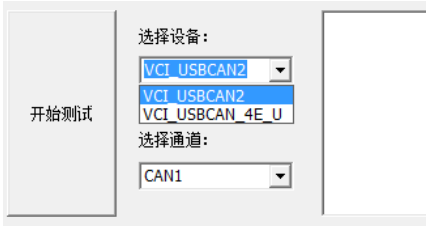
将 testBench 整个文件夹放在 windows 平台下，进入文件夹后，双击“智能驾驶-ACU 模拟平台.exe”，此软件是基于 API 生成的库文件进行开发的软件，可以验证 API 的可用性、可行性。首先出现如下界面，默认是先显示“命令设置”（可以通过左上角的 tab 控件对“命令设置”和“车辆状态”进行切换）界面。在如下界面中，未进入自动驾驶，则屏蔽方向盘和目标加速度及等操作，事实上整车相关执行部件此时也不响应相应的操作，当请求自动驾驶并成功进入之后，则开发相关的控件进行操作，也可进行退出自动驾驶操作（目前按停车状态下退出自动驾驶设计的，所以在行驶中如果要退出自动驾驶，请踩刹车!）

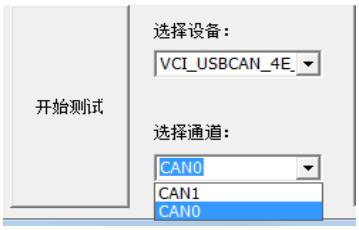


切换至“车辆状态“，从界面可以看到有车轮、车门、ESP 和组合开关相关信息，也有 VCU、档位、方向盘、驾驶模式反馈等自动驾驶相关的信息，可以通过此来查询整车的驾驶状况，如果点击右下角停止刷新，则可单独对每个项目进行点击以此来单个单次刷新，以此作为分析问题的参考。如果收到紧急退出，ACU 模拟端将停止发送控制命令，此时要按左下角停止测试退出当前状态，然后重新点击开始测试！如果实际是跟网关交互，那么车辆状态的数据也会停止，因为网关也停止转发相关报文。

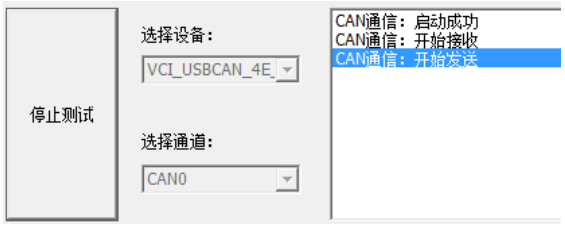


CAN 设备连接：本工具是设置了 2 个 CAN 设备选择（VCI_USBCAN2 和 VCI_USBCAN_4E_U）和 2 个通道选择，在上图的左下角所示（详细如下所示），当然可以根据需求，后续增加相关的设备和通道选择。





在选择完相关的设备和通道后，点击“开始测试”，连接成功则显示如下，如果显示“CAN 通信：打开设备失败!”，先检查是否有其他 CAN 通信软件连接到 CAN 设备，有则关闭其他软件，如果没有可以点击如下的“停止测试”按钮退出，然后重新点击“开始测试”。



5.3 生成可执行文件

第一步：进入 Developer_Platform_Release/examples/Idriving_CANlib_Example，如下图所示：

```
gaoshangtian@level3-HP:~/Developer_Platform_Release/examples/Idriving_CANlib_Example$ ls
binary  include  Makefile  source
```

第二步：输入 make 回车后，显示文字如下：

```
gaoshangtian@level3-HP:~/Developer_Platform_Release/examples/Idriving_CANlib_Example$ make
gcc -g -Wall source/api_test.c source/main.c source/api_test_get.c source/api_test_autodrive.c source/api_test_set.c source/api_test_get_demo.c source/api_test_set_demo.c  ./../../libs/idriving_CANlib.a -o binary/test_app -I./include -I./../../include -lpthread -lusbcan -lm -lrt
```

此时在 Developer_Platform_Release/examples/Idriving_CANlib_Example/binary 文件下生成文件名 test_app 可执行文件，如下图所示：

```
gaoshangtian@level3-HP:~/Developer_Platform_Release/examples/Idriving_CANlib_Example/binary$ ls
test_app
```

5.4 运行可执行文件

生成可执行文件后，先将“智能驾驶-秦 Pro 模拟平台.exe”在 windows 端开启并通过 CAN 通信连接到 linux 端，此时输入./binary/test_app 运行，如果出现如下出错（1542874280.500217：usbcan:ERR:unsigned int VCI_Transmit(unsigned int, unsigned int, CAN_OBJ*, unsigned int)<1084>:device not opened），请按 ctrl+c，然后输入 sudo ./binary/test_app 回车输入密码运行。

```
1542874280.480272: usbcan: ERR: unsigned int VCI_Transmit(unsigned int, unsigned int, unsigned int, CAN_OBJ*, unsigned int)<1084>: device not opened
1542874280.490207: usbcan: ERR: unsigned int VCI_Transmit(unsigned int, unsigned int, unsigned int, CAN_OBJ*, unsigned int)<1084>: device not opened
1542874280.500183: usbcan: ERR: unsigned int VCI_Transmit(unsigned int, unsigned int, unsigned int, CAN_OBJ*, unsigned int)<1084>: device not opened
```

成功运行后出现如下打印界面：

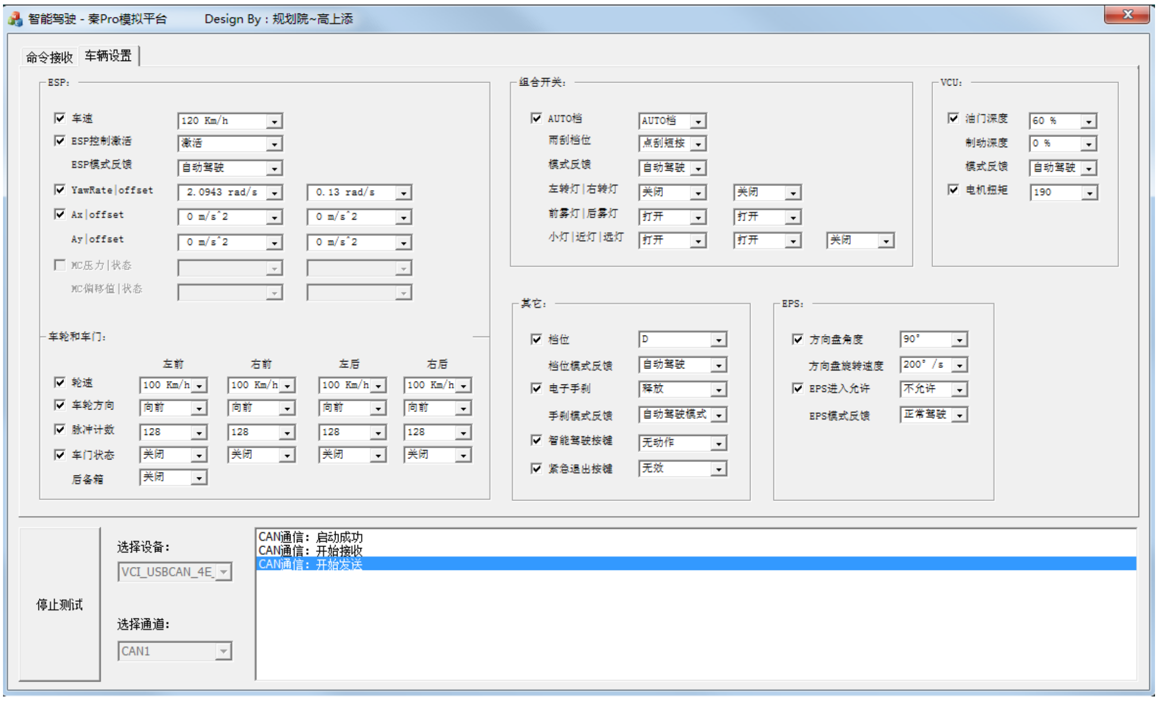
```
gaoshangtian@level3-HP:~/forwrok/07 开发者平台/Developer_Platform/Idriving_CANlib_Example$ sudo ./binary/test_app
VCI_OpenDevice succeeded
The AutoDrive init is done!
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Please enter ESC to terminate testing.
please enter 'g' or 'G' to start Get-func testing.
please enter 's' or 'S' to start Set-func testing.
please enter 'a' or 'A' to start AutoDrive testing.
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

5.4.1 命令‘g’或者‘G’

输入此命令后将进入智能驾驶开发者平台 API 的整车状态获取功能部分进行测试如下图所示，可以参考打印的文字信息进行操作。

```
g
Developer API Get-func testing Go...
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Please enter 'q' or 'Q' to Quit.
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Otherwise, enter:
g1  or G1:  GetVelocityInfo(uint16_t *pnVelocity, uint64_t *pnTimeStamp)
g2  or G2:  GetWheelSpeedInfo(uint16_t *pnFL, uint16_t *pnFR, uint16_t *pnRL, uint16_t *pnRR, uint64_t *pnTimeStamp)
g3  or G3:  GetLampInfo(uint8_t *pnSmall, uint8_t *pnNear, uint8_t *pnFar, uint64_t *pnTimeStamp)
g4  or G4:  GetLampTurnInfo(uint8_t *pnLeft, uint8_t *pnRight, uint64_t *pnTimeStamp)
g5  or G5:  GetLampFogInfo(uint8_t *pnFront, uint8_t *pnRear, uint64_t *pnTimeStamp)
g6  or G6:  GetRainWiperInfo(uint8_t *pnRainWiper, uint64_t *pnTimeStamp)
g7  or G7:  GetSwitchDriveModeFeedbackInfo(uint8_t *pnSwitchDriveModeFeedback, uint64_t *pnTimeStamp)
g8  or G8:  GetLampAutoInfo(uint8_t *pnLampAuto, uint64_t *pnTimeStamp)
g9  or G9:  GetDoorInfo(uint8_t *pnFL, uint8_t *pnFR, uint8_t *pnRL, uint8_t *pnRR, uint8_t *pnTrunk, uint64_t *pnTimeStamp)
g10 or G10: GetWheelDriveDirectionInfo(uint8_t *pnFL, uint8_t *pnFR, uint8_t *pnRL, uint8_t *pnRR, uint64_t *pnTimeStamp)
g11 or G11: GetESPDriveModeFeedbackInfo(uint8_t *pnESPDriveModeFeedback, uint64_t *pnTimeStamp)
g12 or G12: GetESPActiveInfo(uint8_t *pnActive, uint64_t *pnTimeStamp)
g13 or G13: GetESPAliveInfo(uint8_t *pnAlive, uint64_t *pnTimeStamp)
g14 or G14: GetYawRateInfo(uint16_t *pnYawRate, uint16_t *pnYawRateOffset, uint64_t *pnTimeStamp)
g15 or G15: GetAccelerationXInfo(uint16_t *pnAX, uint16_t *pnAXOffset, uint64_t *pnTimeStamp)
g16 or G16: GetAccelerationYInfo(uint16_t *pnAY, uint16_t *pnAYOffset, uint64_t *pnTimeStamp)
g17 or G17: GetWheelPulseCounterInfo( uint16_t *pnFL, uint16_t *pnFR, uint16_t *pnRL, uint16_t *pnRR, uint64_t *pnTimeStamp)
g18 or G18: GetAccelerateDeepnessInfo( uint8_t *pnAccelerateDeepness, uint64_t *pnTimeStamp)
g19 or G19: GetBrakeDeepnessInfo(uint8_t *pnBrakeDeepness, uint64_t *pnTimeStamp)
g20 or G20: GetBrakePedalSignalInfo(uint8_t *pnBrakePedalSignal, uint64_t *pnTimeStamp)
g21 or G21: GetCarGearInfo(uint8_t *pnCarGear, uint64_t *pnTimeStamp)
g22 or G22: GetGearDriveModeFeedbackInfo(uint8_t* pnGearDriveModeFeedback, uint64_t* pnTimeStamp)
g23 or G23: GetEleBrakeInfo(uint8_t *pnEleBrake, uint64_t *pnTimeStamp)
g24 or G24: GetEleBrakeDriveModeFeedbackInfo(uint8_t* pnEleBrakeDriveModeFeedback, uint64_t* pnTimeStamp)
g25 or G25: GetAngularInfo(uint16_t *pnAngular, uint64_t *pnTimeStamp)
g26 or G26: GetEPSDriveModeFeedbackInfo(uint8_t *pnEPSDriveModeFeedback, uint64_t *pnTimeStamp)
g27 or G27: GetDriveAutoPermissionInfo(uint8_t *pnDriveAutoPermission, uint64_t *pnTimeStamp)
g28 or G28: GetRotationSpeedInfo(uint8_t *pnRotationSpeed,  uint64_t *pnTimeStamp)
g29 or G29: GetMotorTorqueInfo(uint16_t *pnMotorTorque, uint64_t *pnTimeStamp)
g30 or G30: GetMCDriveModeFeedbackInfo(uint8_t *pnMCDriveModeFeedback, uint64_t *pnTimeStamp)
g31 or G31: GetAutoDriveKeyInfo(uint8_t *pnAutoDriveKey, uint64_t *pnTimeStamp)
g32 or G32: GetEmergencyExitKeyInfo(uint8_t *pnEmergencyExitKey, uint64_t *pnTimeStamp)
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

在 windows 端，我们先对“秦 Pro 模拟平台”的车辆状态进行设置，如下所示（仅作参考，可以根据自己的需求进行设置）。



接下来在上面界面输入‘g1’ or ‘G1’则对 GetVelocityInfo(uint16_t *pnVelocity, uint64_t *pnTimeStamp)进行测试，显示如下结果，速度先显示报文值 0x6d1，而后进行转化生成实际值 120(Km/h)。

```
g1
call GetVelocityInfo(uint16_t *pnVelocity, uint64_t *pnTimeStamp);
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
*pnVelocity: 0x6d1, *pnTimeStamp: 0x1bd.
real vehicle speed: 120.
```

再如输入’g32’或’G32’获取紧急退出信息 GetEmergencyExitKeyInfo(uint8_t *pnEmergencyExitKey, uint64_t *pnTimeStamp)，显示结果如下，可以对照 windows 端的工具设置值是否一致。

```
g32
call GetEmergencyExitKeyInfo(uint8_t *pnEmergencyExitKey, uint64_t *pnTimeStamp);
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
*pnEmergencyExitKey: 0x0, *pnTimeStamp: 0x41588.
emergency quit key is not pressed.
```

5.4.2 命令'q'或者'Q'

如果处于上面的车辆状态获取 API 的测试中，输入‘q’或者’Q’退出当前获取测试，回到上一层中。同样的在命令设置的相关 API 测试中也是输入‘q’或者’Q’来返回上层。具体可以参考每次给出的提示，如下：

```
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Please enter 'q' or 'Q' to Quit.
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

5.4.3 命令's'或者'S'

当退回到最初始的命令状态，可以输入’s’或者’S’，就可以进入命令设置相关 API 的测试，如下图所示：

```
s
Developer API Set-func testing Go...

-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Please enter 'q' or 'Q' to Quit.
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Otherwise, enter:
s1  or S1:  SendAutoDriveModeRequest(uint8_t nModeRequest)
s2  or S2:  SetLampAutoSwitch(uint8_t nAutoSwitch)
s3  or S3:  SetLampTurnSwitch(uint8_t nLeft, uint8_t nRight)
s4  or S4:  SetLampBrakeSwitch(uint8_t nBrakeSwitch)
s5  or S5:  SetAngular(uint16_t nAngular)
s6  or S6:  SetEpbState(uint8_t nEpbState)
s7  or S7:  SetRainWiper(uint8_t nRainWiper)
s8  or S8:  SetCarGear(uint8_t nCarGear)
s9  or S9:  SetDriveModeFeedback(uint8_t nDriveModeFeedback)
s10 or S10: SetAcceleratedVelocity(uint8_t nAcceleratedVelocity)
s11 or S11: SetDriveAutoState(uint8_t nDriveAutoState)
s12 or S12: SendAliveCount(uint8_t nCount)
s13 or S13: SetLampEmergencyWarningSwitch(uint8_t nSwitch)
s14 or S14: SetLampOutSideSwitch(uint8_t nArea, uint8_t nSwitch)
s15 or S15: SetHornSwitch(uint8_t nSwitch)
s16 or S16: SetMultimediaDisplaySwitch(uint8_t nSwitch)
```

先输入’s1’或者’S1’测试 ACU 向整车发起的进入自动驾驶请求 SendAutoDriveModeRequest(uint8_t nModeRequest)，显示如下：

```
s1
call SendAutoDriveModeRequest(uint8_t nModeRequest);

-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Please enter 'q' or 'Q' to Quit.
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Please enter ----
Number 0: Invalid or Hold current drive mode.
Number 1: Auto-drive mode request.
Number 2: Manual-drive mode request.
```

此时提供 API 函数 3 种输入情况，可以根据提示输入如’1’，显示结果如下，显示发送成功。

```
1
BYD_AUTO_DRIVE_MODE_OPEN: Auto-drive mode request.
```

根据提示输入’q’或者’Q’退出此当前 API 函数的设置，如下显示’SendAutoDriveModeRequest() Testing Quit.’ 从而进入新的命令设置相关的 API 测试当中。

```
q
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
SendAutoDriveModeRequest() Testing Quit.
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Please enter 'q' or 'Q' to Quit.
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Otherwise, enter:
s1  or S1:  SendAutoDriveModeRequest(uint8_t nModeRequest)
s2  or S2:  SetLampAutoSwitch(uint8_t nAutoSwitch)
s3  or S3:  SetLampTurnSwitch(uint8_t nLeft, uint8_t nRight)
s4  or S4:  SetLampBrakeSwitch(uint8_t nBrakeSwitch)
s5  or S5:  SetAngular(uint16_t nAngular)
s6  or S6:  SetEpbState(uint8_t nEpbState)
s7  or S7:  SetRainWiper(uint8_t nRainWiper)
s8  or S8:  SetCarGear(uint8_t nCarGear)
s9  or S9:  SetDriveModeFeedback(uint8_t nDriveModeFeedback)
s10 or S10: SetAcceleratedVelocity(uint8_t nAcceleratedVelocity)
s11 or S11: SetDriveAutoState(uint8_t nDriveAutoState)
s12 or S12: SendAliveCount(uint8_t nCount)
s13 or S13: SetLampEmergencyWarningSwitch(uint8_t nSwitch)
s14 or S14: SetLampOutSideSwitch(uint8_t nArea, uint8_t nSwitch)
s15 or S15: SetHornSwitch(uint8_t nSwitch)
s16 or S16: SetMultimediaDisplaySwitch(uint8_t nSwitch)
```

5.4.4 命令’a’或者’A’

当在最上层的命令选择当中，输入命令’a’或者’A’，则进入自动驾驶请求中，一旦整体各个执行器进入则 ACU 回馈自动驾驶状态给整车，那么整体进入自动驾驶状态中，并进行命令设置和状态获取等相关 API 的测试,如下界面所示。此时请求’q’或者’Q’则退回上一层界面。如果请求失败，则无法进入命令设置和车辆状态获取等 API 的测试，仍在最上层。

```
Developer API AutoDriving testing Go...

-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Please enter 'q' or 'Q' to quit.
Please enter 'o' or 'O' to quit AutoDrive.
Please enter 'h' or 'H' to show the Item again.
********************************************************************
**Under the automatic driving mode,please don't set s1、s9 and s11**
**that will change the state of the automatic driving mode!      **
**And enter 'b' or 'B' can reduce the speed and stop the vehicle! **
********************************************************************
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
Otherwise, enter:
-+-+-+-+-+-+-+-+-+-+-+-+Get the state of vehicle under the automatic driving mode!-+-+-+-+-+-+-+-+-+-+-+-+-+
g1  or G1:  GetVelocityInfo(uint16_t *pnVelocity, uint64_t *pnTimeStamp)
g2  or G2:  GetWheelSpeedInfo(uint16_t *pnFL, uint16_t *pnFR, uint16_t *pnRL, uint16_t *pnRR, uint64_t *pnTimeStamp)
g3  or G3:  GetLampInfo(uint8_t *pnSmall, uint8_t *pnNear, uint8_t *pnFar, uint64_t *pnTimeStamp)
g4  or G4:  GetLampTurnInfo(uint8_t *pnLeft, uint8_t *pnRight, uint64_t *pnTimeStamp)
g5  or G5:  GetLampFogInfo(uint8_t *pnFront, uint8_t *pnRear, uint64_t *pnTimeStamp)
g6  or G6:  GetRainWiperInfo(uint8_t *pnRainWiper, uint64_t *pnTimeStamp)
g7  or G7:  GetSwitchDriveModeFeedbackInfo(uint8_t *pnSwitchDriveModeFeedback, uint64_t *pnTimeStamp)
g8  or G8:  GetLampAutoInfo(uint8_t *pnLampAuto, uint64_t *pnTimeStamp)
g9  or G9:  GetDoorInfo(uint8_t *pnFL, uint8_t *pnFR, uint8_t *pnRL, uint8_t *pnRR, uint8_t *pnTrunk, uint64_t *pnTimeStamp)
g10 or G10: GetWheelDriveDirectionInfo(uint8_t *pnFL, uint8_t *pnFR, uint8_t *pnRL, uint8_t *pnRR, uint64_t *pnTimeStamp)
g11 or G11: GetESPDriveModeFeedbackInfo(uint8_t *pnESPDriveModeFeedback, uint64_t *pnTimeStamp)
g12 or G12: GetESPAutoDrivePermissionInfo(uint8_t *pnAutoDrivePermission, uint64_t *pnTimeStamp)
g13 or G13: GetESPAliveInfo(uint8_t *pnAlive, uint64_t *pnTimeStamp)
g14 or G14: GetYawRateInfo(uint16_t *pnYawRate, uint16_t *pnYawRateOffset, uint64_t *pnTimeStamp)
g15 or G15: GetAccelerationXInfo(uint16_t *pnAX, uint16_t *pnAXOffset, uint64_t *pnTimeStamp)
g16 or G16: GetAccelerationYInfo(uint16_t *pnAY, uint16_t *pnAYOffset, uint64_t *pnTimeStamp)
g17 or G17: GetWheelPulseCounterInfo( uint16_t *pnFL, uint16_t *pnFR, uint16_t *pnRL, uint16_t *pnRR, uint64_t *pnTimeStamp)
g18 or G18: GetAccelerateDeepnessInfo( uint8_t *pnAccelerateDeepness, uint64_t *pnTimeStamp)
g19 or G19: GetBrakeDeepnessInfo(uint8_t *pnBrakeDeepness, uint64_t *pnTimeStamp)
g20 or G20: GetBrakePedalSignalInfo(uint8_t *pnBrakePedalSignal, uint64_t *pnTimeStamp)
g21 or G21: GetCarGearInfo(uint8_t *pnCarGear, uint64_t *pnTimeStamp)
g22 or G22: GetGearDriveModeFeedbackInfo(uint8_t* pnGearDriveModeFeedback, uint64_t* pnTimeStamp)
g23 or G23: GetEleBrakeInfo(uint8_t *pnEleBrake, uint64_t *pnTimeStamp)
g24 or G24: GetEleBrakeDriveModeFeedbackInfo(uint8_t* pnEleBrakeDriveModeFeedback, uint64_t* pnTimeStamp)
g25 or G25: GetAngularInfo(uint16_t *pnAngular, uint64_t *pnTimeStamp)
g26 or G26: GetEPSDriveModeFeedbackInfo(uint8_t *pnEPSDriveModeFeedback, uint64_t *pnTimeStamp)
g27 or G27: GetDriveAutoPermissionInfo(uint8_t *pnDriveAutoPermission, uint64_t *pnTimeStamp)
g28 or G28: GetRotationSpeedInfo(uint8_t *pnRotationSpeed,  uint64_t *pnTimeStamp)
g29 or G29: GetMotorTorqueInfo(uint16_t *pnMotorTorque, uint64_t *pnTimeStamp)
g30 or G30: GetMCDriveModeFeedbackInfo(uint8_t *pnMCDriveModeFeedback, uint64_t *pnTimeStamp)
g31 or G31: GetAutoDriveKeyInfo(uint8_t *pnAutoDriveKey, uint64_t *pnTimeStamp)
g32 or G32: GetEmergencyExitKeyInfo(uint8_t *pnEmergencyExitKey, uint64_t *pnTimeStamp)
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
-+-+-+-+-+-+-+-+-+-+-+-+Set the state of vehicle under the automatic driving mode!-+-+-+-+-+-+-+-+-+-+-+-+
b   or B:   It' used to reduce the speed of vehicle and stop the vehicle!
s1  or S1:  SendAutoDriveModeRequest(uint8_t nModeRequest)
s2  or S2:  SetLampAutoSwitch(uint8_t nAutoSwitch)
s3  or S3:  SetLampTurnSwitch(uint8_t nLeft, uint8_t nRight)
s4  or S4:  SetLampBrakeSwitch(uint8_t nBrakeSwitch)
s5  or S5:  SetAngular(uint16_t nAngular)
s6  or S6:  SetEpbState(uint8_t nEpbState)
s7  or S7:  SetRainWiper(uint8_t nRainWiper)
s8  or S8:  SetCarGear(uint8_t nCarGear)
s9  or S9:  SetDriveModeFeedback(uint8_t nDriveModeFeedback)
s10 or S10: SetAcceleratedVelocity(uint8_t nAcceleratedVelocity)
s11 or S11: SetDriveAutoState(uint8_t nDriveAutoState)
s12 or S12: SendAliveCount(uint8_t nCount)
s13 or S13: SetLampEmergencyWarningSwitch(uint8_t nSwitch)
s14 or S14: SetLampOutSideSwitch(uint8_t nArea, uint8_t nSwitch)
s15 or S15: SetHornSwitch(uint8_t nSwitch)
s16 or S16: SetMultimediaDisplaySwitch(uint8_t nSwitch)
```

5.4.5 命令'o'或者'O'

在上述进入自动驾驶后，可以先看提示,此时的'o'或者'O'是请求退出自动驾驶，目前测试是在车速为 0Km/h 情况下进行退出操作（当然如果在行驶中需要退出自动驾驶模式或者遇到特别情况，可以通过踩刹车来实现,目前例程中在踩刹车状态下，纵向退出自动驾驶模式，当松开脚踏板，又重新进入自动驾驶模式，开发者可以根据自己的需求定制自己的退出逻辑），所以在实车测试此 test_app 应用需要注意这一点。

因为已经进入自动驾驶，所以在提示中也表明了's1'或者'S1'、's9'或者'S9'、's11'或者'S11'三个涉及自动驾驶请求或退出、纵向的开启或关闭以及 ACU 模式反馈等特殊命令控制最好不要操作，当然在车速为 0km/h 情况下，可以对 3 个命令设置 API 进行测试，尽量避免为好!!! 其他的命令设置和车辆状态获取等操作同上面的一样!

5.4.6 命令'h'或者'H'

如果看不到提示部分，可以输入'h'或者'H'来重新显示提示，此命令仅限在自动驾驶模式中。

5.4.7 命令'b'或者'B'

在自动驾驶模式中，输入'b'或者'B'命令用来减少车速并停车，这样可以应对通过命令端's10'或者'S10'对应的加速度控制 SetAccelerateVelocity(uint8_t nAcceleratedVelocity)因人为手动输入而导致的延时，当然应急情况下也可以通过踩刹车应对。如果在“智能驾驶-秦 Pro 模拟平台.exe”的“车辆设置”中将车辆设置不小于 0 值，则会到进入死循环中，如下图所示，此时需在模拟端，将车速设置成 0 值！这样就相当于模拟减速停车的效果!

```
b
The current speed of vehicle is 120 Km/h.
The current speed of vehicle is 120 Km/h.
The current speed of vehicle is 120 Km/h.
The current speed of vehicle is 120 Km/h.
The current speed of vehicle is 120 Km/h.
The current speed of vehicle is 120 Km/h.
The current speed of vehicle is 120 Km/h.
The current speed of vehicle is 120 Km/h.
```

5.4.8 命令 ESC

test_app 在运行的整个生命周期中，当接收到键盘输入的 ESC+回车，则退出程序，如下图所示，如果出现异常，可以按“ctrl+c”强制退出。当然在自动驾驶模式下，先输入’o’或者’O’请求退出自动驾驶模式，然后在 ESC 退出，否则只是软件的退出，秦 Pro 整车仍在自动驾驶模式当中，此时靠刹车退出自动驾驶模式较为稳当!!!

```
^[
cm_rcvThread End.
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
API Get-func Testing Quit.
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

acu_ctrlThread End.
can_rcvThread End.
VCI_CloseDevice
gaoshangtian@level3-HP:~/Developer_Platform_Release/examples/Idriving_CANlib_Example$
```

5.5 声明

至此，LINUX 下 DEMO 使用说明基本完结，需要了解具体详细的，请查看代码，里面已经按设置、获取、自动驾驶模式等模块分别进行编码，另外本案例只是用于测试 API 的使用，若要进入自动驾驶模式，请务必确认各种安全条件!!!