TSMasterAPI CPP 编程指导 V0.1

目录

1. 什么情况下需要此文档？

用户基于 C++平台的编程语言，对目前市面上主流的如 Vector，TOSUN，Peak，英特佩斯等工具进行二次开发的时候，需要参考本文档，调用 API 函数来实现对设备的程序控制。

2. 使用 SDK 工程

安装 TSMaster 文件后，在 SDK 的目录下面，提供了 C/C++,C#，LabView 等的示例工程。以本电脑为例，C++的 Demo 工程目录如下所示：

C:\Program Files (x86)\TOSUN\TSMaster\bin\Data\SDK\examples\C

进入目录后，可以看到包含内容如下：

demo0_load_api：基于 visual studio 的示例工程，演示如何载入初始化 dll。

demo1_virtualCAN_tx_rx：基于 visual studio 的示例工程，演示基于虚拟 CAN 总线收发 CAN 报文。

demo2_CAN_dev_map_and_rbs：基于 visual studio 的示例工程，演示如何映射不同的硬件设备并启动网络仿真程序。

qt_api_test：基于 QT 环境的示例工程，演示如何在 qtcreator 中载入初始化 dll。

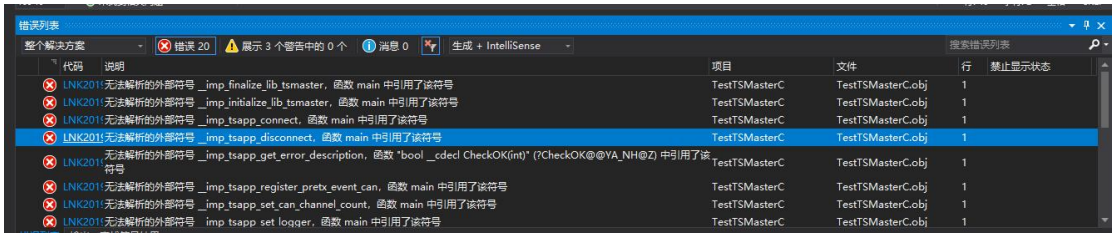
TSMasterApi：dll 接口文件

TSMasterAPI_Demos_CPP.sln：基于 visual studio 的示例工程集合

所有工程运行前必须安装 TSMaster 软件，工程默认使用 C:\Program Files (x86)\TOSUN\TSMaster\路径查找安装的 dll，如果存在区别，请修改代码中 class TSMasterApi 中函数 set_app_and_dll 的初始化 dll 路径参数。

1. 错误提示：

点击 TestTSMasterC.sln 工程，第一次编译的时候，往往会碰到如下报错：



这是因为编译模式默认为 x64，导致部分函数找不到。

2. 解决办法：

修改 vs 的编译模式，改为 x86。如果的确需要 x64 版本开发库，请和同星公司联系索取。

3. 驱动特点

本驱动提供了硬件抽象层，通过映射机制，同样一套 API 函数，可以应用于多款市面上主流的 CAN 工具硬件而不用修改代码。其作用原理图如下所示：

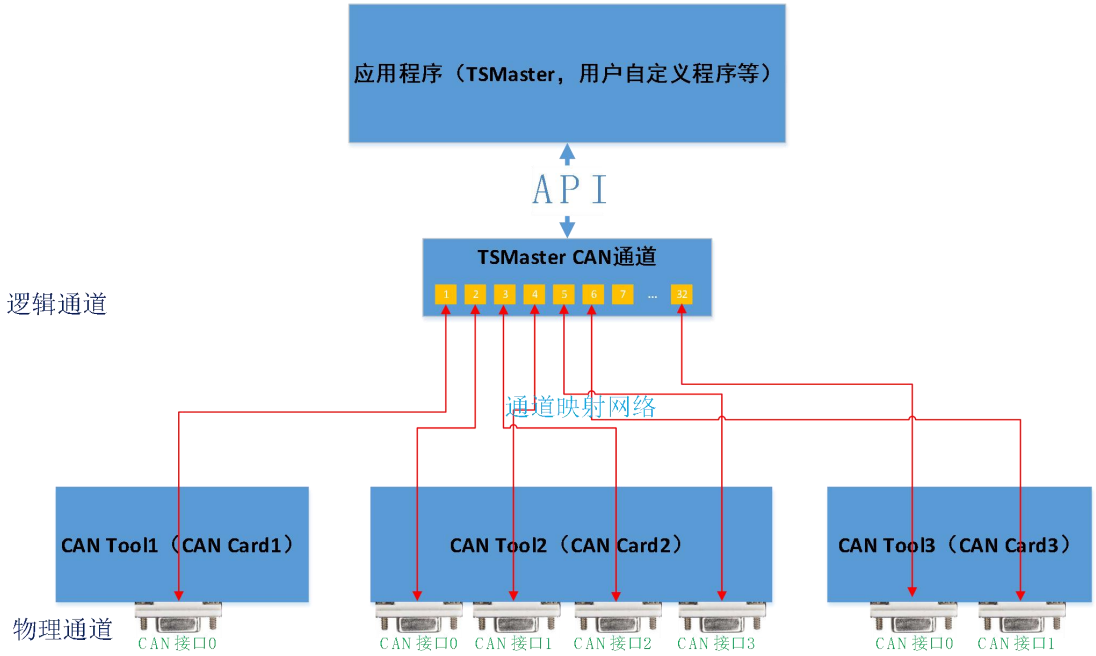


图 1. TSMasterAPI 通过逻辑通道给上层提供统一的 API 接口

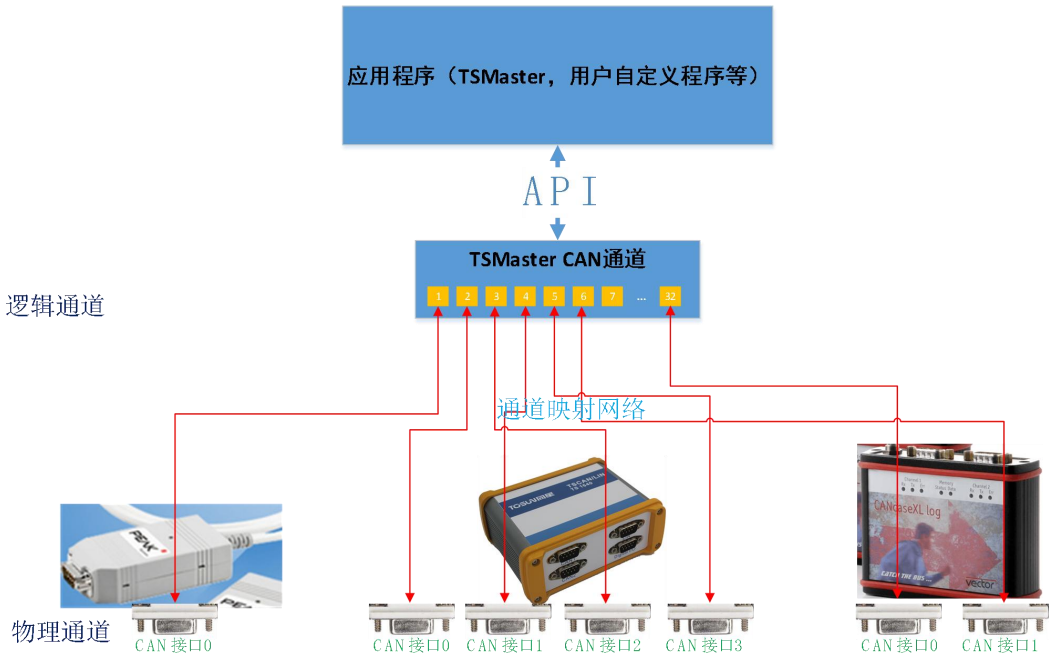


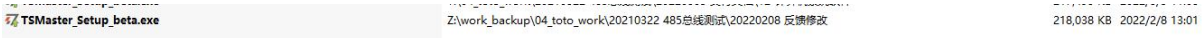
图 2. 基于 TSMasterAPI 混合连接不同硬件板卡示意图

如上图所示，TSMaster API 内部实现了对目前市面上主流硬件 CAN 卡的兼容，提供了统一的 API 接口给上层应用层。用户在开发上层的时候，只需要开发一套上层代码，底层硬件可以任意切换，为用户硬件选择提供了最大的灵活性。

4. 添加 API 库文件

1. 安装驱动运行环境 TSMaster

TSMaster API 提供了对多种工具平台的支持，因此要使用 TSMaster 驱动，首先要安装 TSMaster 运行环境。该运行环境大小 200 多兆，安装文件如下：

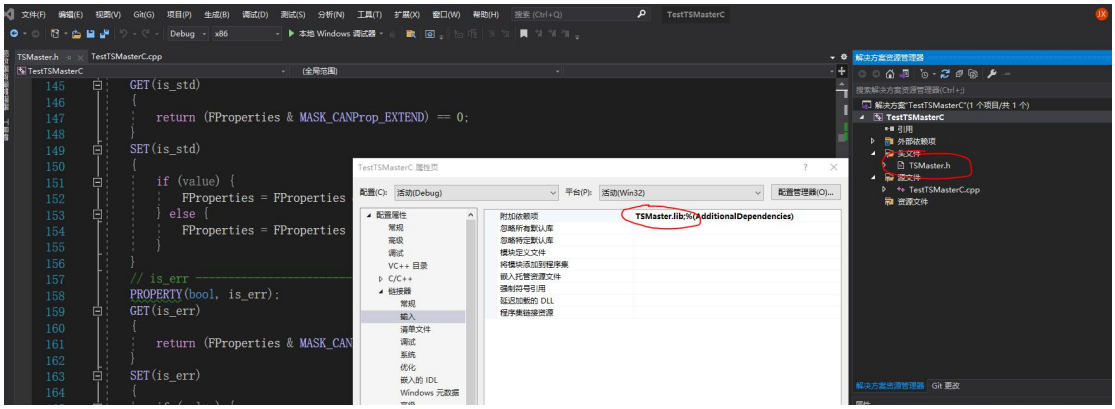


TSMaster 运行时环境为免费软件，可以直接到上海同星公司的官网上下载，下来链接为：http://www.tosun.tech/TOSUNSoftware/TSMaster_Setup.exe

下载该文件，并完成安装，然后进入下一步。

2. 添加库文件引用

安装完运行环境后，用户需要在自己的 visual studio 工程中添加头文件和 Lib 文件用于编译，设置如下图所示。



同时需要把 TSMaster.dll 添加到编译后 exe 所在文件夹，即完成了开发库文件的添加。添加完成后

添加完成后，用户工作目录中只需要留下两个文件（用户自己开发的 exe 文件，TSMaster.dll），即可实现对 CAN 总线工具的访问。如果出现缺失库文件的情况，请检查是否正确安装了 TSMaster 运行时环境，TSMaster.dll 需要安装 Microsoft Visual C++ Runtime library，请至少安装 2015 版本运行环境。

3. 注意事项（必看）：

用户目录下面只需要保留 TSMaster.dll 文件即可，千万不要把 libTSMaster.dll 等库文件也放在用户目录下面，这样会造成程序异常。对于 cpp 程序，用于只需要操作 TSMaster.dll 库文件即可，尽可能的保持文件目录的整洁，同时也避免带来不必要的错误。

4. 版本冲突解决方案

当因为库文件版本升级（比如升级 TSMaster 软件）造成库文件不匹配，从而引起软件不能正常工作时，千万不要慌张，按照如下方式解决：

TSMaster 的库文件 TSMaster.dll 跟 TSMaster 是严格绑定的。因此，如果自动升级了 TSMaster 软件造成的 API 调用异常，最简单的解决方式就是从 TSMaster 安装目录下的 SDK 文件中拷贝对应的 TSMaster.dll 文件。

如果以上操作还不能解决问题，请及时联系上海同星的工程师，会尽快帮忙解决问题。

5. 总线数据类型定义-TSMaster.h

1. TLibCAN：CAN 总线数据类型

```
typedef struct _TLibCAN {
    u8 FIdxChn;                  // channel index starting from 0
    TCANProperty FProperties;    //CAN Property
    u8 FDLC;                     // dlc from 0 to 8
    u8 FReserved;                // reserved to keep alignment
    s32 FIdentifier;             // CAN identifier
    u64 FTimeUS;                 // timestamp in us
    u8x8 FData;                  // 8 data bytes to send
} TLibCAN,*PLibCAN;
```

成员：

FData：帧数据。最大长度为 8Bytes

FDLC：帧长度。

FIdentifier：帧 ID，如果为 0xFFFFFFFF，表示当前帧为错误帧

FIdxChn：帧通道，注意 CHANNEL_INDEX. CHN1 = 0,实际上是从 0 开始计算的。

FTImeUS：帧时间戳，64 位 us 级时间戳。

FProperties：存储 CAN 相关的属性，比如是否远程帧，是否扩展帧。

其中，属性字节定义如下：

【1】 FProperties：CAN 属性定义：该参数默认为 0，共八个 bits，每一个位的定义如下：

Bit	意义
0	0：Rx 接收报文；1：Tx 发送报文
1	0：data frame 数据帧；1：remote frame 远程帧
2	0：std frame 标准帧；1：extended frame 扩展帧
3-5	Reserved
6	0：不记录；1：已经被记录
7	Reserved

构造函数：

本结构体没有构造函数，用户在定义结构体后可以通过 init_w_std_id（标准真）或者 init_w_ext_id（扩展帧）函数对结构体进行初始化，两个函数具有相同参数：

Aid：CAN 报文 ID;

ADLC：报文 DLC；

属性：

is_err:是否错误帧

is_std：是否标准帧

is_tx：是否发送出去的报文。True:该报文是发送出去的报文；false：该报文是接收的报文

is_data：是否为数据帧

方法：

load_data_array：从输入指针复制数据
set_data：指定 CAN 报文数据

调用示例：

```
TCAN can_frame;//创建一个报文结构体
can_frame.init_w_std_id(0x11, 8);//初始化ID为0x11,报文DLC为8
can_frame.set_data(0, 1, 2, 3, 4, 5, 6, 7);//设置报文数据
```

2. TLibCANFD: CANFD 总线数据类型

```
typedef struct _TLibCANFD {
    u8 FIdxChn;                          // channel index starting from 0
    TCANProperty FProperties;            //CAN Property
    u8 FDLC;                             // dlc from 0 to 15
    TCANFDProperty FFDProperties;        //FD Property
    s32    FIdentifier;                  // CAN identifier
    u64 FTimeUS;                         // timestamp in us
    u8x64 FData;                         // 64 data bytes to send
}TLibCANFD, * PLibCANFD;
```

关联函数：

tsapp_transmit_canfd_async 发送报文
tsfifo_receive_canfd_msgs 接收报文

成员：

FData：帧数据。最大长度为 64Bytes
FDLC：帧长度。
FIdentifier：帧 ID，如果为 0xFFFFFFFF，表示当前帧为错误帧
FIdxChn：帧通道，注意 CHANNEL_INDEX. CHN1 = 0,实际上是从 0 开始计算的。
FTImeUS：帧时间戳，64 位 us 级时间戳。
FFDProperties：存储 FD 相关的属性，如是否 FD 报文，发送过程中是否波特率可变。不同的字节位代表不同的属性值。
FProperties：存储 CAN 相关的属性，比如是否远程帧，是否扩展帧。

其中，两个属性字节定义如下：

【1】 FProperties：CAN 属性定义：该参数默认为 0，共八个 bits，每一个位的定义如下：

Bit	意义
0	0：Rx 接收报文；1：Tx 发送报文
1	0：data frame 数据帧；1：remote frame 远程帧
2	0：std frame 标准帧；1：extended frame 扩展帧
3-5	Reserved
6	0：不记录；1：已经被记录
7	Reserved

【2】 FDProperty: FD 属性定义：

Bit	意义
0	0：普通 CAN 报文；1：FDCAN 报文
1	0：关闭 BRS；1：开启 BRS
2	是否发生错误（ESI Flag）
3-7	Reserved

// [7-3] tbd
// [2] ESI, The E RROR S TATE I NDICATOR (ESI) flag is transmitted dominant by error active nodes, recessive by error passive nodes. ESI does not exist in CAN format frames
// [1] BRS, If the bit is transmitted recessive, the bit rate is switched from the standard bit rate of the A RBITRATION P HASE to the preconfigured alternate bit rate of the D ATA P HASE . If it is transmitted dominant, the bit rate is not switched. BRS does not exist in CAN format frames.
// [0] EDL: 0-normal CAN frame, 1-FD frame, added 2020-02-12, The E XTENDED D ATA L ENGTH (EDL) bit is recessive. It only exists in CAN FD format frames

构造函数：

本结构体没有构造函数，用户在定义结构体后可以通过 init_w_std_id（标准真）或者 init_w_ext_id（扩展帧）函数对结构体进行初始化，两个函数具有相同参数：

Aid：CAN 报文 ID;

ADLC：报文 DLC;

属性：

is_error:是否错误帧

is_std：是否标准帧

is_data：是否数据帧

is_tx：是否发送出去的报文。True:该报文是发送出去的报文；false：该报文是接收的报文

is_edl：是否是 CANFD 帧，如果设置为 false，则依然作为 Classic 帧发送。

Is_rbs：如果是 CANFD 帧，发送过程中是否切换波特率。如果为 false，则整个报文按照仲裁场速率发送。

Is_esi：发送节点错误状态，false 为被动错误，true 为主动错误。

方法：

load_data_array：从输入指针复制数据

get_data_length：获取报文数据长度

to_tcan：转化为 CAN 报文结构体

调用示例：

```
TCANFD can_frame;
can_frame.init_w_std_id(0x11, 0xf);//初始化ID为0x11,报文DLC为8
```

```
    for (int i = 0; i < 64; i++)
    {
        can_frame.FData[i] = i; //初始化报文发送内}
    }
```

3. TLibLIN：LIN 总线数据类型

```
typedef struct _TLIN {
    u8 FIdxChn;                 // channel index starting from 0
    u8 FErrCode;                //   0: normal
    TLINProperty FProperties;           // Property of LIN Message
    u8 FDLC;                    // dlc from 0 to 8
    u8 FIdentifier;             // LIN identifier:0--64
    u8 FChecksum;               // LIN checksum
    u8 FStatus;                 // place holder 1
    u64 FTimeUS;                // timestamp in us    //Modified by Eric 0321
    u8x8 FData;                 // 8 data bytes to send
  }TLibLIN, *PLibLIN;
```

成员：

FData：帧数据。最大程度为 8Bytes

FDLC：帧长度。

FIdentifier：帧 ID，如果为 0xFFFFFFFF，表示当前帧为错误帧

FIdxChn：帧通道，注意 CHANNEL_INDEX. CHN1 = 0,实际上是从 0 开始计算的。

FTImeUS：帧时间戳，64 位 us 级时间戳。

FProperties：存储 LIN 相关的属性，比如报文方向，是接收报文还是发送报文。

FStatus：报文状态。

FErrStatus：如果是错误帧，对应的错误类型。

其中，属性字节定义如下：

【1】 Properties：LIN 属性定义：该参数默认为 0，共八个 bits，每一个位的定义如下：

Bit	意义
0	0：Rx 接收报文；1：Tx 发送报文
1-3	Reserved
4-5	设备类型：主节点，从节点，监听节点
6	0：不记录；1：已经被记录
7	Reserved

构造函数

本结构体没有构造函数，使用函数 init_w_id 初始化报文结构，其中包括参数

Aid：LIN 报文 PID;

ADLC：报文长度；

属性：

is_tx: true: 报文是发送出去的报文；false：报文是接收回来的报文。

方法：

load_data：从输入指针复制数据

调用示例：

```
TLIN lin_frame;
lin_frame.init_w_id(0x11, 8); //初始化PID为0x11  报文长度8
lin_frame.property__set_is_tx(true);//设置为发送报文
for (u32 i = 0; i < 8; i++) {
    lin_frame.FData[i] = 0;
}
```

4. TLibFLexRay：flexray 总线数据类型

```
typedef struct _TFlexRay {
    u8  FIdxChn;               // channel index starting from 0
    u8  FChannelMask;          // 0: reserved, 1: A, 2: B, 3: AB
    u8  FDir;                  // 0: Rx, 1: Tx, 2: Tx Request
    u8  FPayloadLength;        // payload length in bytes
    u8  FActualPayloadLength;  // actual data bytes
    u8  FCycleNumber;          // cycle number: 0~63
    u8  FCCType;               // 0 = Architecture independent, 1 = Invalid CC type, 2 =
                                  Cyclone I, 3 = BUSDOCTOR, 4 = Cyclone II, 5 = Vector VN
                                  interface, 6 = VN - Sync - Pulse(only in Status Event,
                                  for debugging purposes only)
    u8  FFrameType;            // 0 = raw flexray frame, 1 = error event, 2 = status, 3 =
                                  start cycle
    u16 FHeaderCRCA;           // header crc A
    u16 FHeaderCRCB;           // header crc B
    u16 FFrameStateInfo;       // bit 0~15, error flags
    u16 FSlotId;               // static seg: 0~1023
    u32 FFrameFlags;           // bit 0~22
                               // 0 1 = Null frame.
                               // 1 1 = Data segment contains valid data
                               // 2 1 = Sync bit
                               // 3 1 = Startup flag
                               // 4 1 = Payload preamble bit
                               // 5 1 = Reserved bit
                               // 6 1 = Error flag(error frame or invalid frame)
                               // 7 Reserved
                               // 15 1 = Async.monitoring has generated this event
                               // 16 1 = Event is a PDU
                               // 17 Valid for PDUs only.The bit is set if the PDU is
```

```
                                    valid(either if the PDU has no update bit, or the update
                                    bit for the PDU was set in the received frame).
                                // 18 Reserved
                                // 19 1 = Raw frame(only valid if PDUs are used in the
                                    configuration).A raw frame may contain PDUs in its
                                    payload
                                // 20 1 = Dynamic segment  0 = Static segment
                                // 21 This flag is only valid for frames and not for PDUs.
                                    1 = The PDUs in the payload of this frame are logged in
                                    separate logging entries. 0 = The PDUs in the payload of
                                    this frame must be extracted out of this frame.The
                                    logging file does not contain separate  // PDU-entries.
                                // 22 Valid for PDUs only.The bit is set if the PDU has an
                                    update bit
    u32 FFrameCRC;              // frame crc
    u64 FReserved1;             // 8 reserved bytes
    u64 FReserved2;             // 8 reserved bytes
    u64 FTimeUs;                // timestamp in us
    u8  FData[254];             // 254 data bytes
} TFlexRay, *PFlexRay;
```

成员：

FIdxChn:通道索引
FChannelMask：通道掩码
FDir：
FPayloadLength：有效负载长度(字节)
FActualPayloadLength：实际数据字节
FCycleNumber：周期数
FCCType：
FFrameType：报文类型
FHeaderCRCA：CRCA 标头
FHeaderCRCB：CRCB 标头
FFrameStateInfo：帧状态信息
FSlotId：静态段
FFrameFlags：22 位帧标志
FFrameCRC:帧 CRC
FReserved1：8 个预留字节
FReserved2：8 个预留字节
FTimeUs:时间戳(微妙)
FData:254 数据字节

构造函数

本结构体没有构造函数，使用函数 init_w_slot_id 初始化报文结构，其中包括参数
ASlotId：flexray 报文 ID;

ADLC：报文长度；

属性：

is_tx:true: 报文是发送出去的报文；false：报文是接收回来的报文
is_null:是否为空
is_data:是否为数据帧
is_sync:是否异步发送
is_startup：是否启动
is_err:是否错误帧

调用示例：

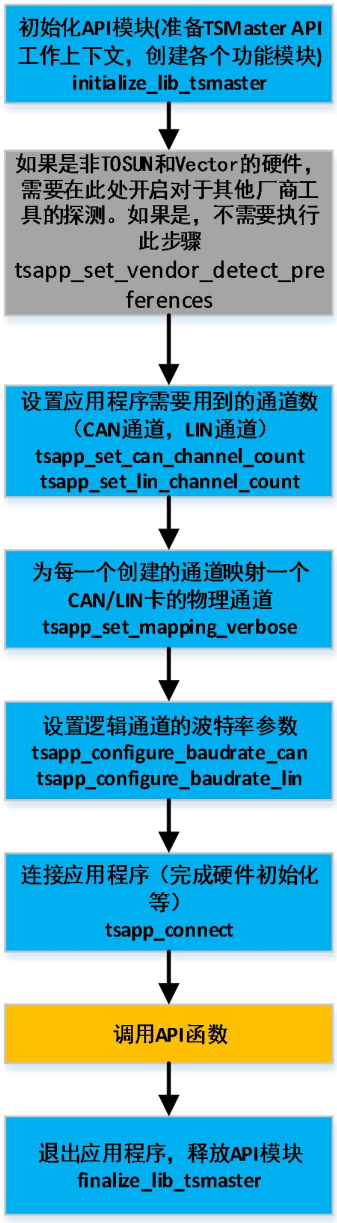
```
TFlexRay flexray_frame;
flexray_frame.init_w_slot_id(0x11, 8); //初始化PID为0x11  报文长度8
flexray_frame.property__set_is_tx(true);//设置为发送报文
for (u32 i = 0; i < 254; i++) {
    flexray_frame.FData[i] = 0;
}
```

6. 设备初始化

1. CAN 初始化流

程图：

在实现 CAN 设备通讯之前，首先要进行设备初始化。TSMaster 设备初始化包括以下步骤：



注意：

- 在使用 API 模块之前，需要调用 initialize_lib_tsmaster 函数创建该模块；在使用过后，需要调用 finalize_lib_tsmaster 释放该模块。这两个函数一定是成对出现的。
- 第二步中，如果用户使用的是英特佩斯，Peak，Kavsar，ZLG 等硬件，需要在这一步开启对这些工具的探测，否则程序无法查询到这些硬件。

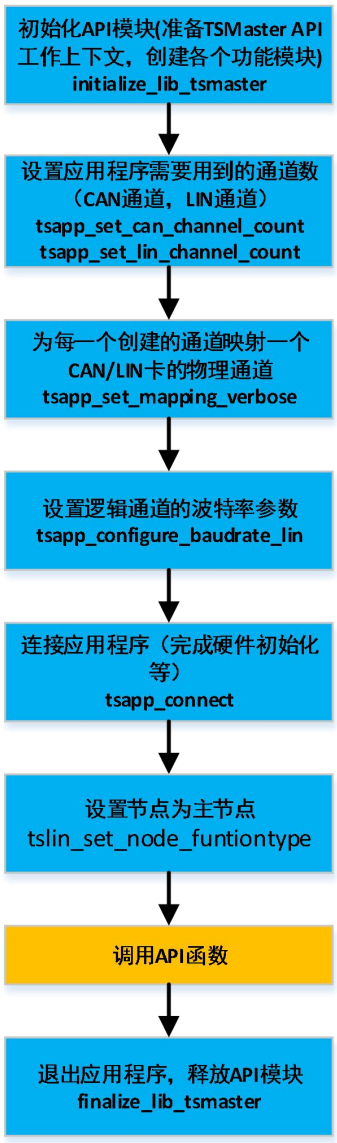
CAN 初始化示例代码：

```
//第一步：初始化API模块
    if (!CheckOK(initialize_lib_tsmaster("TSMasterCDemo"))) return vErrorCode;
    std::cout << "Application initialized: TSMasterCDemo\n";
    if (!CheckOK(tsapp_set_logger(TSMasterLogger))) return vErrorCode;
    std::cout << "TSMaster logger redirected to the current program\n";
    //第二步：根据应用程序需要设置CAN，LIN通道数量，比如，这里需要2个CAN通道。
     if (!CheckOK(tsapp_set_can_channel_count(2))) break;
     std::cout << "TSMasterCDemo has 2 CAN channels\n";//第三步：按需创建通道映射:
    //把TC1005板卡的硬件通道1映射到驱动的逻辑通道1上面
    TLIBTSMapping m;
    m.init();
    sprintf_s(m.FAppName, "%s", "TSMasterCDemo");
    sprintf_s(m.FHWDeviceName, "%s", "TC1005");
    m.FAppChannelIndex = CH1;
    m.FAppChannelType = APP_CAN;
    m.FHWDeviceType = TS_TC1005_DEVICE;
    m.FHWDeviceSubType = -1;            // TC1005 has no series
    m.FHWIndex = 0;                     // the first hardware
    m.FHWChannelIndex = CH1;            // channel 1/5 of TC1005
    if (!CheckOK(tsapp_set_mapping(&m))) break;
            std::cout << "TSMasterCDemo logical CH1 mapped\n";
    m.FAppChannelIndex = CH2;
    m.FHWChannelIndex = CH2;            // channel 2/5 of TC1005
    if (!CheckOK(tsapp_set_mapping(&m))) break;
            std::cout << "TSMasterCDemo logical CH2 mapped\n";
    //第四步：初始化通道参数
    if (!CheckOK(tsapp_configure_baudrate_can(CH1, 500.0, false, true))) break;
        std::cout << "TSMasterCDemo logical CH1 baudrate set to 500Kbps, with 120Ohm term.
resistor\n";
    if (!CheckOK(tsapp_configure_baudrate_can(CH2, 500.0, false, true))) break;
        std::cout << "TSMasterCDemo logical CH2 baudrate set to 500Kbps, with 120Ohm term.
resistor\n";
    //第五步：连接application：连接硬件通道并开启接收FIFO
    if (!CheckOK(tsapp_connect())) break;
    tsfifo_enable_receive_fifo();  //使能接收缓冲区
    std::cout << "TSMasterCDemo application connected, communication starts here...\n";
```

2. LIN 初始化

流程图：

LIN 通道初始化示例流程图如下：



代码示例：

下述代码为初始化为主节点的代码示例：

```
void CreateLINApplicationDemo()
{
            //FProgramName:唯一名称，后面的各种映射跟他绑定
           //第一步：初始化API模块,如果已经调用，这里则不需要调用
            initialize_lib_tsmaster(FProgramName);
            //第二步：按需设置需要的通道数，比如，这里需要0个CAN通道，1个LIN通道
            if (tsapp_set_can_channel_count(0) == 0)
            {
                Log("Set CAN Channel Count Success!");
            }
            else
                Log("Set CAN Channel Count Failed!");
            if (tsapp_set_ lin_channel_count(0) == 0)
```

```
            {
                Log("Set LIN Channel Count Success!");
            }
            else
                Log("Set LIN Channel Count Failed");
            //第三步：按需创建通道映射:
            if (tsapp_set_mapping_verbose(FProgramName,
TLIBApplicationChannelType.APP_LIN,
                  CH1, "TL1001", TLIBBusToolDeviceType.TS_USB_DEVICE,
(int)TLIB_TS_Device_Sub_Type.TL1001, 0, HARDWARE_CHANNEL.CHN1) == 0)
            {
            }
            //第四步：初始化通道参数:设置LIN通道1的波特率为19.2k
            if (tsapp_configure_baudrate_lin((int)CH1, 19.2) == 0)
            {
                Log("LIN Channel " + (1).ToString() + " baudrate has been configured");
            }
            else
            {
                Log("LIN Channel " +  (1).ToString() + " baudrate failed");
            }
            //第五步：连接application：连接硬件通道并开启接收FIFO
            string connectResult =
            tsapp_get_error_description(TsMasterApi.tsapp_connect());
            if (connectResult == "OK")
            {
                Log("Connect Application Success!");
                tsapp_enable_receive_fifo();
                Log("Start Receive FIFO!");  //如果不使能内部FIFO，无法使用Receive函数读
取内部报文
            }
            else
            {
                Log(connectResult);
                Log("Connect Application Failed! Please check the mapping table and whether
the Hardware is Ready?!");
            }
            //设置LIN模块工作在主节点
            if (tslin_set_node_funtiontype(CH1, TLINNodeType.T_MasterNode) == 0x00)
            {
                Log("Set LIN Node Master Mode Success!");
            }
            else
            {
```

```
            Log("Set LIN Node Master Mode Failed!");
        }
}
```

7. 报文发送

1. CAN 报文发送

```
TCAN can_frame;//创建一个报文结构体
can_frame.init_w_std_id(0x11, 8);//初始化ID为0x11,报文DLC为8
can_frame.set_data(0, 1, 2, 3, 4, 5, 6, 7);//设置报文数据
```

单帧异步发送：

```
tsapp_transmit_can_async(can_frame)
```

单帧同步发送：

```
# 100 表示超时参数
tsapp_transmit_can_sync(can_frame,100)
```

周期发送：

```
# 100ms 周期发送 ACAN 报文
tsapp_add_cyclie_msg_can(can_frame,100)
```

删除周期发送：

```
# 删除周期发送 ACAN 报文
tsapp_delete_cyclie_msg_can(can_frame)
```

2. CANFD 报文发送

```
TCANFD can_frame;
can_frame.init_w_std_id(0x11, 0xf);
```

单帧异步发送：

```
tsapp_transmit_canfd_async(can_frame);
```

单帧同步发送：

```
# 100 表示超时参数
tsapp_transmit_canfd_sync(can_frame,100)
```

周期发送：

```
# 100ms 周期发送 ACAN 报文
tsapp_add_cyclie_msg_canfd(can_frame,100)
```

删除周期发送：

```
# 删除周期发送 ACAN 报文
tsapp_delete_cyclie_msg_canfd(can_frame)
```

3. LIN 报文发送

```
TLIN lin_frame;
lin_frame.init_w_id(0x11, 8); //初始化PID为0x11  报文长度8
lin_frame.property__set_is_tx(true);//设置为发送报文
for (u32 i = 0; i < 8; i++) {
    flexRay_frame.FData[i] = 0;
}
```

单帧异步发送：

```
tsapp_transmit_lin_async(lin_frame);
```

单帧同步发送：

```
# 100 表示超时参数
tsapp_transmit_lin_sync(lin_frame,100);
```

4. Flexray 报文发送

```
TFlexRay flexRay_frame;
flexRay_frame.init_w_slot_id(0x11, 8); //初始化PID为0x11  报文长度8
flexRay_frame.property__set_is_tx(true);//设置为发送报文
for (u32 i = 0; i < 254; i++) {
    flexRay_frame.FData[i] = 0;
}
```

单帧异步发送：

```
tsapp_transmit_flexray_async( flexRay_frame);
```

单帧同步发送：

```
# 100 表示超时参数
tsapp_transmit_flexray_sync( flexRay_frame, 100);
```

8. 报文接收

1. 回调函数方式：

简介：

设备接收到报文过后，把报文整理成标准的 TCAN/TCANFD 数据结构。然后调用用户注册的接收回调函数，通过参数把接收到的报文传递给调用者。TSCAN 内部维护一个独立的线程，

每当消息达到后，就会通过回调函数主动把数据传递给调用者，用户不需要主动去调用读取函数。

注册回调函数：

采用函数 tsapp_register_event_can、tsapp_register_event_canfd、tsapp_register_event_lin、tsapp_register_event_flexray 注册报文接收回调函数。回调函数中报文处理结构体参考其它章节相关内容。

回调函数使用：

```
//注：在回调事件中，尽量只做数值变换操作，避免耗时操作
```

CAN 回调：

```
void OnCANEvent(const ps32 AObj, const PCAN ACAN)
{
    //处理收到的报文数据 AData
}
s32 obj;
```

使用回调函数时：

```
tsapp_register_event_can(&obj,OnCANEvent);
```

不使用回调函数时：

```
tsapp_unregister_event_can(&obj, OnCANEvent);
```

CANFD 回调：

```
void OnCANFDEvent(const ps32 AObj, const PCANFD ACANFD)
{
    //处理收到的报文数据 AData
}
s32 obj;
```

使用回调函数时：

```
tsapp_register_event_canfd(&obj,OnCANFDEvent);
```

不使用回调函数时：

```
tsapp_unregister_event_canfd(&obj, OnCANFDEvent);
```

LIN 回调：

```
void OnLINEvent(const ps32 AObj, const PLIN ALIN)
{
    //处理收到的报文数据 AData
}
s32 obj;
```

使用回调函数时：

```
tsapp_register_event_lin(&obj,OnLINEvent);
```

不使用回调函数时：

```
tsapp_unregister_event_lin(&obj, OnLINEvent);
```

Flexray 回调：

```
void OnFlexrayEvent(const ps32 AObj, const PFlexRay AFlexRay)
{
    //处理收到的报文数据 AData
```

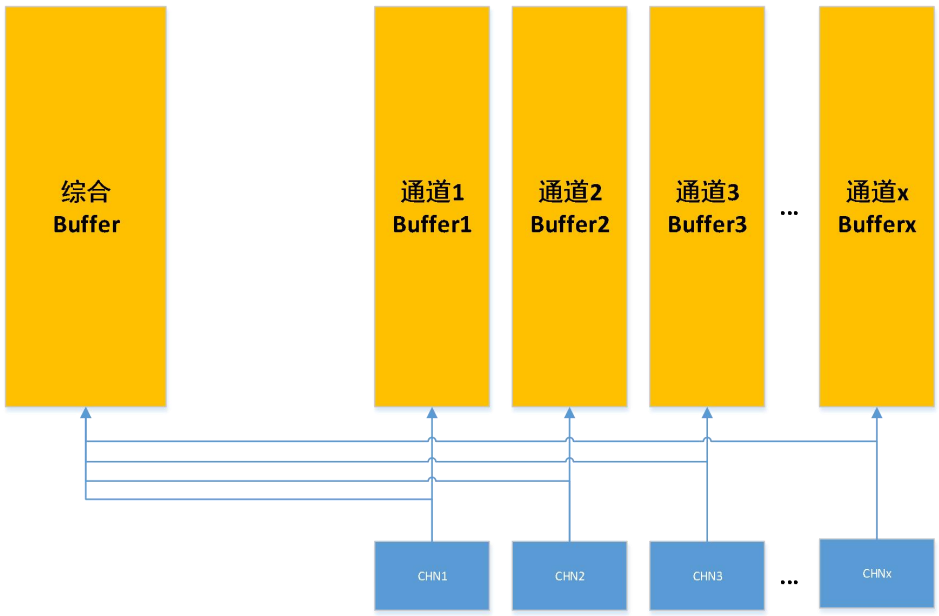
```
}
s32 obj;
使用回调函数时：
tsapp_register_event_flexray(&obj,OnFlexrayEvent);
不使用回调函数时：
tsapp_unregister_event_flexray(&obj, OnFlexrayEvent);
```

注：可以看到上述各总线回调的注册不同地方在 ONXXEvent 不同，使用注册函数不同

2. 读取设备消息缓存的方式：

简介：

设备接收到报文过后，缓存在设备内部的 FIFO 中，外部程序调用函数接口从设备 FIFO 中把报文读取出来，FIFO 指针往后面移动；如果调用者一直不主动读取，会造成驱动内部 FIFO 溢出，最新的报文覆盖最旧的报文。TSMaster API 内部，报文缓存机制如下图所示：



综合FIFO：所有通道的报文根据接收顺序放在里面。
特点：
可以看到不同通道报文的相对接收顺序。报文在里面按照接收顺序存放，最新的报文覆盖最旧接收的报文。

通道FIFO：每一个通道有一个自己单独的报文FIFO
特点：
专用于存储跟本通道相关的报文,通道之间互不干扰。报文在里面按照接收顺序存放，最新的报文覆盖最旧接收的报文。

TSMaster 提供了报文读取驱动。可以选择读取指定通道的报文集合，也可以读取综合报文里面的报文集合。参考章节 10 中 tsfifo 相关函数

CAN 报文获取

获取 Rx 报文：

```
TCAN canBuffer[100] ;
```

```
    S32 revCnt    = 100;          #revCnt 的大小为 TCANBuffer 的长度，可以小，当一定不能比 TCANBuffer 大
    #注：每次传入 tsfifo_receive_can_msgs 的 revCnt 需要重新赋值，因为该变量为一个输入输出量，返回读到的报文数量，如果存在没有读到的情况，该变量会变为 0，此时再往里面传入，将一直读不到数据
tsfifo_receive_can_msgs(
        canBuffer,
        &revCnt,
        CH1,                    //读取通道1的报文数据
        1);  //接收TX/RX所有报文，如果只读取接收端的报文，则修改为 READ_TX_RX_DEF.ONLY_RX_MESSAGES
```

获取 Rx Tx 报文：

```
    TCAN canBuffer[100] ;
    S32 revCnt    = 100;          //revCnt 的大小为 TCANBuffer 的长度，可以小，当一定不能比 TCANBuffer 大
    #注：每次传入 tsfifo_receive_can_msgs 的 revCnt 需要重新赋值，因为该变量为一个输入输出量，返回读到的报文数量，如果存在没有读到的情况，该变量会变为 0，此时再往里面传入，将一直读不到数据
tsfifo_receive_can_msgs(
        canBuffer,
        &revCnt,
        CH1,                    //读取通道1的报文数据
        1);  //接收TX/RX所有报文，如果只读取接收端的报文，则修改为 READ_TX_RX_DEF.ONLY_RX_MESSAGES
```

获取 fifo 报文数量：

```
#读取通道 0 fifo 的报文数量
S32 ACount = 0;
tsfifo_read_can_buffer_frame_count(0,ACount);
```

获取 fifo Tx 报文数量：

```
#读取通道 0 fifo Tx 的报文数量
s32 ACount = 0;
tsfifo_read_can_tx_buffer_frame_count(0,ACount);
```

获取 fifo Rx 报文数量：

```
#读取通道 0 fifo Rx 的报文数量
s32 ACount = 0;
tsfifo_read_can_rx_buffer_frame_count(0,ACount);
```

清空 fifo 报文：

```
#读取通道 0 fifo Rx 的报文数量
tsfifo_clear_can_receive_buffers(0)
```

CANFD 报文获取

注：CANFD 向下包含 CAN，因此 CANFD 报文获取，是会包含 CAN 数据

获取 Rx 报文：

```
TCANFD canfdBuffer[100];
s32 revCnt = 0;          //buffersize 的大小为 TCANFDBuffer 的长度，可以小，当一定不能比 TCANBuffer 大
//注：每次传入 tsfifo_receive_can_msgs 的 revCnt 需要重新赋值，因为该变量为一个输入输出量，返回读到的报文数量，如果存在没有读到的情况，该变量会变为 0，此时再往里面传入，将一直读不到数据
tsfifo_receive_canfd_msgs(
        canfdBuffer,
        &revCnt,
        CH1,                //读取通道1的报文数据
        READ_TX_RX_DEF.ONLY_RX_MESSAGES);
```

获取 Rx Tx 报文：

```
TCANFD canfdBuffer[100];
s32 revCnt = 0;          #revCnt 的大小为 TCANFDBuffer 的长度，可以小，当一定不能比 TCANBuffer 大
#注：每次传入 tsfifo_receive_can_msgs 的 revCnt 需要重新赋值，因为该变量为一个输入输出量，返回读到的报文数量，如果存在没有读到的情况，该变量会变为 0，此时再往里面传入，将一直读不到数据
tsfifo_receive_canfd_msgs(
        canfdBuffer,
        &revCnt,
        CH1,                //读取通道1的报文数据
        READ_TX_RX_DEF.ONLY_RX_MESSAGES);
```

获取 fifo 报文数量：

```
#读取通道 0 fifo 的报文数量
s32 ACount = 0;
tsfifo_read_canfd_buffer_frame_count(0,ACount);
```

获取 fifo Tx 报文数量：

```
#读取通道 0 fifo Tx 的报文数量
s32 ACount = 0;
tsfifo_read_canfd_tx_buffer_frame_count(0,ACount);
```

获取 fifo Rx 报文数量：

```
#读取通道 0 fifo Rx 的报文数量
```

```
s32 ACount = 0;
tsfifo_read_canfd_rx_buffer_frame_count(0,ACount);
```

清空 fifo 报文：

```
#读取通道 0 fifo Rx 的报文数量
tsfifo_clear_canfd_receive_buffers(0);
```

LIN 报文获取

获取 Rx 报文：

```
TLIN linBuffer[100];
s32 revCnt = 100;          #revCnt 的大小为 TLINBuffer 的长度，可以小，当一定不能比 TCANBuffer 大
#注：每次传入 tsfifo_receive_can_msgs 的 revCnt 需要重新赋值，因为该变量为一个输入输出量，返回读到的报文数量，如果存在没有读到的情况，该变量会变为 0，此时再往里面传入，将一直读不到数据
tsfifo_receive_lin_msgs(
        linBuffer,
        &revCnt,
        CH1,                    //读取通道1的报文数据
        READ_TX_RX_DEF.ONLY_RX_MESSAGES);
```

获取 Rx Tx 报文：

```
TLIN linBuffer[100];
s32 revCnt = 100;          #revCnt 的大小为 TLINBuffer 的长度，可以小，当一定不能比 TCANBuffer 大
#注：每次传入 tsfifo_receive_can_msgs 的 revCnt 需要重新赋值，因为该变量为一个输入输出量，返回读到的报文数量，如果存在没有读到的情况，该变量会变为 0，此时再往里面传入，将一直读不到数据
tsfifo_receive_lin_msgs(
        linBuffer,
        &revCnt,
        CH1,                    //读取通道1的报文数据
        READ_TX_RX_DEF.ONLY_RX_MESSAGES);
```

获取 fifo 报文数量：

```
#读取通道 0 fifo 的报文数量
s32 ACount = 0;
tsfifo_read_lin_buffer_frame_count(0,ACount);
```

获取 fifo Tx 报文数量：

```
#读取通道 0 fifo Tx 的报文数量
s32c ACount = 0;
tsfifo_read_lin_tx_buffer_frame_count(0,ACount);
```

获取 fifo Rx 报文数量：

#读取通道 0 fifo Rx 的报文数量
s32 ACount = 0;
tsfifo_read_lin_rx_buffer_frame_count(0,ACount);

清空 fifo 报文：

#读取通道 0 fifo Rx 的报文数量
tsfifo_clear_lin_receive_buffers(0);

Flexray 报文获取

获取 Rx 报文：

TFlexray FlexrayBuffer[100];
int revCnt = sizeof(FlexrayBuffer)/sizeof(FlexrayBuffer[0]); //revCnt的大小为FlexrayBuffer的长度，可以小，当一定不能比FlexrayBuffer大
//注：每次传入 tsfifo_receive_can_msgs 的 revCnt 需要重新赋值，因为该变量为一个输入输出量，返回读到的报文数量，如果存在没有读到的情况，该变量会变为 0，此时再往里面传入，将一直读不到数据

```
tsfifo_receive_flexray_msgs(
        FlexrayBuffer,
        &revCnt,
        CH1,                    //读取通道1的报文数据
        READ_TX_RX_DEF.ONLY_RX_MESSAGES);
```

获取 Rx Tx 报文：

TFlexray FlexrayBuffer[100];
int revCnt = sizeof(FlexrayBuffer)/sizeof(FlexrayBuffer[0]); //revCnt 的大小为 FlexrayBuffer 的长度，可以小，当一定不能比 FlexrayBuffer 大
#注：每次传入 tsfifo_receive_can_msgs 的 revCnt 需要重新赋值，因为该变量为一个输入输出量，返回读到的报文数量，如果存在没有读到的情况，该变量会变为 0，此时再往里面传入，将一直读不到数据

```
tsfifo_receive_flexray_msgs(
        FlexrayBuffer,
        &revCnt,
        CH1,                    //读取通道1的报文数据
        READ_TX_RX_DEF.ONLY_RX_MESSAGES);
```

获取 fifo 报文数量：

#读取通道 0 fifo 的报文数量
s32 ACount =0;
tsfifo_read_flexray_buffer_frame_count(0,ACount);

获取 fifo Tx 报文数量：

```
#读取通道 0 fifo Tx 的报文数量
s32 ACount = 0;
tsfifo_read_flexray_tx_buffer_frame_count(0,ACount);
```

获取 fifo Rx 报文数量：

```
#读取通道 0 fifo Rx 的报文数量
s32 ACount = 0;
tsfifo_read_flexray_rx_buffer_frame_count(0,ACount);
```

清空 fifo 报文：

```
#读取通道 0 fifo Rx 的报文数量
tsfifo_clear_flexray_receive_buffers(0);
```

9. 接口函数介绍

1. initialize_lib_tsmaster

函数名称	void initialize_lib_tsmaster(const char* AAppName);
功能介绍	初始化 TSMasterAPI 模块
调用位置	在使用 TSMasterAPI 之前，必须先执行模块初始化函数，初始化内部参数，为驱动的运行准备好环境。
输入参数	AAppName:应用程序名称，不能为空。硬件通道参数等都跟此名称绑定。
返回值	无
示例	initialize_lib_tsmaster(&FProgramName);

2. initialize_lib_tsmaster_with_project

函数名称	void initialize_lib_tsmaster_with_project(const char* AAppName,const char* projectfile);
功能介绍	初始化 TSMasterAPI 模块，为程序调用准备好运行环境
调用位置	在使用 TSMasterAPI 之前，必须先执行模块初始化函数，初始化内部参数，为驱动的运行准备好环境。
输入参数	AAppName:应用程序名称，不能为空。硬件通道参数等都跟此名称绑定。
返回值	无
示例	Const char* AAppName = “TSMaster”; Const char* projectfile = “”; initialize_lib_tsmaster_with_project(FProgramName,projectfile);

3. finalize_lib_tsmaster

函数名称	void finalize_lib_tsmaster();
功能介绍	释放 TSMasterAPI 模块
调用位置	在退出程序之前，释放掉 TSMasterAPI 所使用的资源。
输入参数	无

返回值	无
示例	finalize_lib_tsmaster();

4. tsapp_get_error_description

函数名称	int tsapp_get_error_description(const s32 ACode, char** ADesc)
功能介绍	根据函数执行的返回值编码 ACode，查询该编码代表的意义
调用位置	TSCAN 的 API 函数执行过后，会返回一个 ErroCode 编码，通过调用此函数可以查询该编码代表的具体含义
输入参数	ACode：错误编码值 ADesc：转换结果指针所在内存地址
返回值	错误编码代表具体含义字符串

示例	bool CheckOK(const int ACode)
	{
	char* desc;
	vErrorCode = ACode;
	if (0 == ACode) return true;
	if (0 == tsapp_get_error_description(ACode, &desc)) {
	std::cout << "API error: " << desc << "\n";
	} else {
	std::cout << "API error code: " << ACode << "\n";
	}
	return false;
	}

5. tsapp_enumerate_hw_devices

函数名称	int tsapp_enumerate_hw_devices(const ps32 ACount);
功能介绍	枚举当前插在电脑上的能够使用的 CAN 设备数量
调用位置	这个函数必须在 Application 没有连接（Connect）之前使用。因为在连接状态下调用此函数的话，会影响一些设备的 USB 通讯。
输入参数	ACount:　传入out参数，枚举到的当前设备数量
返回值	==0：枚举设备成功 其他值：枚举设备失败
示例	tsapp_enumerate_hw_devices(out hardwareNum); 获取当前 PC 端可以使用的设备数量，获取值存放在 hardwareNum 变量中。

6. tsapp_get_hw_info_by_index

函数名称	int tsapp_get_hw_info_by_index(int AIndex, const PLIBHWInfo AHWInfo);
功能介绍	根据索引值获取硬件设备的信息
调用位置	在需要查询硬件设备信息的场合。同星，Vector 等设备都有自己的唯一信

	息，用户应用程序可以读取此唯一信息，用于自己应用程序进行权限管理，加密等场合。
输入参数	AIndex:该设备的索引值 AHWInfo:设备的详细信息存放在此结构体中
返回值	==0：注册成功 其他值：注册失败
示例	TLIBHWInfo info; tsapp_get_hw_info_by_index(0, &info);

7. tsapp_get_hw_info_by_index_verbose

函数名称	int tsapp_get_hw_info_by_index_verbose(const s32 AIndex, PLIBBusToolDeviceType ADeviceType, char* AVendorNameBuffer, //array[0..31] of AnsiChar; s32 AVendorNameBufferSize, char* ADeviceNameBuffer, //array[0..31] of AnsiChar; s32 ADeviceNameBufferSize, char* ASerialStringBuffer, //array[0..63] of AnsiChar s32 ASerialStringBufferSize);
功能介绍	根据设备索引值获取设备信息。跟 tsapp_get_hw_info_by_index 函数相比，获取的信息直接存储在输入的数组中，用户根据自己习惯调用。
调用位置	在需要查询硬件设备信息的场合。同星，Vector 等设备都有自己的唯一信息，用户应用程序可以读取此唯一信息，用于自己应用程序进行权限管理，加密等场合。
输入参数	AIndex:设备索引值 ADeviceType：设备类型：CAN/LIN AVendorNameBuffer ：存放供应商名称的数组 AVendorNameBufferSize：存放供应商名称的数组大小 ADeviceNameBuffer：存放设备名称的数组 ADeviceNameBufferSize：存放设备名称的数组大小 ASerialStringBuffer：存放设备串行编号数组 ASerialStringBufferSize：存放设备串行编号数组大小
返回值	==0：注册成功 其他值：注册失败
示例	

8. tsapp_set_can_channel_count

函数名称	int tsapp_set_can_channel_count(const s32 ACount);

功能介绍	设置应用程序用到的 CAN 通道数量。可以通过 TSMaster 界面直接完成。
调用位置	映射 CAN 通道之前，用户可以根据应用程序的需求，分配需要用到的 CAN 通道数量。
输入参数	ACount：需要设置的CAN通道数量
返回值	==0:设置成功 其他：失败，根据错误码查询错误类型。

示例	//设置当前应用程序用到4个CAN通道
	if (tsapp_set_can_channel_count(4) == 0)
	Log("Set CAN Channel Count Success!");
	else
	Log("Set CAN Channel Count Failed!");

9. tsapp_set_lin_channel_count

函数名称	int tsapp_set_lin_channel_count(int ACount);
功能介绍	设置应用程序用到的通道数量。可以通过 TSMaster 界面直接完成。
调用位置	映射 LIN 通道之前，用户可以根据应用程序的需求，分配需要用到的 LIN 通道数量。
输入参数	ACount：需要设置的 LIN 通道数量
返回值	==0:设置成功 其他：失败，根据错误码查询错误类型

示例	//设置当前应该程序使用0个LIN通道
	if (tsapp_set_lin_channel_count(0) == 0)
	Log("Set LIN Channel Count Success!");
	else
	Log("Set LIN Channel Count Failed!");

10. tsapp_set_channel_mapping_verbose

函数名称	int tsapp_set_mapping_verbose(const char* AAppName, const TLIBApplicationChannelType AAppChannelType, const s32 AAppChannel, const char* AHardwareName, const TLIBBusToolDeviceType AHardwareType, const s32 AHardwareSubType, const s32 AHardwareIndex, const s32 AHardwareChannel, const bool AEnableMapping);
功能介绍	使用硬件设备之前，为应用程序的通道（CAN/CANFD/LIN）映射（也就是绑定）指定 CAN 设备的指定通道。这个操作也可以在 TSMaster 的设备管理界面完成，代码中可以不需要这部分操作。关于通道映射函数，还有不清晰的地方，请查看文档：TSMasterAPI

	HardwareMap.pdf
调用位置	在应用程序连接 CAN 工具之前，调用此函数完成硬件的绑定。

输入参数	AAppName:应用程序名称：该 Map 绑定到的应用程序名称，跟 initialize_lib_tsmaster 中名称要一致
	AAppChannelType:通道类型,包括 APP_CAN, APP_LIN</param>
	AAppChannel 应用程序的通道编号,这是上层程序使用的通道编号</param>
	AHardwareName:硬件名称,比如 TOSUN,Vector 等,也可以不填写,主要是方便用户查看</param>
	AHardwareType:硬件类型,根据 TLIBBusToolDeviceType 囊括了目前市面上所有主流的 CAN 卡硬件，</param>. 其中，工具枚举类型定义如下：
	typedef enum {
	BUS_UNKNOWN_TYPE = 0,
	TS_TCP_DEVICE = 1,
	XL_USB_DEVICE = 2,
	TS_USB_DEVICE = 3,
	PEAK_USB_DEVICE = 4,
	KVASER_USB_DEVICE = 5,
	ZLG_USB_DEVICE = 6,
	ICS_USB_DEVICE = 7,
	TS_TC1005_DEVICE = 8
	} TLIBBusToolDeviceType, *PLIBBusToolDeviceType;/// <param name="AHardwareSubType">
	在该设备类型下面的子设备，比如同星的 TS_USB_DEVICE，下面就包含 TC1001，TL1002，TC1011 等等系列产品
	AHardwareIndex:支持同时插多个完全一样的硬件，比如同时插两个 TC1001，这里用于选择是 1 号设备的通道还是 2 号设备的通道
	AHardwareChannel:硬件设备的通道，比如 TC1005 有 5 个硬件通道，这个参数就用于选择对应的硬件通道
	AEnableMapping:是否使能此映射，如果设置为 false，则该通道是不能使用的

返回值	==0:执行成功 其他：查询错误码
示例	参考章节 2 示例工程

11. tsapp_get_mapping

函数名称	int tsapp_get_mapping (const PLIBTSMapping AMapping)
功能介绍	获取指定通道的映射信息值
调用位置	想查询应用程序指定通道当前绑定的硬件信息等内容的时候，调用此函数
输入参数	AMapping：指向映射结构的指针
返回值	==0:执行成功 其他：查询错误码

示例	TLIBTSMapping m;
	m.init();
	// just set the following 3 fields to retrieve mapping info:
	sprintf(m.FAppName, "TSMaster"); // Application is TSMaster

```
        m.FAppChannelIndex = 0;            // Channel 1
        m.FAppChannelType = APP_CAN;       // Application channel type can be APP_CAN or APP_LIN
        if (0 == app.get_mapping(&m)){
          log("application CAN channel 1 mapping has been retrieved: ");
          log("FHWDeviceType = %d, FHWIndex = %d, FHWChannelIndex = %d, FHWDeviceSubType = %d, FHWDeviceName = %s", m.FHWDeviceType, m.FHWIndex, m.FHWChannelIndex, m.FHWDeviceSubType, m.FHWDeviceName);
        }
```

12. tsapp_get_mapping_verbose

函数名称	int tsapp_get_mapping_verbose(const char* AAppName, const TLIBApplicationChannelType AAppChannelType, const s32 AAppChannel, const PLIBTSMapping AMapping);
功能介绍	获取映射信息，设置应用程序通道和硬件的绑定。
调用位置	如果想配置硬件和应用程序通道的绑定，调用此函数配置映射信息。
输入参数	AAppName:应用程序名称 AAppChannelType:通道类型，类型定义如下： AAppChannel:应用程序通道编号 AMapping:映射信息结构体指针
返回值	==0:连接成功 其他：查询错误码
示例	TLIBTSMapping m; tsapp_get_mapping_verbose("TSMaster",APP_CAN, 0,&m);

13. tsapp_del_mapping_verbose

函数名称	int tsapp_del_mapping_verbose(const char* AAppName, const TLIBApplicationChannelType AAppChannelType, const s32 AAppChannel);
功能介绍	删除映射信息，解除应用程序通道和硬件的绑定。
调用位置	如果想接触硬件和应用程序通道的绑定，调用此函数删除映射信息，解除配置信息即可。

输入参数	AAppName:应用程序名称
	AAppChannelType:通道类型，类型定义如下：
	public enum TLIBApplicationChannelType : int
	{
	APP_CAN = 0,
	APP_LIN = 1
	};
	AAppChannel:应用程序通道编号

返回值	==0:连接成功 其他：查询错误码

示例	TLIBTSMapping m;
	m.init();
	// just set the following 3 fields to delete the mapping info:
	sprintf(m.FAppName, "TSMaster"); // Application is TSMaster
	m.FAppChannelIndex = 0; // Channel 1
	m.FAppChannelType = APP_CAN; // Application channel type can be APP_CAN or APP_LIN
	if (0 == tsapp_del_mapping_verbose(&m)){
	log("application CAN channel 1 mapping has been deleted");
	}

14. tsapp_configure_baudrate_can

函数名称	int tsapp_configure_baudrate_can(const s32 AIdxChn, const float ABaudrateKbps, const bool AListenOnly, const bool AInstallTermResistor120Ohm);
功能介绍	设置 CAN 通道的波特率等参数
调用位置	连接硬件设备之前，先设置通道的参数
输入参数	AIdxChn:应用程序通道编号 ABaudrateKbps:波特率（kbps） AListenOnly:是否只听模式，如果开启，则只能接收数据 AInstallTermResistor120Ohm:是否只能设备内部终端电阻
返回值	==0:连接成功 其他：检查错误码

示例	// set CAN baudrate to 500 Kbps on CAN channel 1, 120 Ohm terminator active
	if (0 == tsapp_configure_baudrate_can (0, 500, false, true)){
	log("CAN baud-rate on channel 1 has been set to 500 Kbps");
	}

15. tsapp_configure_baudrate_canfd

函数名称	int tsapp_configure_baudrate_canfd(const s32 AIdxChn, const float ABaudrateArbKbps, const float ABaudrateDataKbps, const TCANFDControllerType AControllerType, const TCANFDControllerMode AControllerMode, const bool AInstallTermResistor120Ohm);
功能介绍	配置 CANFD 通道的波特率等硬件参数
调用位置	连接 CAN 工具之前，先调用此函数配置硬件设备参数
输入参数	AIdxChn:应用程序通道 AArbRateKbps:仲裁场波特率 ADataRateKbps:数据场波特率 AControllerType:控制器类型，包括： 经典 CAN(lfdtCAN = 0),ISOCANFD(lfdtISOCAN = 1),NoISOCANFD(lfdtNonISOCAN = 2) AControllerMode:控制器工作模式,包括: 正常工作模式(lfdmNormal = 0),关闭 ACK 模式(lfdmACKOff = 1),受限制模式(lfdmRestricted = 2)

	AInstallTermResistor120Ohm:是否使能内部终端电阻
返回值	==0:连接成功 其他：检查错误码

示例	// configure ISO CAN FD controller on channel 1:
	// normal mode, 500 Kbps in arbitration field, 2000 Kbps in data field, termination resistor is active
	if (0 == tsapp_configure_baudrate_canfd (0, 500, 2000, fdtISOCANFD, fdmNormal, true)){
	log("CAN FD controller on channel 1 has been configured");
	}

16. tsfifo_enable_receive_fifo

函数名称	void tsfifo_enable_receive_fifo();
功能介绍	**使能设备内部缓存接收模式**
调用位置	应用程序如果想通过读取缓存的方式读取数据，在执行 tsfifo_receive_can/canfd/lin_message_list 之前，必须要执行此函数。
输入参数	无
返回值	无
示例	tsfifo_enable_receive_fifo();

17. tsfifo_disable_receive_fifo

函数名称	void tsfifo_disable_receive_fifo();
功能介绍	**关闭驱动内部的接收缓存机制**
调用位置	用户不需要采用读取缓存的方式接收报文的时候，调用此函数关闭内部缓存。
输入参数	无
返回值	无
示例	tsfifo_disable_receive_fifo();

18. tsapp_connect

函数名称	int tsapp_connect();
功能介绍	**连接应用程序。本函数执行的时候，会检查映射的硬件通道是否全部就绪，设置各个硬件参数，并完成设备和应用程序的连接。**
调用位置	完成各个硬件参数的设置过后，调用此函数完成设备的连接。后面就可以调用设备完成数据收发等功能了。
输入参数	无
返回值	==0:连接成功 其他，查询错误码
示例	tsapp_connect();

19. tsapp_disconnect

函数名称	int tsapp_disconnect();
功能介绍	断开设备连接
调用位置	不需要使用硬件设备，调用此函数断开设备连接
输入参数	无
返回值	==0:断开设备成功 其他值，查询错误码
示例	tsapp_disconnect();

20. tsapp_transmit_can_async

函数名称	int tsapp_transmit_can_async(TLIBCAN ACAN);
功能介绍	以异步的方式发送 CAN 报文
调用位置	发送 CAN 报文
输入参数	ACAN：CAN 数据包。TLIBCAN 数据组成请查看章节：TLIBCAN：Classic CAN 数据类型。
返回值	==0：发送成功 其他值：发送失败，失败原因请查看错误码

示例	//先准备好要发送的CAN报文
	TCAN can_frame;//创建一个报文结构体
	can_frame.init_w_std_id(0x11, 8);//初始化ID为0x11,报文DLC为8
	can_frame.set_data(0, 1, 2, 3, 4, 5, 6, 7);//设置报文数据
	//然后调用发送函数把该报文数据发送出去
	if (tsapp_transmit_can_async(&can_frame) == 0)
	{
	Log("ASync Send CAN Message Success!");
	}
	else
	{
	Log("ASync Send CAN Message Failed");
	}

21. tsapp_transmit_can_sync

函数名称	int tsapp_transmit_can_sync(const PCAN ACAN, const s32 ATimeoutMS);
功能介绍	发送 CAN 报文，并检测到发送成功后，才退出此函数。此函数返回成功，代表 CAN 报文一定已经成功发送到了 CAN 总线上面。
局限性	因为此函数需要设备 API 深度支持，因此，目前支持此函数接口的只有 TS 和 Vector 系列设备。
调用位置	同步发送 CAN 报文的场合
输入参数	ACAN:报文数据

	ATimeoutMS:同步等待的超时时间
返回值	==0：发送成功 其他值：发送失败，查询错误码

示例	//先准备好要发送的CAN报文
	TCAN can_frame;//创建一个报文结构体
	can_frame.init_w_std_id(0x11, 8);//初始化ID为0x11,报文DLC为8
	can_frame.set_data(0, 1, 2, 3, 4, 5, 6, 7);//设置报文数据
	//然后调用发送函数把该报文数据发送出去
	if (tsapp_transmit_can_sync(&can_frame, 100) == 0)
	{
	Log("ASync Send CAN Message Success!");
	}
	else
	{
	Log("ASync Send CAN Message Failed");
	}

22. tsapp_transmit_canfd_async

函数名称	int tsapp_transmit_canfd_async(const PCANFD ACANFD);
功能介绍	以异步的方式发送 CANFD 报文
调用位置	发送 CANFD 报文
输入参数	ACAN：CANFD 数据包。TLIBCAN 数据组成请查看章节：TLIBCANFD：CANFD 数据类型。
返回值	==0：发送成功 其他值：发送失败，失败原因请查看错误码

示例	//先准备好要发送的CAN报文
	TCANFD can_frame;
	can_frame.init_w_std_id(0x11, 0xf);//初始化ID为0x11,报文DLC为8
	for (int i = 0; i < 64; i++)
	{
	can_frame.FData[i] = i; //初始化报文发送内容
	}
	//然后调用发送函数把该报文数据发送出去
	if (tsapp_transmit_canfd_async(&can_frame) == 0)
	{
	Log("ASync Send CAN Message Success!");
	}
	else
	{
	Log("ASync Send CAN Message Failed");
	}

23. tsapp_transmit_canfd_sync

函数名称	int tsapp_transmit_canfd_sync(const PCANFD ACANFD, const s32 ATimeoutMS);
功能介绍	发送 CANFD 报文，并检测到发送成功后，才退出此函数。此函数返回成功，代表 CANFD 报文一定已经成功发送到了 CAN 总线上面。
调用位置	发送 CANFD 报文
局限性	因为此函数需要设备 API 深度支持，因此，目前支持此函数接口的只有 TS 和 Vector 系列设备。
输入参数	ACANFD：CANFD 数据包。TLIBCANFD 数据组成请查看章节：TLIBCANFD：CANFD 数据类型。 ATimeoutMS：超时时间
返回值	==0：发送成功 其他值：发送失败，失败原因请查看错误码

示例	//先准备好要发送的CANFD报文
	TCANFD can_frame;
	can_frame.init_w_std_id(0x11, 0xf);//初始化ID为0x11,报文DLC为8
	for (int i = 0; i < 64; i++)
	{
	can_frame.FData[i] = i; //初始化报文发送内容
	}
	//然后调用发送函数把该报文数据发送出去
	if (tsapp_transmit_canfd_sync(&can_frame, 100) == 0)
	{
	Log("ASync Send CAN Message Success!");
	}
	else
	{
	Log("ASync Send CAN Message Failed");
	}

24. tsapp_add_cyclic_msg_can

函数名称	int tsapp_add_cyclic_msg_can(const PCAN ACAN, const float APeriodMS);
功能介绍	增加周期发送的报文，增加完成后，TSMasterAPI 自动完成报文的周期发送。
调用位置	在需要周期性发送 CAN 报文的场合
输入参数	ACAN:CAN 报文数据 APeriodMS:周期值
返回值	==0：添加成功 其他值：添加失败，查询错误码
示例	TCAN t; t.init_w_std_id(0x123, 8);

```
// sending 0x123 every 5ms
if (0 == tsapp_add_cyclic_msg_can (&t, 5)){
  // this message is scheduled
}

// to modify its data...
t.FData[0] = 0x34;
if (0 == tsapp_add_cyclic_msg_can (&t, 5)){
  // this message is updated
}
```

25. tsapp_delete_cyclic_msg_can

函数名称	int tsapp_delete_cyclic_msg_can (const PCAN ACAN);
功能介绍	删除周期性发送 CAN 报文
调用位置	在需要停止周期性发送报文的场合
输入参数	ACAN:需要被删除的周期报文
返回值	==0：删除成功 其他值：删除失败，查询错误码
示例	tsapp_delete_cyclic_msg_can(ACAN);

26. tsapp_delete_cyclic_msg_canfd

函数名称	int tsapp_delete_cyclic_msg_canfd(const PCANFD ACANFD)
功能介绍	删除周期性发送 CANFD 报文
调用位置	在需要停止周期性发送报文的场合
输入参数	ACANFD:需要被删除的周期报文
返回值	==0：删除成功 其他值：删除失败，查询错误码
示例	PCANFD ACANFD= new PCANFD(); tsapp_delete_cyclic_msg_can(ACANFD);

27. tsapp_delete_cyclic_msgs

函数名称	int tsapp_delete_cyclic_msgs(void)
功能介绍	删除所有周期性报文
调用位置	在需要停止周期性发送报文的场合
输入参数	
返回值	==0：删除成功 其他值：删除失败，查询错误码
示例	tsapp_delete_cyclic_msgs(void)

28. tsfifo_clear_can_receive_buffers

函数名称	int tsfifo_clear_can_receive_buffers(const s32 AIdxChn);
功能介绍	清除 CAN 通道里面缓存的报文
调用位置	在执行交互协议之前（比如 UDS 诊断），先调用此命令把缓存中的历史数据清除掉
输入参数	AIdxChn:硬件通道
返回值	返回缓存中 CAN 报文的数量

示例	if (tsfifo_clear_can_receive_buffers(CH1) == 0x00)
	{
	Log("Clear CAN Messages Success!");
	}

29. tsfifo_receive_can_msgs

函数名称	int tsfifo_receive_can_msgs(PCAN ACANBuffers, ps32 ACANBufferSize, const s32 AIdxChn, const bool AIncludeTx);
功能介绍	读取缓存的 CAN 报文帧
调用位置	在需要读取报文数据包的场合
输入参数	ACANBuffers:数据 Buffer，用于存储读取到的报文，该 Buffer 需要函数调用方创建 ACANBufferSize:消息 Buffer 的大小 AIdxChn:目标通道：对于多通道设备，本函数选择读取哪一个通道的数据，该参数可以为空，默认为通道 1 AIncludeTx:==0:仅仅接收 Rx 报文；>0: Tx Rx 报文都读取回来，该参数可以为空，默认为只接收 Rx 报文
返回值	实际读取到的报文数量，如果没有任何报文，则返回值为 0。 // 如果一直无法读取到报文值，请检查是否通过 tsapp_enable_receive_fifo()开启了内部 Buffer

示例	//申请存储100个报文的数组
	TCAN canBuffer[100] ;
	int revCnt = sizeof(canBuffer)/sizeof(canBuffer[0]);
	//读取报文缓存，如果Rev>0，表示读取到了报文
	tsfifo_receive_can_msgs(
	canBuffer,
	&revCnt,
	CH1, //读取通道1的报文数据
	1); //接收TX/RX所有报文，如果只读取接收端的报文，则修改为READ_TX_RX_DEF.ONLY_RX_MESSAGES
	if (revCnt == 0)
	{
	//Log("No Message Received! ");
	return;
	}

30. tsfifo_receive_flexray_msgs

函数名称	int tsfifo_receive_flexray_msgs(PFlexRay AFlexRay,ps32 BufferSize, const s32 AIdxChn, const bool includeTX)
功能介绍	读取缓存的 flexray 报文帧
调用位置	在需要读取报文数据包的场合
输入参数	AFlexRay:flexray 报文 BufferSize:消息 Buffer 的大小 AIdxChn:目标通道：对于多通道设备，本函数选择读取哪一个通道的数据，该参数可以为空，默认为通道 1 AIncludeTx:==0:仅仅接收 Rx 报文；>0: Tx Rx 报文都读取回来，该参数可以为空，默认为只接收 Rx 报文
返回值	实际读取到的报文数量，如果没有任何报文，则返回值为 0。 // 如果一直无法读取到报文值，请检查是否通过 tsapp_enable_receive_fifo()开启了内部 Buffer

示例	//申请存储100个报文的数组
	TFlexray FlexrayBuffer[100] ;
	int revCnt = sizeof(FlexrayBuffer)/sizeof(FlexrayBuffer[0]);
	//读取报文缓存，如果Rev>0，表示读取到了报文
	tsfifo_receive_flexray_msgs(
	FlexrayBuffer,
	&revCnt,
	CH1, //读取通道1的报文数据
	1); //接收TX/RX所有报文，如果只读取接收端的报文，则修改为READ_TX_RX_DEF.ONLY_RX_MESSAGES
	if (revCnt == 0)
	{
	//Log("No Message Received! ");
	return;
	}

31. tsfifo_clear_canfd_receive_buffers

函数名称	int tsfifo_clear_canfd_receive_buffers(const s32 AIdxChn);
功能介绍	清除 CANFD 通道里面缓存的报文
调用位置	在执行交互协议之前（比如 UDS 诊断），先调用此命令把缓存中的历史数据清除掉
输入参数	AIdxChn:硬件通道
返回值	返回缓存中 CAN 报文的数量

示例	if(tsfifo_clear_canfd_receive_buffers(CH1) == 0x00)
	{
	Log(“ Clear CANFD Messages Success!”);
	}

32. tsfifo_read_canfd_buffer_frame_count

函数名称	int tsfifo_read_canfd_buffer_frame_count(const s32 AIdxChn, ps32 ACount);
功能介绍	读取通道内缓存的 CANFD 报文的数量
调用位置	在需要判断缓存中是否有 CANFD 报文的场合
输入参数	AChn:硬件通道 ACount:报文数量指针
返回值	**返回缓存中 CAN 报文的数量**
示例	Int count = 0; tsfifo_read_canfd_buffer_frame_count(0,&count);

33. tsfifo_receive_canfd_msgs

函数名称	int tsfifo_receive_canfd_msgs(PCANFD ACANFDBuffers, ps32 ACANFDBufferSize, const s32 AIdxChn, const bool AIncludeTx);
功能介绍	**读取缓存的 CANFD 报文帧**
调用位置	在需要读取 CANFD 报文的场合
输入参数	ACANFDBuffers:数据 Buffer，用于存储读取到的报文，该 Buffer 需要函数调用方创建 ACANBufferSize:消息 Buffer 的大小 AIdxChn:目标通道：对于多通道设备，本函数选择读取哪一个通道的数据，该参数可以为空，默认为通道 1 AIncludeTx:==0:仅仅接收 Rx 报文；>0: Tx Rx 报文都读取回来，该参数可以为空，默认为只接收 Rx 报文
返回值	返回实际读取的报文数量，如果没有任何报文，则返回值为 0。如果一直无法读取报文，请检查是否通过函数 TsMasterApi.tsapp_enable_receive_fifo();开启了驱动内部 Buffer
示例	参考 CAN 报文示例

34. tsapp_register_event_can

函数名称	int tsapp_register_event_can(const ps32 AObj, const TCANEvent AEvent);
功能介绍	注册 CAN 数据包接收回调函数
调用位置	如果用户想基于接收回调机制来处理接收的报文，在连接工具成功过后，可以调用此函数注册回调函数。
输入参数	AObj:唯一句柄，可以用作标识符用 AEvent:处理接收报文的回调函数
返回值	==0：注册成功 其他值：注册失败
示例	s32 obj = 0 ; tsapp_register_event_can(&obj, vCANQueueEventObj);

35. tsapp_unregister_event_can

函数名称	int tsapp_unregister_event_can(const ps32 AObj, const TCANEvent AEvent);
功能介绍	反注册 CAN 数据接收函数
调用位置	在不需要采用回调机制接收 CAN 报文的时候，调用此函数
输入参数	AObj:唯一句柄，可以用作标识符用 AEvent:处理接收报文的回调函数
返回值	==0：注册成功 其他值：反注册失败
示例	Int obj = 0 ; tsapp_unregister_event_can(&obj, vCANQueueEventObj);

36. tslin_set_node_funtiontype

函数名称	int tslin_set_node_funtiontype(APP_CHANNEL AIdxChn, TLINNodeType AFunctionType);
功能介绍	设置 LIN 节点工作模式：主节点，从节点，监听节点
调用位置	初始化 LIN 通道的函数中
输入参数	AIdxChn：通道编号 AFunctionType：0：主节点；1：从节点；2：监听节点，节点类型定义如下： public enum TLINNodeType { T_MasterNode = 0, T_SlaveNode = 1, T_MonitorNode = 2 };</param>
返回值	==0：设置成功 其他值：设置失败
示例	tslin_set_node_funtiontype(CH1, TLINNodeType.T_MasterNode)

37. tsapp_configure_baudrate_lin

函数名称	int tsapp_configure_baudrate_lin(int AIdxChn, Single ABaudrateKbps, s32 AProtocol);
功能介绍	**设置 LIN 通道的波特率等参数**
调用位置	连接硬件设备之前，先设置通道的参数
输入参数	AIdxChn：应用程序通道编号 ABaudrateKbps：波特率（kbps） AProtocol：{0:}LIN_PROTOCL_13 ,{1:}LIN_PROTOCL_20 ,{;2:}LIN_PROTOCL_21 ,{;3:}LIN_PROTOCL_J2602
返回值	==0:连接成功 其他：检查错误码

示例	if(tsapp_configure_baudrate_lin((int)CH1, 19.2) == 0)
	{
	Log("LIN Channel " + (1).ToString() + " baudrate has been configured");
	}
	else
	{
	Log("LIN Channel " + (1).ToString() + " baudrate failed");
	}

38. tsapp_transmit_lin_async

函数名称	int tsapp_transmit_lin_async(const PLIN ALIN);
功能介绍	异步发送 LIN 报文
调用位置	在需要发送 LIN 报文的场合
输入参数	ALIN：LIN 报文数据
返回值	==0：发送成功 其他值：发送失败

示例	TLIN lin_frame;
	lin_frame.init_w_id(0x11, 8); //初始化PID为0x11 报文长度8
	lin_frame.property__set_is_tx(true);//设置为发送报文
	for (u32 i = 0; i < 8; i++) {
	lin_frame.FData[i] = 0;
	}
	if (tsapp_transmit_lin_async(&lin_frame) == 0x00)
	{
	Log("Async lin message success!");
	}
	else
	{
	Log("Async lin message failed!");
	}

39. tsfifo_clear_lin_receive_buffers

函数名称	int tsfifo_clear_lin_receive_buffers(const s32 AIdxChn);
功能介绍	清除 LIN 模块的接收 Buffer 缓存
调用位置	在准备开始一次发送和接收之前，调用此函数清除驱动中的 LIN 缓存中的数据，确保接收到的数据是最新的
输入参数	AIdxChn：需要清除缓存的 LIN 通道编号
返回值	==0：清除成功 其他值：清除失败
示例	tsfifo_clear_lin_receive_buffers(CH1);

40. tsfifo_read_lin_buffer_frame_count

函数名称	int tsfifo_read_lin_buffer_frame_count(const s32 AIdxChn, ps32 ACount);
功能介绍	读取通道内缓存的 LIN 报文的数量
调用位置	在需要判断缓存中是否有 LIN 报文的场合
输入参数	AChn：硬件通道 ACount ：报文数量
返回值	**返回缓存中 LIN 报文的数量**
示例	Int count = 0; tsfifo_read_lin_buffer_frame_count(0,&count);

41. tsfifo_receive_lin_msgs

函数名称	int tsfifo_receive_lin_msgs(PLIN ALINBuffers, ps32 ALINBufferSize, const s32 AIdxChn, const bool AIncludeTx);
功能介绍	**读取缓存的 LIN 报文帧**
调用位置	**在需要读取 LIN 报文数据包的场合**
输入参数	ALINBuffers：数据 Buffer，用于存储读取到的报文，该 Buffer 需要函数调用方创建 ALINBufferSize：LIN 消息 Buffer 的大小 AIdxChn：目标通道：对于多通道设备，本函数选择读取哪一个通道的数据，该参数可以为空，默认为通道 1 AIncludeTx ：==0:仅仅接收 Rx 报文；>0: Tx Rx 报文都读取回来，该参数可以为空，默认为只接收 Rx 报文
返回值	实际读取到的报文数量，如果没有任何报文，则返回值为 0。 // 如果一直无法读取到报文值，请检查是否通过 TsMasterApi.tsapp_enable_receive_fifo()开启了内部 Buffer
示例	参考 CAN 总线代码

42. tsfifo_clear_fastlin_receive_buffers

函数名称	Int tsfifo_clear_fastlin_receive_buffers(const s32 AIdxChn);
功能介绍	清除 FastLIN 模块的接收 Buffer 缓存
调用位置	在准备开始一次发送和接收之前，调用此函数清除驱动中的 LIN 缓存中的数据，确保接收到的数据是最新的
输入参数	AIdxChn：需要清除缓存的 LIN 通道编号>
返回值	==0：清除成功 其他值：清除失败
示例	tsfifo_clear_fastlin_receive_buffers(CH1);

43. tsfifo_read_fastlin_buffer_frame_count

函数名称	int tsfifo_read_fastlin_buffer_frame_count(const s32 AIdxChn, ps32 ACount);
功能介绍	读取通道内缓存的 LIN 报文的数量

调用位置	在需要判断缓存中是否有 LIN 报文的场合
输入参数	AChn：硬件通道 ACount ： 报文数量
返回值	**返回缓存中 LIN 报文的数量**
示例	tsfifo_read_fastlin_buffer_frame_count(0,&ACount);

44. tsfifo_receive_fastlin_msgs

函数名称	int tsfifo_receive_fastlin_msgs(PLIN ALINBuffers, ps32 ALINBufferSize, const s32 AIdxChn, const bool AIncludeTx);
功能介绍	**读取缓存的 LIN 报文帧**
调用位置	**在需要读取 LIN 报文数据包的场合**
输入参数	ALINBuffers：数据 Buffer，用于存储读取到的报文，该 Buffer 需要函数调用方创建 ALINBufferSize：LIN 消息 Buffer 的大小 AIdxChn：目标通道：对于多通道设备，本函数选择读取哪一个通道的数据，该参数可以为空，默认为通道 1 AIncludeTx ： ==0:仅仅接收 Rx 报文；>0: Tx Rx 报文都读取回来，该参数可以为空，默认为只接收 Rx 报文
返回值	实际读取到的报文数量，如果没有任何报文，则返回值为 0。 // 如果一直无法读取到报文值，请检查是否通过 TsMasterApi.tsfifo_enable_receive_fifo()开启了内部 Buffer
示例	参考 CAN 报文示例

45. tsapp_start_logging

函数名称	int tsapp_start_logging(const char* AFullFileName)
功能介绍	启动报文记录，记录文件格式未 blf 格式
调用位置	需要记录整个运行过程中总线上的报文通讯
输入参数	AFullFileName ： 记录文件名（后缀为.blf），可以为空字符串“” 注意：如果 AFileName 传输为空字符串“”，则数据文件默认存储在 TSMaster 的安装路径下面。AFile 数据格式路径不能出错，如果路径是无效的，创建该数据文件无效，会触发模块内部异常。
返回值	==0：执行成功 其他值：查询错误码
示例	tsapp_start_logging(".\TestData.blf");

46. tsapp_stop_logging

函数名称	int tsapp_stop_logging()
功能介绍	停止报文记录
调用位置	该函数需要和 tsapp_start_logging 配合使用，停止报文记录，报文会保存为相应的 blf 文件。
输入参数	无

返回值	==0：执行成功 其他值：查询错误码
示例	tsapp_stop_logging();

47. tsdb_unload_can_dbs

函数名称	int tsdb_unload_can_dbs()
功能介绍	卸载所有已经加载的 DBC 文件
调用位置	该函数必须在应用处于断开状态（执行 tsapp_connect 之前或者执行了 tsapp_disconnect 之后）调用。当应用处于连接状态的时候，不能删除，加载数据库文件。
输入参数	无
返回值	==0：执行成功 其他值：查询错误码
示例	tsdb_unload_can_dbs();

48. tsdb_unload_can_db

函数名称	int tsdb_unload_can_db(UInt32 AId)
功能介绍	卸载指定编号的数据库文件
调用位置	该函数必须在应用处于断开状态（执行 tsapp_connect 之前或者执行了 tsapp_disconnect 之后）调用。当应用处于连接状态的时候，不能删除，加载数据库文件。
输入参数	AId：加载数据库成功过后，返回系统为该数据库分配的唯一编号
返回值	==0：执行成功 其他值：查询错误码
示例	tsdb_unload_can_db(0);

49. tsdb_load_can_db

函数名称	int tsdb_load_can_db(const char* ADBC, const char* ASupportedChannelsBased0, u32* AId);
功能介绍	把数据库加载到指定的通道上，并获取加载数据库的编号
调用位置	该函数必须在应用处于断开状态（执行 tsapp_connect 之前或者执行了 tsapp_disconnect 之后）调用。当应用处于连接状态的时候，不能删除，加载数据库文件。
输入参数	ADBC:DBC 绝对路径 ASupportedChannelsBased0：绑定的通道数组 AId：加载数据库成功过后，返回系统为该数据库分配的唯一编号
返回值	==0：执行成功 其他值：查询错误码
示例	

50. tsdb_set_signal_value_can

函数名称	int tsdb_set_signal_value_can(const PCAN ACAN, const char* AMsgName, const char* ASgnName, const double AValue);
功能介绍	根据 Message 名称，信号名称，把信号值更新到 CAN 报文中。
调用位置	加载数据库过后，任意位置（应用连接状态，断开状态）都可以调用此函数。
输入参数	ACAN：原始 CAN 报文(引用) AMsgName：报文名称 ASgnName：信号名称 AValue：信号值
返回值	==0：执行成功 其他值：查询错误码
示例	

51. tsdb_get_signal_value_can

函数名称	int tsdb_get_signal_value_can(const PCAN ACAN, const char* AMsgName, const char* ASgnName, double* AValue);
功能介绍	根据 Message 名称，信号名称，从 CAN 报文中读取信号值
调用位置	加载数据库过后，任意位置（应用连接状态，断开状态）都可以调用此函数。
输入参数	ACAN：原始 CAN 报文(引用) AMsgName：报文名称 ASgnName：信号名称 AValue：信号值(引用类型)
返回值	==0：执行成功 其他值：查询错误码

示例	if (tsdb_get_signal_value_can(&Data, tBMessageName.Text,tBSignalName.Text, &testSignalValue) == 0x00)
	{
	  //ReadData Success
	};

52. tsdb_set_signal_value_canfd

函数名称	int tsdb_set_signal_value_canfd(const PCANFD ACANFD, const char* AMsgName, const char* ASgnName, const double AValue);
功能介绍	根据 Message 名称，信号名称，从 CANFD 报文中读取信号值
调用位置	加载数据库过后，任意位置（应用连接状态，断开状态）都可以调用此函数。
输入参数	ACANfd：CANFD 报文（引用） AMsgName：报文名称 ASgnName：信号名称 AValue：信号值

返回值	==0：执行成功 其他值：查询错误码

示例	//首先准备好原始 CANFD 报文，或者使用其他全局 CANFD 报文
	TLIBCANFD CANFDFrame = new TLIBCANFD(0, 0x123, true, false, false, 8); double srcSignalValue = 1.6; //信号物理值
	//通过该函数把信号值更新到 CAN 报文中
	if (CheckResultOK(tsdb_set_signal_value_canfd(&CANFDFrame, "EngineData", "Engine_Speed", srcSignalValue)))
	{
	Log("Set SignalValue:" + srcSignalValue.ToString("F2") + "Success!");
	}
	//调用发送函数把 CAN 报文发送出去
	tsapp_add_cyclic_msg_canfd(&CANFDFrame , 10);

53. tsdb_get_signal_value_canfd

函数名称	int tsdb_get_signal_value_canfd(const PCANFD ACANFD, const char* AMsgName, const char* ASgnName, double* AValue);
功能介绍	根据 Message 名称，信号名称，把信号值更新到 CANFD 报文中。
调用位置	加载数据库过后，任意位置（应用连接状态，断开状态）都可以调用此函数。
输入参数	ACANfd：CANFD 报文（引用） AMsgName：报文名称 ASgnName：信号名称 AValue：信号值（引用）
返回值	==0：执行成功 其他值：查询错误码

示例	//首先准备好原始 CANFD 报文，或者使用其他全局 CANFD 报文
	TLIBCANFD CANFDFrame = new TLIBCANFD(0, 0x123, true, false, false, 8); double srcSignalValue = 1.6; //信号物理值
	//通过该函数把信号值更新到 CAN 报文中
	if (CheckResultOK(tsdb_set_signal_value_canfd(&CANFDFrame, "EngineData", "Engine_Speed", srcSignalValue)))
	{
	Log("Set SignalValue:" + srcSignalValue.ToString("F2") + "Success!");
	}
	//调用发送函数把 CAN 报文发送出去
	tsapp_add_cyclic_msg_canfd(&CANFDFrame , 10);

54. tsapp_add_application

函数名称	int tsapp_add_application(const char* AAppName);

功能介绍	通过名称添加应用，可以添加或编辑通道映射
调用位置	
输入参数	AAppName：应用程序名称
返回值	==0：执行成功 其他值：查询错误码

示例	// add a new application named "TSMaster"
	tsapp_add_application("TSMaster");

55. tsapp_add_cyclic_msg_canfd

函数名称	int tsapp_add_cyclic_msg_canfd(const PCANFD ACANFD, const float APeriodMS);
功能介绍	定期发送特定的 CAN FD 消息，或更新已安排的 CAN 消息
调用位置	
输入参数	AAppName：应用程序名称 APeriodMS：周期
返回值	==0：执行成功 其他值：查询错误码

示例	TCANFD t;
	t.init_w_std_id(0x123, 8);
	// sending 0x123 every 5ms
	if (0 == tsapp_add_cyclic_message_canfd(&t, 5)){
	// this message is scheduled
	}
	
	// to modify its data...
	t.FData[0] = 0x34;
	if (0 == tsapp_add_cyclic_message_canfd(&t, 5)){
	// this message is updated
	}

56. tsapp_clear_bus_statistics

函数名称	void tsapp_clear_bus_statistics ();
功能介绍	清除总线统计数据
调用位置	
输入参数	无
返回值	无

示例	// clear all statistics data
	tsapp_clear_bus_statistics();

57. tsapp_connect

函数名称	void tsapp_connect ();
功能介绍	通过与每个映射通道建立连接来连接到指定的应用程序
调用位置	

输入参数	无
返回值	无

示例	// connect the application using stored configuration
	if (0 == tsapp_connect()){
	log("Application has been connected");
	}

58. tsapp_del_mapping

函数名称	int tsapp_del_mapping(const PLIBTSMapping AMapping);
功能介绍	删除一个从 PLIBTSMapping 指针读取的应用映射，在调用这个函数之前应该指定 FAppName, FAppChannelIndex 和 FAppChannelType
调用位置	
输入参数	AMapping：指针指向已经分配的 map 结构体
返回值	==0：执行成功 其他值：查询错误码

示例	TLIBTSMapping m;
	m.init();
	// just set the following 3 fields to delete the mapping info:
	sprintf(m.FAppName, "TSMaster"); // Application is TSMaster
	m.FAppChannelIndex = 0; // Channel 1
	m.FAppChannelType = APP_CAN; // Application channel type can be APP_CAN or APP_LIN
	if (0 == tsapp_del_mapping(&m)){
	log("application CAN channel 1 mapping has been deleted");
	}

59. tsapp_del_application

函数名称	int tsapp_del_application(const char* AAppName);
功能介绍	按名称删除应用程序，删除后，指定应用程序的所有映射将消失
调用位置	
输入参数	AAppName：应用程序名称
返回值	==0：执行成功 其他值：查询错误码

示例	// delete application "TSMaster"
	tsapp_del_application("TSMaster");

60. tsapp_enable_bus_statistics

函数名称	int tsapp_enable_bus_statistics(const bool AEnable);
功能介绍	使能总线统计定时器，计算总线统计信息
调用位置	
输入参数	AEnable： 是否启用总线计时器

返回值	==0：执行成功 其他值：查询错误码

示例	// enable bus statistics timer
	sapp_enable_bus_statistics(true);

61. execute_python_string

函数名称	int tsapp_execute_python_string(const char* AString, const bool AIsSync, const bool AIsX64, char** AResultLog);
功能介绍	以字符串形式执行 python 代码
调用位置	
输入参数	AString ： Python 字符串代码 AIsSync ：True:同步执行 false:异步执行 AIsX64：true:使用 x64 发行版，false:使用 x86 发行版，注意:x64 发行版默认不安装，需要手动拷贝文件夹到 bin\Data\Python\active_version\x64 AResultLog ：从 python 执行环境打印结果或堆栈跟踪
返回值	==0：执行成功 其他值：查询错误码

示例	// note: x64 distribution is not installed by default, you need to manually copy folder to bin\Data\Python\active_version\x64
	char* s =
	"\
	import sys\n\
	import win32api, win32con\n\
	win32api.MessageBox(0, 'test with arguments: ' + sys.argv[1] + ' ' + sys.argv[2], 'title', win32con.MB_OK)\n\
	print('OK')\n\
	";
	
	char* p;
	if (tsapp_check(tsapp_execute_python_string(s, "arg1 arg2", true, false, &p))){
	log("execution result is %s", p);
	} else {
	log("execution failed with result %s", p);
	}

62. tsapp_execute_python_script

函数名称	int tsapp_execute_python_script(const char* AFilePath, const char*AArguments,const bool AIsSync, const bool AIsX64, char** AResultLog);
功能介绍	执行 python 脚本文件并获取其执行结果
调用位置	

输入参数	AFilePath ： python 脚本文件的完整路径 AArguments ： 传递给 python 的参数 AIsSync ： true:同步执行，false:异步执行 AIsX64： true:使用 x64 环境，false:使用 x86 环境，注意：默认情况下未安装 x64 环境，您需要手动将文件夹复制到 bin\Data\Python\active_version\x64 AResultLog ： 从 python 执行环境打印结果或堆栈跟踪
返回值	==0：执行成功 其他值：查询错误码

示例	// note: x64 distribution is not installed by default, you need to manually copy folder to bin\Data\Python\active_version\x64
	char* s =
	"\
	import sys\n\
	import win32api, win32con\n\
	win32api.MessageBox(0, 'test with arguments: ' + sys.argv[1] + ' ' + sys.argv[2], 'title', win32con.MB_OK)\n\
	print('OK')\n\
	";
	
	char* p;
	if (tsapp_check(tsapp_execute_python_string(s, "arg1 arg2", true, false, &p))){
	log("execution result is %s", p);
	} else {
	log("execution failed with result %s", p);
	}tsapp_execute_python_script()

63. tsapp_get_application_list

函数名称	int tsapp_get_application_list(char** AAppNameList);;
功能介绍	获取此计算机上现有应用程序名称的列表，每个应用程序名称以";"分隔。
调用位置	
输入参数	AAppNameList ： 检索到的应用程序名称由";"分隔
返回值	==0：执行成功 其他值：查询错误码

示例	// get current application list
	char* p;
	if (0 == tsapp_get_application_list(&p)){
	log("application listed: %s", p);
	}

64. tsapp_get_bus_statistics

函数名称	int tsapp_get_bus_statistics(const TLIBApplicationChannelType ABusType, const s32 AIdxChn, const TLIBCANBusStatistics AIdxStat, pdouble AStat);
功能介绍	获取总线负载率
调用位置	
输入参数	ABusType ： 应用程序通道类型 AIdxChn ： 通道索引从 0 开始 AIdxStat ： 总线统计值索引 AStat ： 检索到的状态值
返回值	==0：执行成功 其他值：查询错误码

示例	/*
	
	typedef enum {
	cbsBusLoad = 0, cbsPeakLoad, cbsFpsStdData, cbsAllStdData,
	cbsFpsExtData, cbsAllExtData, cbsFpsStdRemote, cbsAllStdRemote,
	cbsFpsExtRemote, cbsAllExtRemote, cbsFpsErrorFrame, cbsAllErrorFrame
	} TLIBCANBusStatistics;
	
	*/
	
	// to get bus load of channel 1
	double d = 0;
	if (0 == tsapp_get_bus_statistics(
	APP_CAN,
	0,
	cbsBusLoad,
	&d
	)){
	log("Bus statistics is %.2f%%", d);
	}

65. tsapp_get_can_channel_count

函数名称	int tsapp_get_can_channel_count(const ps32 ACount);
功能介绍	获取应用程序 CAN 通道的总数
调用位置	
输入参数	ACount ： 检索到的 CAN 通道计数
返回值	==0：执行成功 其他值：查询错误码
示例	// get CAN channel count of current application

```
    s32 n;
    if (0 == tsapp_get_can_channel_count(&n)){
      // CAN channel count is stored in variable n
      log("CAN channel count = %d", n);
}
```

66. tsapp_get_current_application

函数名称	int tsapp_get_current_application(const char** AAppName);
功能介绍	获取应用程序 CAN 通道的总数
调用位置	
输入参数	AAppName ： 应用程序名称
返回值	==0：执行成功 其他值：查询错误码

示例	// to read the current app name
	char* p;
	tsapp_get_current_application(&p);
	log("the current app name is %s", p);

67. tsapp_get_fps_can

函数名称	int tsapp_get_fps_can(const s32 AIdxChn, const s32 AIdentifier, ps32 AFPS);
功能介绍	获取特定标识符的每秒帧数，注意:此功能需要启用总线统计
调用位置	
输入参数	AIdxChn ： 通道索引 AIdentifier ： 指定 CAN 报文标识符 AFPS：特定标识符的每秒帧数
返回值	==0：执行成功 其他值：查询错误码

示例	// get fps of 0x100 on channel 1
	s32 fps;
	if (0 == tsapp_get_fps_can(0, 0x100, &fps)){
	// fps retrieved
	}

68. tsapp_get_fps_canfd

函数名称	int tsapp_get_fps_canfd(const s32 AIdxChn, const s32 AIdentifier, ps32 AFPS);
功能介绍	获取特定标识符的每秒帧数，注意:此功能需要启用总线统计
调用位置	
输入参数	AIdxChn ： 通道索引 AIdentifier ： 报文标识符 AFPS ： 指定标识符报文的每秒帧数
返回值	==0：执行成功

	其他值：查询错误码

示例	// get fps of 0x100 on channel 1
	s32 fps;
	if (0 == tsapp_get_fps_canfd(0, 0x100, &fps)){
	// fps retrieved
	}

69. tsapp_get_fps_lin

函数名称	int tsapp_get_fps_lin(const s32 AIdxChn, const s32 AIdentifier, ps32 AFPS);
功能介绍	获取特定标识符的每秒帧数，注意:此功能需要启用总线统计
调用位置	
输入参数	AIdxChn ：通道索引 AIdentifier ：报文标识符 AFPS ： 指定报文的每秒帧数
返回值	==0：执行成功 其他值：查询错误码

示例	// get fps of 0x10 on channel 1
	s32 fps;
	if (0 == tsapp_get_fps_lin(0, 0x10, &fps)){
	// fps retrieved
	}

70. tsapp_get_lin_channel_count

函数名称	int tsapp_get_lin_channel_count(const ps32 ACount);
功能介绍	获取应用程序 LIN 通道的总数
调用位置	
输入参数	ACount：检索到的 LIN 通道计数
返回值	==0：执行成功 其他值：查询错误码

示例	// get LIN channel count of current application
	s32 n;
	if (0 == tsapp_get_lin_channel_count(&n)){
	// LIN channel count is stored in variable n
	log("LIN channel count = %d", n);
	}

71. tsapp_get_timestamp

函数名称	int tsapp_get_timestamp(s64* ATimestamp);
功能介绍	获取当前测量的 PC 时间戳
调用位置	
输入参数	ATimestamp ：检索到的时间戳（微秒）

返回值	==0：执行成功 其他值：查询错误码

示例	s64 t;
	if (0 == tsapp_get_timestamp(&t)) {
	// timestamp retrieved in t
	log("current timestamp is %f", t / 1000000.0);
	}

72. tsapp_get_turbo_mode

函数名称	int tsapp_get_turbo_mode(const bool* AEnable);
功能介绍	获取 turbo 模式启用状态
调用位置	
输入参数	AEnable ： 检索到激活状态
返回值	==0：执行成功 其他值：查询错误码

示例	// get the turbo mode activation status of TSMaster
	bool a;
	if (0 == tsapp_get_turbo_mode(&a)){
	log("Turbo mode status is %d", a);
	}

73. tsfifo_enable_receive_error_frames

函数名称	int tsapp_get_turbo_mode(const bool* AEnable);
功能介绍	获取 turbo 模式启用状态
调用位置	
输入参数	AEnable ： 检索到激活状态
返回值	==0：执行成功 其他值：查询错误码

示例	// get the turbo mode activation status of TSMaster
	bool a;
	if (0 == tsapp_get_turbo_mode("TSMaster", &a)){
	log("Turbo mode status is %d", a);
	}

74. tsapp_set_current_application

函数名称	int tsapp_set_current_application(const char* AAppName);
功能介绍	设置当前应用程序名称，不同的应用程序拥有不同的资源映射
调用位置	
输入参数	AAppName ：应用程序名称
返回值	==0：执行成功 其他值：查询错误码
示例	tsapp_set_current_application("TSMaster");

75. tsapp_set_mapping

函数名称	int tsapp_set_mapping(const PLIBTSMapping AMapping);
功能介绍	保存一个从 PLIBTSMapping 指针读取的应用程序映射
调用位置	
输入参数	AMapping ： 指针指向已经分配的 map 结构体
返回值	==0：执行成功 其他值：查询错误码

示例	TLIBTSMapping m;
	m.init();
	sprintf(m.FAppName, "TSMaster"); // Application is TSMaster
	sprintf(m.FHWDeviceName, "TSCANMini"); // set device name
	m.FAppChannelIndex = 0; // Channel 1
	m.FAppChannelType = APP_CAN; // Application channel type can be APP_CAN or APP_LIN
	m.FHWDeviceType = TS_USB_DEVICE; // TLIBBusToolDeviceType can be TS_USB_DEVICE, XL_USB_DEVICE or TS_TCP_DEVICE
	m.FHWIndex = 0; // The first hardware connected to computer
	m.FHWChannelIndex = 0; // The first CAN channel of the hardware
	m.FHWDeviceSubType = 3; // 3 means TS.CAN Mini
	if (0 == tsapp_set_mapping(&m)){
	log("channel mapping on application CAN channel 1 has been set");
	}

76. tsapp_set_turbo_mode

函数名称	int tsapp_set_turbo_mode(const bool AEnable);
功能介绍	启用或禁用 turbo 模式，在 turbo 模式下，通信延迟可以最小化，但功耗会增加
调用位置	
输入参数	AEnable ： true: turbo mode is active; false: turbo mode is not active
返回值	==0：执行成功 其他值：查询错误码

示例	// set turbo mode active on TSMaster
	if (0 == tsapp_set_turbo_mode("TSMaster", true)){
	log("Turbo mode has been activated");
	}

77. tsapp_transmit_lin_sync

函数名称	int tsapp_transmit_lin_sync(const PLIN ALIN, const s32 ATimeoutMS);

功能介绍	同步发送一个 LIN 帧，当前的执行被阻塞，直到帧被发送或超时
调用位置	
输入参数	ALIN ：指向 LIN 数据结构的指针 ATimeoutMS ： 超时时间
返回值	==0：执行成功 其他值：查询错误码

示例	// transmit a LIN frame (0x15) on LIN channel 1 with 1s timeout
	TLIN l;
	l.init_w_id(0x15, 8);
	if (0 != tsapp_connect()) return;
	if (0 == tsapp_transmit_lin_sync(&l, 1000)){
	log("LIN message 0x%02x has been transmitted on LIN Channel 1", l.FIdentifier);
	}

78. tscom_can_rbs_start

函数名称	void tscom_can_rbs_start ();
功能介绍	启动 CAN - RBS 发动机
调用位置	
输入参数	无
返回值	无
示例	tscom_can_rbs_start();

79. tscom_can_rbs_stop

函数名称	void tscom_can_rbs_stop ();
功能介绍	停止 CAN - RBS 发动机
调用位置	
输入参数	无
返回值	无
示例	tscom_can_rbs_stop();

80. tscom_can_rbs_is_running

函数名称	int tscom_can_rbs_is_running(bool* AIsRunning);;
功能介绍	检查 CAN RBS 是否正在运行
调用位置	
输入参数	AIsRunning ： 是否运行
返回值	==0：执行成功 其他值：查询错误码
示例	bool isRunning; tscom_can_rbs_is_running(&isRunning);

81. tscom_can_rbs_configure

函数名称	int tscom_can_rbs_configure(const bool AAutoStart, const bool AAutoSendOnModification, const bool AActivateNodeSimulation, const TLIBRBSInitValueOptions AInitValueOptions);
功能介绍	配置 CAN RBS 引擎
调用位置	
输入参数	AAutoStart ： 连接应用程序时 RBS 是否自动启动 AAutoSendOnModification ： 如果其信号被修改，则是否自动发送修改后的消息，仅对非循环消息有效 AActivateNodeSimulation： 是否激活节点行为模拟 AInitValueOptions ： 初始值选项
返回值	==0：执行成功 其他值：查询错误码
示例	// typedef enum {rivUseDB = 0, rivUseLast, rivUse0} TLIBRBSInitValueOptions; tscom_can_rbs_configure(true, true, true, rivUseDB);

82. tscom_can_rbs_activate_all_networks

函数名称	int tscom_can_rbs_activate_all_networks(const bool AEnable, const bool AIncludingChildren);
功能介绍	设置激活或停用所有网络
调用位置	
输入参数	AEnable： 是否激活所有网络 AIncludingChildren：是否激活所有节点报文
返回值	==0：执行成功 其他值：查询错误码
示例	tscom_can_rbs_activate_all_networks(true, true);

83. tscom_can_rbs_activate_all_networks

函数名称	int tscom_can_rbs_activate_all_networks(const bool AEnable, const bool AIncludingChildren);
功能介绍	设置激活或禁用所有网络
调用位置	
输入参数	AEnable ：是否激活所有网络 AIncludingChildren ： 是否激活所有节点报文
返回值	==0：执行成功 其他值：查询错误码

示例	tscom_can_rbs_activate_all_networks(true, true);

84. tscom_can_rbs_activate_network_by_name

函数名称	int tscom_can_rbs_activate_network_by_name(const s32 AIdxChn, const bool AEnable, const char* ANetworkName, const bool AIncludingChildren);
功能介绍	是否激活或禁用网络
调用位置	
输入参数	AIdxChn ：通道索引 AEnable ： 是否激活 ANetworkName ： 要激活或停用的网络名称 AIncludingChildren ：是否激活所有节点报文
返回值	==0：执行成功 其他值：查询错误码
示例	tscom_can_rbs_activate_network_by_name(CH1, true, "Network1", true);

85. tscom_can_rbs_activate_node_by_name

函数名称	int tscom_can_rbs_activate_node_by_name(const s32 AIdxChn, const bool AEnable, const char* ANetworkName, const char* ANodeName, const bool AIncludingChildren);
功能介绍	是否激活或去激活节点
调用位置	
输入参数	AIdxChn ： 通道索引 AEnable ： 是否激活指定节点 ANetworkName ： 包含此节点的网络名称 ANodeName：要激活或停用的节点名称 AIncludingChildren：是否激活所有节点报文
返回值	==0：执行成功 其他值：查询错误码
示例	tscom_can_rbs_activate_node_by_name(CH1, true, "Network1", "Node1", true);

86. tscom_can_rbs_activate_message_by_name

函数名称	int tscom_can_rbs_activate_message_by_name(const s32 AIdxChn, const bool AEnable, const char* ANetworkName, const char* ANodeName, const char* AMsgName);
功能介绍	是否激活或取消激活报文
调用位置	
输入参数	AIdxChn ： 通道索引 AEnable： 是否激活指定报文 ANetworkName ： 包含此节点和报文的网络名称

	ANodeName ：包含此报文的节点名称 AMsgName ：报文名称
返回值	==0：执行成功 其他值：查询错误码
示例	tscom_can_rbs_activate_message_by_name(CH1, true, "Network1", "Node1", "Message1");

87. tscom_can_rbs_get_signal_value_by_element

函数名称	int tscom_can_rbs_get_signal_value_by_element(const s32 AIdxChn, const char* ANetworkName, const char* ANodeName, const char* AMsgName, const char* ASignalName, double* AValue);
功能介绍	使用元素名从 CAN RBS 获取信号实时值
调用位置	
输入参数	AIdxChn ：通道编号 ANetworkName ：要激活或停用的网络名称 ANodeName： 包含此信号的节点名称 AMsgName ：包含此信号的报文名称 ASignalName ：要获取实值的信号名称 AValue ：该信号的实值
返回值	==0：执行成功 其他值：查询错误码
示例	double d; tscom_can_rbs_get_signal_value_by_element(CH1, "Network1", "Node1", "Message1", "Signal1", &d);

88. tscom_can_rbs_get_signal_value_by_address

函数名称	int tscom_can_rbs_get_signal_value_by_address(const char* ASymbolAddress, double* AValue);
功能介绍	使用元素名从 CAN RBS 获取信号实时值
调用位置	
输入参数	ASymbolAddress ：符号地址，可通过双击所选符号的状态栏复制到 can 数据库选择器中 AValue ：该信号的实值
返回值	==0：执行成功 其他值：查询错误码
示例	// Note: symbol address has the following structure: //

	channel_index/network_name/node_name/message_name/signal_name double d; tscom_can_rbs_get_signal_value_by_address("0/CAN_FD_Powertrain/Engine/EngineData/Gear", &d);

89. tscom_can_rbs_set_signal_value_by_element

函数名称	int tscom_can_rbs_set_signal_value_by_element(const s32 AIdxChn, const char* ANetworkName, const char* ANodeName, const char* AMsgName, const char* ASignalName, const double AValue);
功能介绍	使用元素名称设置来自 CAN RBS 的信号实时值
调用位置	
输入参数	AIdxChn ：通道索引 ANetworkName：包含此信号的网络名称 ANodeName ：包含此信号的节点名称 AMsgName ：包含此信号的报文名称 ASignalName ：要获取实值的信号名称 AValue ：该信号的实值
返回值	==0：执行成功 其他值：查询错误码
示例	tscom_can_rbs_set_signal_value_by_element(0, "Network1", "Node1", "Message1", "Signal1", 12.3);

90. tscom_can_rbs_set_signal_value_by_address

函数名称	int tscom_can_rbs_set_signal_value_by_address(const char* ASymbolAddress, const double AValue);
功能介绍	使用信号数据库地址设置来自 CAN RBS 的信号实时值
调用位置	
输入参数	ASignalName：获取实值的信号名称 AValue ：该信号的实时物理值
返回值	==0：执行成功 其他值：查询错误码
示例	// Note: symbol address has the following structure: // channel_index/network_name/node_name/message_name/signal_name Tscom_can_rbs_set_signal_value_by_address(

	"1/CAN_FD_Powertrain/Engine/EngineData/Gear", 12.3);

91. tslog_set_online_replay_config

函数名称	int tslog_set_online_replay_config(const s32 AIndex, const char* AName, const char* AFileName, const bool AAutoStart, const bool AIsRepetitiveMode, const TLIBOnlineReplayTimingMode AStartTimingMode, const s32 AStartDelayTimeMs, const bool ASendTx, const bool ASendRx, const char* AMappings);
功能介绍	按引擎索引设置在线重放引擎的属性
调用位置	
输入参数	AIndex ：重放引擎索引 Name ： 重放引擎名称 AFileName ：要重放的 blf 文件 AAutoStart ：连接应用程序后是否自动重放日志文件 AIsRepetitiveMode ：True：日志文件将连续重放，False：日志文件只重放一次 AStartTimingMode ： ortimmediate（0）：回放开始时，将直接发送日志文件中的第一帧；ortAsLog（1）：根据日志文件发送第一帧；ortDelayed（2）：第一帧将在参数中指定的某个延迟后发送 AStartDelayTimeMs：第一帧将在以毫秒为单位的指定延迟时间之后发送 ASendTx ：重放 TX ASendRx ：重放 RX AMappings： 每个通道的映射在由“，”分隔的字符串中指定，通道 0 表示忽略
返回值	==0：执行成功 其他值：查询错误码
示例	tsapp_tslog_set_online_replay_config(0, "replay1", "C:\Temp\log1.blf", true, false, ortImmediately, 0, true, true, "1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32");

92. tslog_get_online_replay_count

函数名称	int tslog_get_online_replay_count(s32* ACount);
功能介绍	获得在线重放引擎数

调用位置	
输入参数	ACount： 返回的计数
返回值	==0：执行成功 其他值：查询错误码
示例	s32 i; tscom_tslog_get_online_replay_count(&i);

93. tslog_get_online_replay_config

函数名称	int tslog_get_online_replay_config(const s32 AIndex, char** AName, char** AFileName, bool* AAutoStart, bool* AIsRepetitiveMode, TLIBOnlineReplayTimingMode* AStartTimingMode, s32* AStartDelayTimeMs, bool* ASendTx, bool* ASendRx, char** AMappings);
功能介绍	按引擎索引获取在线重放引擎属性
调用位置	
输入参数	AIndex ： 重放引擎索引 AName ： 重放引擎名称 AFileName ： 重放的 blf 文件 AAutoStart ： 连接应用程序后是否自动重播日志文件 AIsRepetitiveMode ： True：日志文件将连续重放，False：日志文件只重放一次 AStartTimingMode ： ortimmediate（0）：回放开始时，将直接发送日志文件中的第一帧；ortAsLog（1）：根据日志文件发送第一帧；ortDelayed（2）：第一帧将在参数中指定的某个延迟后发送 AStartDelayTimeMs ： 第一帧将在以毫秒为单位的指定延迟时间之后发送 ASendTx ： 重放 TX ASendRx ： 重放 RX AMappings ： 每个通道的映射在由“，”分隔的字符串中指定，通道 0 表示忽略
返回值	==0：执行成功 其他值：查询错误码
示例	char* AName; char* AFileName; bool AAutoStart; bool AIsRepetitiveMode; TLIBOnlineReplayTimingMode AStartTimingMode; s32 AStartDelayTimeMs; bool ASendTx; bool ASendRx; char* AMappings; tscom_tslog_get_online_replay_config(0, &AName, &AFileName,

	&AAutoStart, &AIsRepetitiveMode, &AStartTimingMode, &AStartDelayTimeMs, &ASendTx, &ASendTx, &AMappings);

94. tslog_del_online_replay_configs

函数名称	int tslog_del_online_replay_configs(void);
功能介绍	删除所有在线重播配置
调用位置	
输入参数	无
返回值	无
示例	tscom_tslog_del_online_replay_configs();

95. tslog_start_online_replay

函数名称	int tslog_start_online_replay(const s32 AIndex);
功能介绍	启动在线重放引擎索引。注意:应用程序应在发动机启动前连接好
调用位置	
输入参数	AIndex ：重放发动机索引
返回值	==0：执行成功 其他值：查询错误码
示例	tscom_tslog_start_online_replay(0);

96. tslog_start_online_replays

函数名称	Void tslog_start_online_replays ();
功能介绍	启动所有在线回放引擎
调用位置	
输入参数	无
返回值	无
示例	tscom_tslog_start_online_replays();

97. tslog_pause_online_replay

函数名称	int tslog_pause_online_replay(const s32 AIndex);
功能介绍	按引擎索引暂停在线重放引擎
调用位置	
输入参数	AIndex ：重放引擎索引
返回值	==0：执行成功 其他值：查询错误码

示例	tscom_tslog_pause_online_replay(0);

98. tslog_pause_online_replays

函数名称	Void tslog_pause_online_replays();
功能介绍	暂停所有在线重播引擎
调用位置	
输入参数	无
返回值	无
示例	tscom_tslog_pause_online_replays();

99. tslog_stop_online_replay

函数名称	int tslog_stop_online_replay(const s32 AIndex);
功能介绍	根据引擎索引停止在线重放引擎
调用位置	
输入参数	AIndex ： 重放引擎索引
返回值	==0：执行成功 其他值：查询错误码
示例	tscom_tslog_stop_online_replay(0);

100. tslog_stop_online_replays

函数名称	Void tslog_stop_online_replays();
功能介绍	停止所有在线回放引擎
调用位置	
输入参数	无
返回值	无
示例	tscom_tslog_stop_online_replays();

101. tslog_get_online_replay_status

函数名称	int tslog_get_online_replay_status(const s32 AIndex, TLIBOnlineReplayStatus* AStatus, float* AProgressPercent100);
功能介绍	通过引擎索引获取指定在线重播引擎的重播状态
调用位置	
输入参数	AIndex ： 重放引擎索引 AStatus ： orsNotStarted=0:重放引擎尚未启动；orsRunning=1：回放引擎正在运行； AProgressPercent100 ： 处理百分比
返回值	==0：执行成功 其他值：查询错误码
示例	TLIBOnlineReplayStatus stat; float percent100; tscom_tslog_get_online_replay_status(0, &stat, &percent100);

102. tslog_blf_write_start

函数名称	int tslog_blf_write_start(char* AFileName, ps32 AHandle); //stdcall
功能介绍	创建一个用于写入的 blf 文件。注意：必须在退出小程序之前调用 tslog_write_end 函数停止写入 blf 文件，否则会造成写入数据丢失。
调用位置	必须在退出小程序之前调用 tslog_write_end 函数停止写入 blf 文件
输入参数	AFileName ：blf 文件路径 AHandle ： 新创建的 blf 文件句柄
返回值	==0：执行成功 其他值：查询错误码

示例	// 输入一个完整的 blf 绝对路径
	s32 h;
	if (tscom_tslog_blf_write_start("C:\\LC_Ramdisk\\log1.blf", &h)){
	log("blf 文件创建成功");
	}
	
	// 如果没有使用"set_default_output_dir"设定默认输出路径时，这里输入相对于工程根目录的相对路径
	// 如果使用"set_default_output_dir"设定默认输出路径时，这里输入相对于 set_default_output_dir 的相对路径
	s32 h;
	if (tscom_tslog_blf_write_start("..\\blf2.blf", &h)){
	log("blf 文件创建成功");
	}

103. tslog_blf_write_set_max_count

函数名称	int tslog_blf_write_set_max_count(s32 AHandle, u32 ACount);
功能介绍	在切换到新文件之前设置最大写次数，如果写次数超过此限制，则创建新的 BLF 文件
调用位置	
输入参数	AHandle ：文件句柄 ACount ：范围内的最大计数[1000,4294967295]
返回值	==0：执行成功 其他值：查询错误码

示例	s32 h;
	if (com.tslog_blf_write_start("C:\\LC_Ramdisk\\log1.blf", &h)){
	log("blf 文件创建成功");
	}
	if (0 == com.tslog_blf_write_set_max_count(h, 5000000)){
	log("max write count is 5000000");
	}

104. tslog_blf_write_can

函数名称	int tslog_blf_write_can(s32 AHandle, PCAN ACAN);
功能介绍	在 blf 文件中写入 CAN 报文对象
调用位置	
输入参数	AHandle ： blf 文件句柄 ACAN ： CAN 报文对象指针
返回值	==0：执行成功 其他值：查询错误码

示例	// 输入一个完整的 blf 绝对路径
	s32 h;
	if (tslog_blf_write_start("C:\\LC_Ramdisk\\log1.blf", &h)){
	log("blf 文件创建成功");
	}
	TCAN c;
	c.init_w_std_id(0x55, 8);
	if (0 ==tslog_blf_write_can(h, &c)){
	log("write CAN object successfully");
	}

105. tslog_blf_write_can_fd

函数名称	int tslog_blf_write_can_fd(s32 AHandle, PCANFD ACANFD);
功能介绍	在 blf 文件中写入 CAFD 报文对象
调用位置	
输入参数	AHandle ： blf 文件句柄 ACANFD ： CANFD 报文对象指针
返回值	==0：执行成功 其他值：查询错误码

示例	// 输入一个完整的 blf 绝对路径
	s32 h;
	if (com.tslog_blf_write_start("C:\\LC_Ramdisk\\log1.blf", &h)){
	log("blf 文件创建成功");
	}
	TCANFD cFD;
	cFD.init_w_std_id(0x55, 8);
	if (0 == com.tslog_blf_write_can_fd(h, &cFD)){
	log("write CAN FD object successfully");
	}

106. tslog_blf_write_lin

函数名称	Int tslog_blf_write_lin(s32 AHandle, PLIN ALIN);
功能介绍	在 blf 文件中写入 LIN 消息对象
调用位置	

输入参数	AHandle ： blf 文件句柄 ALIN ： LIN 报文对象指针
返回值	==0：执行成功 其他值：查询错误码

示例	// 输入一个完整的 blf 绝对路径
	s32 h;
	if (tscom_tslog_blf_write_start("C:\\LC_Ramdisk\\log1.blf", &h)){
	log("blf 文件创建成功");
	}
	TLIN c;
	c.init_w_id(0x15, 8);
	if (0 == tscom_tslog_blf_write_lin(h, &c)){
	log("write LIN object successfully");
	}

107. tslog_blf_write_realtime_comment

函数名称	Int tslog_blf_write_realtime_comment(s32 AHandle, s64 ATimeUs, char* AComment);
功能介绍	将实时注释写入日志文件
调用位置	
输入参数	AHandle ： blf 文件句柄 ATimeUs ： 时间戳（以微秒为单位） AComment ： 注释
返回值	==0：执行成功 其他值：查询错误码

示例	// 输入一个完整的 blf 绝对路径
	s32 h;
	s32 t;
	if (tslog_blf_write_start("C:\\LC_Ramdisk\\log1.blf", &h)){
	log("blf 文件创建成功");
	}
	tslog_blf_write_realtime_comment(h, t, "this is a comment");

108. tslog_blf_write_end

函数名称	Int tslog_blf_write_end(s32 AHandle);
功能介绍	保存并关闭当前打开的 blf 文件
调用位置	
输入参数	AHandle ： blf 文件句柄
返回值	==0：执行成功 其他值：查询错误码
示例	s32 h; if (tslog_blf_write_start("C:\\LC_Ramdisk\\log1.blf", &h)){ log("blf 文件创建成功");

	} tslog_blf_write_end(h); //h: handle of the blf file

109. tslog_blf_read_start

函数名称	Int tslog_blf_read_start(char* AFileName, ps32 AHandle, ps32 AObjCount);
功能介绍	打开一个 blf 文件读取内部内容
调用位置	
输入参数	AFileName ： blf 文件路径，可以是绝对路径也可以是相对于 TSMaster.exe 的相对路径 AHandle ： blf 文件句柄，传入指针，在函数内部赋予句柄值 AObjCount ： blf 文件包含的数据个数，传入指针，在函数内部赋值
返回值	==0：执行成功 其他值：查询错误码

示例	// 输入文件绝对路径
	s32 h;
	s32 readCnt; //dataObject in blf files
	if (tscom_tslog_blf_read_start("C:\\LC_Ramdisk\\log1.blf", &h, &readCnt)){
	log("blf file opened, data objects:%d",readCnt);
	}
	
	// 如果没有使用"set_default_output_dir"设定默认输出路径时，这里输入相对于工程根目录的相对路径
	// 如果使用"set_default_output_dir"设定默认输出路径时，这里输入相对于 set_default_output_dir 的相对路径
	if (tscom_tslog_blf_read_start("..\\blf2.blf", &h, &readCnt)){
	log("blf file opened, data objects:%d",readCnt);
	}

110. tslog_blf_read_status

函数名称	Int tslog_blf_read_status(s32 AHandle, ps32 AObjReadCount);
功能介绍	读取已打开 BLF 文件的状态信息
调用位置	
输入参数	AHandle ： blf 文件句柄 AObjReadCount ： 被读取到的对象数量
返回值	==0：执行成功 其他值：查询错误码

示例	// 输入文件绝对路径
	s32 h;
	s32 readCnt; //dataObject in blf files
	if (tscom_tslog_blf_read_start("C:\\LC_Ramdisk\\log1.blf", &h, &readCnt)){
	log("blf file opened, data objects:%d",readCnt);

```
    }
    s32 cnt;
    if (0 == tscom_tslog_blf_read_status(h, &cnt)){
      log("object read count = %d", cnt);
}
```

111. tslog_blf_read_object

函数名称	Int tslog_blf_read_object(s32 AHandle, ps32 AProgressedCnt, ps32 AType/*PSupportedObjType*/, PCAN ACAN,PLIN ALIN, PCANFD ACANFD);
功能介绍	从 BLF 文件中读取一个支持的对象
调用位置	
输入参数	AHandle ： blf 文件句柄 AProgressedCnt ： 已经被读取的对象数量 AType ： 指向支持的对象类型的指针，应为 TSupportedBLFObjType 类型 ACAN ： CAN 结构体指针 ALIN ： LIN 结构体指针 ACANFD ： CANFD 结构体指针
返回值	==0：执行成功 其他值：查询错误码

示例	// 输入文件绝对路径
	s32 h;
	s32 readCnt; //dataObject in blf files
	if (tscom_tslog_blf_read_start("C:\\LC_Ramdisk\\log1.blf", &h, &readCnt)){
	log("blf file opened, data objects:%d",readCnt);
	}
	TCAN c;
	TCANFD cFD;
	TLIN l;
	s32 cnt;
	TSupportedBLFObjType t;
	if (0 == tscom_tslog_blf_read_object(h, &cnt, &t, &c, &l, &cFD)){
	if (t == sotCAN){
	// a CAN object is read out
	log("a CAN object is read out ,with ID:%d",c.FIdentifier);
	} else if (t == sotCANFD){
	// a CAN FD object is read out
	log("a CANFD object is read out ,with ID:%d",cFD.FIdentifier);
	} else if (t == sotLIN){
	// a LIN object is read out
	log("a LIN object is read out ,with ID:%d",l.FIdentifier);
	}
	}}

112. tslog_blf_read_object_w_comment

函数名称	Int tslog_blf_read_object_w_comment(s32 AHandle, ps32 AProgressedCnt, ps32 AType/* PSupportedObjType*/, PCAN ACAN, PLIN ALIN, PCANFD ACANFD, Prealtime_comment_t AComment);
功能介绍	从 BLF 文件中读取一个支持的对象
调用位置	
输入参数	AHandle ： blf 文件句柄 AProgressedCnt ： 已经被读取的对象数量 AType ： TSupportedBLFObjType 类型指针 ACAN ： CAN 结构体指针 ALIN :LIN 结构体指针 ACANFD : CANFD 结构体指针 AComment :Trealtime_comment_t 结构体指针
返回值	==0：执行成功 其他值：查询错误码

示例	// 输入文件绝对路径
	s32 h;
	s32 readCnt; //dataObject in blf files
	if (tscom_tslog_blf_read_start("C:\\LC_Ramdisk\\log1.blf", &h, &readCnt)){
	log("blf file opened, data objects:%d",readCnt);
	}
	#define STR_LEN_COMMENT 1024
	TCAN c;
	TCANFD cFD;
	TLIN l;
	char s[STR_LEN_COMMENT];
	Trealtime_comment_t cmt;
	s32 cnt;
	TSupportedBLFObjType t;
	cmt.FCapacity = STR_LEN_COMMENT-1;
	cmt.FComment = s;
	if (0 == tscom_tslog_blf_read_object_w_comment(h, &cnt, &t, &c, &l, &cFD, &cmt)){
	if (t == sotCAN){
	// a CAN object is read out
	log("a CAN object is read out ,with ID:%d",c.FIdentifier);
	} else if (t == sotCANFD){
	// a CAN FD object is read out
	log("a CANFD object is read out ,with ID:%d",cFD.FIdentifier);
	} else if (t == sotLIN){
	// a LIN object is read out

```
            log("a LIN object is read out ,with ID:%d",l.FIdentifier);
        } else if (t == sotRealtimeComment){
            // a realtime comment is read out
            log("comment read: %s", cmt.FComment);
        }
}
```

113. tslog_blf_read_end

函数名称	Int tslog_blf_read_end(s32 AHandle);
功能介绍	关闭打开的 BLF 文件
调用位置	
输入参数	AHandle ： blf 文件句柄
返回值	==0：执行成功 其他值：查询错误码

示例	// 输入文件绝对路径
	s32 h;
	s32 readCnt; //dataObject in blf files
	if (tscom_tslog_blf_read_start("C:\\LC_Ramdisk\\log1.blf", &h, &readCnt)){
	log("blf file opened, data objects:%d",readCnt);
	}
	tscom_tslog_blf_read_end(h); //h: handle of the blf file

114. tslog_blf_seek_object_time

函数名称	Int tslog_blf_seek_object_time(s32 AHandle, const double AProg100, s64* ATime, ps32 AProgressedCnt);
功能介绍	获取接近指定文件对象比率的时间戳
调用位置	
输入参数	AHandle ： blf 文件句柄 AProg100 ： blf 文件对象百分比 ATime ： 时间戳（以微秒为单位） AProgressedCnt ： 已读取的对象数
返回值	==0：执行成功 其他值：查询错误码

示例	s32 h;
	// to find timestamp near 75% of the blf file
	s64 t;
	s32 cnt;
	if (0 == com.tslog_blf_seek_object_time(h, 75, &t, &cnt)){
	log("timestamp near 75%% is %ld", t);
	}

115. tsapp_transmit_flexray_async

函数名称	Int tsapp_transmit_flexray_async(const PFlexRay AFlexray);
功能介绍	异步传输 flexray 报文
调用位置	
输入参数	AFlexray :flexray 报文指针
返回值	==0：执行成功 其他值：查询错误码

示例	TFlexRay fme;
	// ...
	if (0 == tscom_transmit_flexray_async(&fme)){
	// transmit success
	}

116. tscom_flexray_rbs_set_signal_value_by_address

函数名称	Int tscom_flexray_rbs_set_signal_value_by_address(const char* AAdrr,const double value);
功能介绍	通过数据库地址设置 FlexRay RBS 的信号值
调用位置	
输入参数	AAdrr ： 符号地址，可通过双击所选符号的状态栏复制到 FlexRay 数据库选择器中 value ： 信号实时值
返回值	==0：执行成功 其他值：查询错误码
示例	// Note: symbol address has the following structure: // channel_index/network_name/node_name/message_name/signal_name tscom_flexray_rbs_set_signal_value_by_address("0/PowerTrain/Engine/EngineData/EngForce", 12.3);

117. tscom_flexray_rbs_get_signal_value_by_address

函数名称	Int tscom_flexray_rbs_get_signal_value_by_address(const char* AAdrr, pdouble value);
功能介绍	通过数据库地址从 FlexRay RBS 获取信号值
调用位置	
输入参数	AAdrr ： 符号地址，可通过双击所选符号的状态栏复制到 FlexRay 数据库选择器中 value ： 信号实时值
返回值	==0：执行成功 其他值：查询错误码

示例	// Note: symbol address has the following structure: // channel_index/network_name/node_name/message_name/signal_name double d; tscom_flexray_rbs_get_signal_value_by_address("0/PowerTrain/Engine/EngineData/EngForce", &d);

118. tscom_flexray_rbs_set_signal_value_by_element

函数名称	Int tscom_flexray_rbs_set_signal_value_by_element(const s32 AIdxChn,const char* AClusterName, const char* AECUName, const char* AFrameName,const char* ASignalName, const double value);
功能介绍	使用元素名称设置 FlexRay RBS 的信号实时值
调用位置	
输入参数	AIdxChn ： 通道索引 AClusterName ： 包含此信号的组名 AECUName ：包含此信号的 ecu 名称 AFrameName ： 包含此信号的报文名称 ASignalName ： 信号名称 value ：信号的实值
返回值	==0：执行成功 其他值：查询错误码
示例	// Address: 0/PowerTrain/Engine/EngineData/EngForce tscom_flexray_rbs_set_signal_value_by_element(0, "PowerTrain", "Engine", "EngineData", "EngForce", 12.3);

119. tscom_flexray_rbs_get_signal_value_by_element

函数名称	Int tscom_flexray_rbs_get_signal_value_by_element(const s32 AIdxChn,const char* AClusterName, const char* AECUName, const char* AFrameName, const char* ASignalName, pdouble value);
功能介绍	使用元素名称设置 FlexRay RBS 的信号实时值
调用位置	
输入参数	AIdxChn ： 通道索引 AClusterName ： 包含此信号的组名 AECUName ； 包含此信号的 ecu 名称 AFrameName ： 包含此信号的报文名称

	ASignalName ： 信号名称 value ：信号实值
返回值	==0：执行成功 其他值：查询错误码
示例	double d; // Address: 0/PowerTrain/Engine/EngineData/EngForce tscom_flexray_rbs_get_signal_value_by_element(CH1, "PowerTrain", "Engine", "EngineData", "EngForce", &d);

120. tscom_flexray_rbs_start

函数名称	Void tscom_flexray_rbs_start();
功能介绍	启动 FlexRay RBS 引擎
调用位置	
输入参数	无
返回值	无
示例	tscom_flexray_rbs_start();

121. tscom_flexray_rbs_stop();

函数名称	Void tscom_flexray_rbs_stop();
功能介绍	停止 FlexRay RBS 引擎
调用位置	
输入参数	无
返回值	无
示例	tscom_flexray_rbs_stop();

122. tscom_flexray_rbs_is_running

函数名称	Int tscom_flexray_rbs_is_running(bool* AIsRunning);
功能介绍	检查 FlexRay RBS 是否正在运行
调用位置	
输入参数	AIsRunning ：是否运行
返回值	==0：执行成功 其他值：查询错误码
示例	bool isRunning = true; tscom_flexray_rbs_is_running(&isRunning);

123. tscom_flexray_rbs_configure

函数名称	Int tscom_flexray_rbs_configure(const bool AAutoStart,const bool

	AAutoSendOnModification, const bool AActivateECUSimulation,const TLIBRBSInitValueOptions AInitValueOptions);
功能介绍	检查 FlexRay RBS 是否正在运行
调用位置	
输入参数	AAutoStart ：连接应用程序时 RBS 是否自动启动 AAutoSendOnModification ： 如果其信号被修改，是否自动发送修改后的帧 AActivateECUSimulation： 是否激活 ecu 仿真 AInitValueOptions ：TLIBRBSInitValueOptions 类型指针
返回值	==0：执行成功 其他值：查询错误码
示例	// typedef enum {rivUseDB = 0, rivUseLast, rivUse0} TLIBRBSInitValueOptions; tscom_flexray_rbs_configure(true, true, true, rivUseDB);

124. tscom_flexray_rbs_activate_all_clusters

函数名称	Int tscom_flexray_rbs_activate_all_clusters(const bool AEnable, const bool AIncludingChildren);
功能介绍	设置激活或停用所有 flexray rbs
调用位置	
输入参数	AEnable ：是否激活 AIncludingChildren：是否所有 ecu 和报文
返回值	==0：执行成功 其他值：查询错误码
示例	tscom_flexray_rbs_activate_all_clusters(true, true);

125. tscom_flexray_rbs_activate_cluster_by_name

函数名称	Int tscom_flexray_rbs_activate_cluster_by_name(const s32 AIdxChn, const bool AEnable, const char* AClusterName, const bool AIncludingChildren);
功能介绍	通过 network name 激活 network，是否包括子节点
调用位置	
输入参数	AIdxChn ：通道索引 AEnable ： 是否激活指定的组 AClusterName ： 要激活或停用的组名 AIncludingChildren ：是否激活所有 ecu 和报文
返回值	==0：执行成功 其他值：查询错误码
示例	tscom_flexray_rbs_activate_cluster_by_name(CH1, true, "Cluster1", true);

126. tscom_flexray_rbs_activate_ecu_by_name

函数名称	Int tscom_flexray_rbs_activate_ecu_by_name(const s32 AIdxChn, const bool AEnable, const char* AClusterName, const char* AECUName, const bool AIncludingChildren);
功能介绍	通过 node name 激活 node 是否包括子节点
调用位置	
输入参数	AIdxChn :通道索引 AEnable : 是否激活指定的 ecu AClusterName : 包含此 ecu 的组名 AECUName：停用或激活的 ecu 名称 AIncludingChildren : 是否激活所有报文
返回值	==0：执行成功 其他值：查询错误码
示例	tscom_flexray_rbs_activate_ecu_by_name(CH1, true, "Cluster1", "ECU1", true);

127. tscom_flexray_rbs_activate_frame_by_name

函数名称	Int tscom_flexray_rbs_activate_frame_by_name(const s32 AIdxChn, const bool AEnable, const char* AClusterName, const char* AECUName, const char* AFrameName);
功能介绍	通过报文名激活报文
调用位置	
输入参数	AIdxChn: 通道编号 AEnable : 是否激活指定报文 AClusterName : 包含此 ecu 和报文的组名 AECUName : 包含此报文的 ecu 名称 AFrameName : 报文名称
返回值	==0：执行成功 其他值：查询错误码
示例	tscom_flexray_rbs_activate_frame_by_name(CH1, true, "Network1", "ECU1", "Message1");

128. tscom_flexray_rbs_enable

函数名称	Int tscom_flexray_rbs_enable(const bool AEnable);
功能介绍	临时启用或禁用 FlexRay RBS 引擎， 该功能用于在 RBS 引擎启动前发布配置
调用位置	该功能用于在 RBS 引擎启动前发布配置

输入参数	AEnable ： 是否启用
返回值	==0：执行成功 其他值：查询错误码

示例	// Disable message transmission after rbs_start
	tscom_flexray_rbs_enable(false);
	tscom_flexray_rbs_start();
	tscom_flexray_rbs_set_signal_value_by_address(
	"0/PowerTrain/Engine/EngineData/EngForce", 1.234);
	tscom_flexray_rbs_enable(true);
	// messages are transmitted after enabling RBS

129. tsapp_get_system_constant_count

函数名称	Int tsapp_get_system_constant_count(s32 AIdxType, ps32 ACount);
功能介绍	获取特定系统常量的计数
调用位置	
输入参数	AIdxType :0：内部常量，1：用户常量，2：系统生成常量 ACount :指定类型的计数
返回值	==0：执行成功 其他值：查询错误码

示例	s32 i;
	if (0 == tsapp_get_system_constant_count(0, &i)){
	log("internal system constant count = %d", i);
	}

130. tsapp_get_system_constant_value_by_index

函数名称	Int tsapp_get_system_constant_value_by_index(s32 AIdxType, s32 AIdxValue,char** AName,pdouble AValue,char**ADesc);
功能介绍	通过指定索引来获取系统常量的名称、描述和值
调用位置	
输入参数	AIdxType :0：内部常量，1：用户常量，2：系统生成常量 AIdxValue ： 系统常量索引 AName ： 系统常量名称 AValue :系统常量值 ADesc ： 系统常量描述
返回值	==0：执行成功 其他值：查询错误码
示例	char* sName; char* sDesc;

```
        double v;
        s32 i = 0;
        if (0 == tsapp_get_system_constant_value_by_index(0, i, &sName, &v, &sDesc)){
            log("Sytem constant %d: %s = %f, %s", i, sName, v, sDesc);
}
```

131. tsapp_log

函数名称	int tsapp_log(const char* AStr, const TLogLevel ALevel)
功能介绍	TSMaster 程序日志
调用位置	程序任意位置
输入参数	AStr:字符串 ALevel:数量
返回值	==0：执行成功 其他值：查询错误码
示例	String Str = “”; int level = 0; tsapp_log(Str, Level)

132. tsfifo_enable_receive_error_frames

函数名称	void tsfifo_enable_receive_error_frames()
功能介绍	启用接收错误帧
调用位置	
输入参数	无
返回值	无
示例	tsfifo_enable_receive_error_frames();

133. tsfifo_disable_receive_error_frames

函数名称	void tsfifo_disable_receive_error_frames()
功能介绍	禁用接收错误帧
调用位置	
输入参数	无
返回值	无
示例	tsfifo_disable_receive_error_frames();

134. tsfifo_disable_receive_fifo

函数名称	void tsfifo_disable_receive_fifo();
功能介绍	关闭驱动内部的接收缓存机制
调用位置	用户不需要采用读取缓存的方式接收报文的时候，调用此函数关闭内部缓存。
输入参数	无
返回值	无

示例	tsfifo_disable_receive_fifo();

135. tsfifo_read_can_rx_buffer_frame_count

函数名称	int tsfifo_read_canfd_rx_buffer_frame_count(const s32 AIdxChn, ps32 ACount)
功能介绍	读取 can 接收缓冲区的数据计数
调用位置	加载数据库之后，任意位置都可以调动此函数
输入参数	AIdxChn:通道索引 ACount:报文数量指针
返回值	==0：执行成功 其他值：查询错误码
示例	int ACount = 0; tsfifo_read_canfd_rx_buffer_frame_count(APP_CHANNEL.CHN1, &ACount)

136. tsfifo_read_can_tx_buffer_frame_count

函数名称	int tsfifo_read_can_tx_buffer_frame_count(const s32 AIdxChn, ps32 ACount)
功能介绍	读取 can 传输缓冲区报文计数
调用位置	加载数据库后
输入参数	AIdxChn:通道索引 ACount:报文数量指针
返回值	==0：执行成功 其他值：查询错误码
示例	int ACount = 0; tsfifo_read_can_tx_buffer_frame_count(APP_CHANNEL.CHN1,&ACount);

137. tsfifo_read_canfd_rx_buffer_frame_count

函数名称	int tsfifo_read_canfd_rx_buffer_frame_count(const s32 AIdxChn, ps32 ACount)
功能介绍	读取 canfd 接收缓冲区的数据计数
调用位置	加载数据库之后，任意位置都可以调动此函数
输入参数	AIdxChn:通道索引 ACount:报文数量指针
返回值	==0：执行成功 其他值：查询错误码
示例	int ACount = 0; tsfifo_read_canfd_rx_buffer_frame_count(APP_CHANNEL.CHN1,&ACount)

138. tsfifo_read_canfd_tx_buffer_frame_count

函数名称	int tsfifo_read_canfd_buffer_frame_count(const s32 AIdxChn, ps32 ACount)
功能介绍	读取 canfd 传输缓冲区报文数量
调用位置	加载数据库后

输入参数	AIdxChn:通道索引 ACount:报文数量指针
返回值	==0：执行成功 其他值：查询错误码
示例	int ACount = 0; tsfifo_read_canfd_buffer_frame_count(APP_CHANNEL.CHN1, &ACount)

139. tsfifo_read_fastlin_tx_buffer_frame_count

函数名称	int tsfifo_read_fastlin_tx_buffer_frame_count(const s32 AIdxChn, ps32 ACount)
功能介绍	读取快速的 lin 缓冲区数据计数
调用位置	加载数据库后
输入参数	AIdxChn:通道索引 ACount:报文数量指针
返回值	==0：执行成功 其他值：查询错误码
示例	Int ACount = 0; tsfifo_read_fastlin_tx_buffer_frame_count(APP_CHANNEL.CHN1, &ACount)

140. tsfifo_read_lin_rx_buffer_frame_count

函数名称	int tsfifo_read_lin_rx_buffer_frame_count(const s32 AIdxChn, ps32 ACount)
功能介绍	读取 lin 接收缓冲区的报文帧数
调用位置	加载数据库之后，任意位置都可以调动此函数
输入参数	AIdxChn:通道索引 ACount:报文数量指针
返回值	==0：执行成功 其他值：查询错误码
示例	int ACount = 0; tsfifo_read_lin_rx_buffer_frame_count(APP_CHANNEL.CHN1, &ACount)

141. tsfifo_read_lin_tx_buffer_frame_count

函数名称	int tsfifo_read_can_tx_buffer_frame_count(const s32 AIdxChn, ps32 ACount)
功能介绍	读取 lin 传输缓冲区报文数量
调用位置	加载数据库后
输入参数	AIdxChn:通道索引 ACount:报文数量指针
返回值	==0：执行成功 其他值：查询错误码
示例	Int ACount = 0; tsfifo_read_can_tx_buffer_frame_count(APP_CHANNEL.CHN1, &ACount)

142. tsapp_register_event_canfd

函数名称	int tsapp_register_event_canfd(const ps32 AObj, const TCANFDEvent AEvent)
功能介绍	注册 canfd 报文 rx_tx 事件
调用位置	
输入参数	AObj:唯一句柄，可以用作标识符 AEvent:处理接收报文的回调函数
返回值	==0：注册成功 其他值：注册失败
示例	int obj = 0; tsapp_unregister_event_canfd(&Obj, vCANFDQueueEventObj);

143. tsapp_register_event_lin

函数名称	int tsapp_register_event_lin(const ps32 AObj, const TLINEvent AEvent)
功能介绍	注册 lin 报文 rx_tx 事件
调用位置	
输入参数	AObj:唯一句柄，可以用作标识符 AEvent:处理接收报文的回调函数
返回值	==0：执行成功 其他值：查询错误码
示例	int obj = 0; tsapp_register_event_lin(&Obj, vLINQueueEventObj)

144. tsapp_register_pretx_event_can

函数名称	int tsapp_register_pretx_event_can(const ps32 AObj, const TCANEvent AEvent)
功能介绍	注册 can 预发送事件
调用位置	
输入参数	AObj:唯一句柄，可以用作标识符 AEvent:处理接受报文的委托函数
返回值	==0：执行成功 其他值：查询错误码
示例	int obj = 0; tsapp_register_pretx_event_can(&Obj, vCANQueueEventObj)

145. tsapp_register_pretx_event_canfd

函数名称	int tsapp_register_pretx_event_canfd(const ps32 AObj, const TCANFDEvent AEvent)
功能介绍	注册 canfd 预发送事件
调用位置	
输入参数	AObj:唯一句柄，可以用作标识符 AEvent:处理接收报文的回调函数

返回值	==0:执行成功 其他值：查询错误码
示例	int obj = 0; tsapp_register_pretx_event_canfd(&Obj, vCANFDQueueEventObj)

146. tsapp_set_mapping_verbose

函数名称	int tsapp_set_mapping_verbose (const char* AAppName, const TLIBApplicationChannelType AAppChannelType, const s32 AAppChannel, const char* AHardwareName, const TLIBBusToolDeviceType AHardwareType, const s32 AHardwareSubType, const s32 AHardwareIndex, const s32 AHardwareChannel, const bool AEnableMapping);
功能介绍	**使用硬件设备之前，为应用程序的通道（CAN/CANFD/LIN）映射（也就是绑定）指定 CAN 设备的指定通道。这个操作也可以在 TSMaster 的设备管理界面完成，代码中可以不需要这部分操作。关于通道映射函数，还有不清晰的地方，请查看文档：TsMasterApi HardwareMap.pdf**
调用位置	在应用程序连接 CAN 工具之前，调用此函数完成硬件的绑定。

输入参数	AAppName ：应用程序名称：该 Map 绑定到的应用程序名称，跟 initialize_lib_tsmaster 中名称要一致
	AAppChannelType:通道类型：包括 APP_CAN, APP_LIN
	AAppChannel:应用程序的通道编号：这是上层程序使用的通道编号
	AHardwareName:硬件名称：比如 TOSUN，Vector 等，也可以不填写，主要是方便用户查看
	AHardwareType:硬件类型：根据 TLIBBusToolDeviceType 囊括了目前市面上所有主流的 CAN 卡硬件
	其中，工具枚举类型定义如下：
	public enum TLIBBusToolDeviceType : int {
	BUS_UNKNOWN_TYPE = 0,
	TS_TCP_DEVICE = 1,
	XL_USB_DEVICE = 2,
	TS_USB_DEVICE = 3,
	PEAK_USB_DEVICE = 4,
	KVASER_USB_DEVICE = 5,
	ZLG_USB_DEVICE = 6,
	ICS_USB_DEVICE = 7,
	TS_TC1005_DEVICE = 8
	};
	AHardwareSubType:在该设备类型下面的子设备，比如同星的 TS_USB_DEVICE，下面就包含 TC1001，TL1002，TC1011 等等系列产品

	AHardwareIndex:支持同时插多个完全一样的硬件，比如同时插两个 TC1001，这里用于选择是 1 号设备的通道还是 2 号设备的通道 AHardwareChannel:硬件设备的通道，比如 TC1005 有 5 个硬件通道，这个参数就用于选择对应的硬件通道 AEnableMapping:是否使能此映射，如果设置为 false，则该通道是不能使用的
返回值	==0:执行成功 其他：查询错误码

示例	//如下代码：表示把TC1005设备的通道1分配给应用程序的通道1使用。这样应用程序使用通道1通讯的时候，实际上使用的就是TC1005的硬件通道1.
	if (tsapp_set_channel_mapping(
	FProgramName, //应用程序名称
	TLIBApplicationChannelType.APP_CAN, //映射CAN/CANFD通道
	APP_CHANNEL.CHN1, //映射（分配给）应用程序的通道1
	"TC1005", //设备名称为TC1005
	TLIBBusToolDeviceType.TS_TC1005_DEVICE, //设备类型为TS_TC1005_DEVICE
	0, //子类型为0，TC1005下面没有子类型，所以直接设置为0
	0, //设备编号为0，如果只接了一个设备，则设置硬件编号为0
	HARDWARE_CHANNEL.CHN1) == 0) //使用TC1005的0物理通道
	{
	}

147. tsapp_show_tsmaster_window

函数名称	int tsapp_show_tsmaster_window(const char* AWindowName, const bool AWaitClose)
功能介绍	显示 TSMaster 主窗口
调用位置	程序生命周期开始时
输入参数	AWindowName:窗口名 AWaitClose:是否等待关闭
返回值	==-0：执行成功 其他值：查询错误码
示例	tsapp_show_tsmaster_window("TSMaster", true);

148. tsapp_unregister_event_canfd

函数名称	int tsapp_unregister_event_canfd(const ps32 AObj, const TCANFDEvent AEvent)
功能介绍	注销 canfd 报文 rx_tx 事件
调用位置	
输入参数	AObj:唯一句柄，可以用作标识符 AEvent:处理接收报文的回调函数
返回值	==0：注销成功 其他值：注销失败
示例	int obj = 0;

	tsapp_unregister_event_canfd(&Obj, vCANFDQueueEventObj)

149. tsapp_unregister_event_lin

函数名称	int tsapp_unregister_event_lin(const ps32 AObj, const TLINEvent AEvent)
功能介绍	注销 lin 事件
调用位置	
输入参数	AObj:唯一句柄，可以用作标识符 AEvent:处理接收报文的回调函数
返回值	==0：执行成功 其他值：查询错误码
示例	int obj = 0; tsapp_unregister_event_lin(&Obj, vLINQueueEventObj);

150. tsapp_unregister_events_all

函数名称	int tsapp_unregister_events_all(const ps32 AObj)
功能介绍	注销所有发送_接收事件
调用位置	
输入参数	AObj:唯一句柄，可以用作标识符
返回值	==0：执行成功 其他值：查询错误码
示例	int obj = 0; tsapp_unregister_events_all(&obj);

151. tsapp_unregister_pretx_event_can

函数名称	int tsapp_unregister_pretx_event_can(const ps32 AObj, const TCANEvent AEvent)
功能介绍	注销 can 预发送事件
调用位置	
输入参数	AObj:唯一句柄，可以用作标识符 AEvent:处理接收报文的回调函数
返回值	==0:执行成功 其他值：查询错误码
示例	int obj = 0; tsapp_unregister_pretx_event_can(&Obj, vCANQueueEventObj);

152. tsapp_unregister_pretx_event_canfd

函数名称	int tsapp_unregister_pretx_event_canfd(const ps32 AObj, const TCANFDEvent AEvent)
功能介绍	注销 canfd 预发送事件
调用位置	
输入参数	AObj:唯一句柄，可以用作标识符 AEvent:处理接收报文的回调函数

返回值	==0:执行成功 其他值：查询错误码
示例	int obj = 0; tsapp_unregister_pretx_event_canfd(&Obj, vCANFDQueueEventObj);

153. tsapp_unregister_pretx_event_lin

函数名称	int tsapp_unregister_pretx_event_lin(const ps32 AObj, const TLINEvent AEvent)
功能介绍	注销 lin 预发送事件
调用位置	
输入参数	AObj:唯一句柄，可以用作标识符 AEvent:处理接收报文的回调函数
返回值	==0：执行成功 其他值：查询错误码
示例	int obj = 0; tsapp_unregister_pretx_event_lin(&Obj, vLINQueueEventObj);

154. tsapp_unregister_pretx_events_all

函数名称	int tsapp_unregister_pretx_events_all(const ps32 AObj)
功能介绍	注销所有预发送事件
调用位置	
输入参数	AObj:唯一句柄，可以用作标识符
返回值	==0：执行成功 其他值：查询错误码
示例	int obj = 0; tsapp_unregister_pretx_events_all(&Obj);

155. tslog_add_online_replay_config

函数名称	int tslog_add_online_replay_config(const char* AFileName, s32* AIndex)
功能介绍	通过日志文件创建一个在线重放配置。
调用位置	
输入参数	AFileName:blf 文件名，可以是绝对路径，也可以是相对路径 AIndex:如果成功创建此在线重放配置，则将返回内部在线重播放引擎索引
返回值	==0：执行成功 其他值：查询错误码
示例	int i = 0; tslog_add_online_replay_config("log1", &i);

156. tslog_del_online_replay_config

函数名称	int tslog_del_online_replay_config(const s32 AIndex)
功能介绍	通过发动机索引删除在线回放发动机
调用位置	

输入参数	AIndex:重放引擎索引
返回值	==0：执行成功 其他值：查询错误码

示例	// delete the first engine
	tslog_del_online_replay_config(0);

157. tsapp_register_event_flexray

函数名称	uint tsapp_register_event_flexray(ps32 obj, const TFlexRayEvent FlexrayEvent)
功能介绍	注册 flexray rx_tx 事件
调用位置	
输入参数	Obj:OnTx_RxFUNC_Flexray FlexrayEvent:处理接收报文的回调函数
返回值	==0：执行成功 其他值：查询错误码
示例	int obj = 0; tsapp_register_event_flexray(&Obj, vFlexrayQueueEventObj);

158. tsapp_register_pretx_event_flexray

函数名称	uint tsapp_register_pretx_event_flexray(ps32 obj, const TFlexRayEvent FlexrayEvent)
功能介绍	注册 flexray 预发送事件
调用位置	
输入参数	Obj:唯一句柄，可以用作标识符 FlexrayEvent:处理接收报文的回调函数
返回值	==0：执行成功 其他值：查询错误码
示例	int obj = 0; tsapp_register_pretx_event_flexray(&Obj, vFlexrayQueueEventObj);

159. tsapp_unregister_event_flexray

函数名称	uint tsapp_unregister_event_flexray(ps32 obj, const TFlexRayEvent FlexrayEvent)
功能介绍	注销 flexray rx_tx 事件
调用位置	

输入参数	Obj:OnTx_RxFUNC_Flexray FlexrayEvent:处理接收报文的回调函数
返回值	==0：执行成功 其他值：查询错误码
示例	tsapp_unregister_event_flexray(&Obj, vFlexrayQueueEventObj);

160. tsapp_unregister_pretx_event_flexray

函数名称	uint tsapp_unregister_pretx_event_flexray(ps32 obj, const TFlexRayEvent FlexrayEvent)
功能介绍	注销 flexray 预发送事件
调用位置	
输入参数	Obj:唯一句柄，可以用作标识符 FlexrayEvent:处理接收报文的回调函数
返回值	==0：执行成功 其他值：查询错误码
示例	int obj = 0; tsapp_unregister_pretx_event_flexray(&Obj, vFlexrayQueueEventObj);

161. tsapp_unregister_events_flexray

函数名称	uint tsapp_unregister_events_flexray(ps32 obj)
功能介绍	注销所有 flexray 数据库事件
调用位置	
输入参数	Obj:唯一句柄，可以用作标识符
返回值	==0：执行成功 其他值：查询错误码
示例	int obj = 0; tsapp_unregister_events_flexray(&Obj)

162. tsapp_unregister_pretx_events_flexray

函数名称	uint tsapp_unregister_pretx_events_flexray(ps32 obj)
功能介绍	注销 flexray 预发送事件
调用位置	

输入参数	Obj:唯一句柄，可以用作标识符
返回值	==0：执行成功 其他值：查询错误码
示例	int obj = 0; tsapp_unregister_pretx_events_flexray(&Obj)

163. tscom_flexray_get_signal_definition

函数名称	uint tscom_flexray_get_signal_definition(const char* ASignalAddress, PFlexRaySignal ASignalDef)
功能介绍	获取 flexray 信号定义
调用位置	
输入参数	ASignalAddress:信号地址 ASignalDef:返回的信号定义
返回值	==0：执行成功 其他值：查询错误码

示例	TFlexRaySignal sgn;
	if (0==tscom_flexray_get_signal_definition(
	"0/PowerTrain/Engine/EngineData/EngForce",
	&sgn
	)){
	log("FFRSgnType = %d", sgn.FFRSgnType);
	log("FCompuMethod = %d", sgn.FCompuMethod);
	log("FReserved = %d", sgn.FReserved);
	log("FIsIntel = %d", sgn.FIsIntel);
	log("FStartBit = %d", sgn.FStartBit);
	log("FUpdateBit = %d", sgn.FUpdateBit);
	log("FLength = %d", sgn.FLength);
	log("FFactor = %f", sgn.FFactor);
	log("FOffset = %f", sgn.FOffset);
	}

164. tscom_flexray_set_signal_value_in_raw_frame

函数名称	double tscom_flexray_set_signal_value_in_raw_frame(const PFlexRaySignal ASignal, pu8 data, double AValue)
功能介绍	在 flexray 数据库原始报文中设置信号值
调用位置	

输入参数	ASignal:flexray 信号 Data:符号地址，可以通过双击状态栏复制到 fleray 数据库选择器中 Avalue:设置的值
返回值	==0：执行成功 其他值：查询错误码
示例	TFleRaySignal fleRaySignal = new TFleRaySignal(); byte [] pdata; tscom_flexray_set_signal_value_in_raw_frame(flexRaySignal, pdata, 24)

165. tsdb_unload_flexray_db

函数名称	int tsdb_unload_flexray_db(const s32 AId)
功能介绍	卸载 flexray 数据库
调用位置	
输入参数	AId:数据库 ID
返回值	==0：执行成功 其他值：查询错误码
示例	tsdb_unload_flexray_db(0);

166. tsdb_unload_flexray_dbs

函数名称	int tsdb_unload_flexray_dbs()
功能介绍	卸载所有 flexray 数据库
调用位置	
输入参数	无
返回值	==0：执行成功 其他值：查询错误码
示例	tsdb_unload_flexray_dbs()

167. tsdb_get_flexray_db_count

函数名称	int tsdb_get_flexray_db_count(ps32 ACount)
功能介绍	获取所有加载的 flexray 数据库数量
调用位置	

输入参数	ACount:返回的数据库计数
返回值	==0：执行成功 其他值：查询错误码

示例	S32 n;
	if (0 == tsdb_get_flexray_database_count(&n)){
	log("loaded database count:%d", n);
	}

168. tsdb_get_flexray_db_id

函数名称	int tsdb_get_flexray_db_id(const s32 AIndex, ps32 AId)
功能介绍	获取加载的 flexray 数据库 ID
调用位置	
输入参数	AIndex:数据库索引 AId:返回的数据库 ID
返回值	==0：执行成功 其他值：查询错误码
示例	tsdb_get_flexray_db_id(AIndex, &AId)

169. tsdb_get_flexray_db_properties_by_address

函数名称	int tsdb_get_flexray_db_properties_by_address(const char* AAddr, PDBProperties Avalue)
功能介绍	按地址获取指定 FlexRay 数据属性
调用位置	
输入参数	AAddr:数据库地址，可以在数据库查看器中复制 Avalue:属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBProperties p = new TDBProperties();
	if(0==tsdb_get_flexray_db_properties_by_address(
	"0/network1", p)){
	// properties retrieved into p
	}

170. tsdb_get_flexray_db_properties_by_index

函数名称	int tsdb_get_flexray_db_properties_by_index(PDBProperties Avalue)
功能介绍	通过索引获取指定 FlexRay 数据库的属性

调用位置	
输入参数	Avalue:属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBFrameProperties p = new PDBFrameProperties();
	// first specify indexes and direction before calling
	p.FDBIndex = 0;
	p.FECUIndex = 0;
	p.FFrameIndex = 0;
	p.FIsTx = 1;
	if (0 == tsdb_get_flexray_db_properties_by_index(p)){
	  // properties retrieved into p
	}

171. tsdb_get_flexray_db_ecu_properties_by_address

函数名称	int tsdb_get_flexray_db_ecu_properties_by_address(const char* AAddr, PDBECUProperties Avalue)
功能介绍	通过地址获取指定 FlexRay 数据库的 ecu 属性
调用位置	
输入参数	AAddr:返回的属性数据结构，请在 TSMaster 标头中查找定义 Avalue:PDBECUProperties 属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBECUProperties p = new PDBECUProperties();
	if(0==tsdb_get_flexray_db_ecu_properties_by_address(
	"0/network1/ecu1", p)){
	  // properties retrieved into p
	}

172. tsdb_get_flexray_db_ecu_properties_by_index

函数名称	int tsdb_get_flexray_db_ecu_properties_by_index(PDBECUProperties Avalue)
功能介绍	通过索引获取指定 FlexRay 数据库的 ecu 属性
调用位置	
输入参数	Avalue:属性数据结构，请在 TSMaster 标头中查找定义

返回值	==0：执行成功 其他值：查询错误码

示例	PDBECUProperties p = new PDBECUProperties();
	// first specify indexes before calling
	p.FDBIndex = 0;
	p.FECUIndex = 0;
	if(0== tsdb_get_flexray_db_ecu_properties_by_index(p)){
	  // properties retrieved into p
	}

173. tsdb_get_flexray_db_frame_properties_by_address

函数名称	int tsdb_get_flexray_db_frame_properties_by_address(const char* AAddr, PDBFrameProperties Avalue)
功能介绍	按地址获取指定 FlexRay 数据库的报文属性
调用位置	
输入参数	AAddr:数据库地址，可以在数据库查看器中复制 Avalue:PDBFrameProperties 属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBFrameProperties p = new PDBFrameProperties();
	if(0==tsdb_get_flexray_db_frame_properties_by_address("0/network1/ecu1/frame1", p)){
	  // properties retrieved into p
	}

174. tsdb_get_flexray_db_frame_properties_by_index

函数名称	Int tsdb_get_flexray_db_frame_properties_by_index(PDBFrameProperties Avalue)
功能介绍	按索引获取指定 FlexRay 数据库的报文属性
调用位置	
输入参数	Avalue:属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBFrameProperties p = new PDBFrameProperties ();
	// first specify indexes and direction before calling
	p.FDBIndex = 0;

	p.FECUIndex = 0;
	p.FFrameIndex = 0;
	p.FIsTx = 1;
	if (0 == tsdb_get_flexray_db_frame_properties_by_index(p)){
	  // properties retrieved into p
	}

175. tsdb_get_flexray_db_signal_properties_by_db_index

函数名称	int tsdb_get_flexray_db_signal_properties_by_db_index(const s32 AIdxDB, const s32 AIndex, PDBSignalProperties Avalue)
功能介绍	通过数据库信号列表中指定的索引获取 FlexRay 数据库的信号属性
调用位置	
输入参数	AIdxDB:数据库索引 AIndex:信号索引 Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBSignalProperties sgn = new PDBSignalProperties();
	if(0==tsdb_get_flexray_db_signal_properties_by_db_index(0,0,sgn)){
	  // retrieve success
	}

176. tsdb_get_flexray_db_signal_properties_by_frame_index

函数名称	int tsdb_get_flexray_db_signal_properties_by_frame_index(const s32 AIdxDB, const s32 Frameidx ,const s32 AIndex, PDBSignalProperties Avalue)
功能介绍	通过帧索引获取 FlexRay 数据库的信号属性
调用位置	
输入参数	AIdxDB:数据库索引 FrameIdx:报文索引 AIndex:信号索引 Avalue:返回的信号属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBSignalProperties sgn = new PDBSignalProperties();
	if(0==tsdb_get_flexray_db_signal_properties_by_frame_index(0, 0, 0, sgn)){
	  // retrieve success
	}

177. tsdb_get_flexray_db_signal_properties_by_frame_index

函数名称	int tsdb_get_flexray_db_signal_properties_by_frame_index(const s32 AIdxDB, const s32 Frameidx ,const s32 AIndex, PDBSignalProperties Avalue)
功能介绍	通过帧索引获取 FlexRay 数据库的信号属性
调用位置	
输入参数	AIdxDB:数据库索引 FrameIdx:数据库信号帧索引 AIndex:帧中的信号索引 Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBSignalProperties sgn = new PDBSignalProperties();
	if(0==tsdb_get_flexray_signal_properties_by_frame_index(0, 0, 0, sgn)){
	  // retrieve success
	}

178. tsdb_load_can_db

函数名称	int tsdb_load_can_db(const char* ADBC, const char* ASupportedChannelsBased0, u32* AId)
功能介绍	把数据库加载到指定的通道上，并获取加载数据库的编号
调用位置	该函数必须在应用处于断开状态（执行 tsapp_connect 之前或者执行了 tsapp_disconnect 之后）调用。当应用处于连接状态的时候，不能删除，加载数据库文件。
输入参数	ADBC:数据库的绝对路径 ASupportedChannelsBased0:通道数组 AId:加载数据库成功过后，返回系统为该数据库分配的唯一编号
返回值	==0：执行成功 其他值：查询错误码
示例	tsdb_load_can_db(DBCpathName, chns, &DBChandle);

179. tsdb_unload_can_db

函数名称	int tsdb_unload_can_db(const u32 AId)
功能介绍	卸载指定编号的数据库文件
调用位置	该函数必须在应用处于断开状态（执行 tsapp_connect 之前或者执行了 tsapp_disconnect 之后）调用。当应用处于连接状态的时候，不能删除，加载数据库文件。
输入参数	AId:数据库 id
返回值	==0：执行成功 其他值：查询错误码
示例	tsdb_unload_can_db(0);

180. tsdb_unload_can_dbs

函数名称	int tsdb_unload_can_dbs()
功能介绍	卸载所有已经加载的 DBC 文件
调用位置	该函数必须在应用处于断开状态（执行 tsapp_connect 之前或者执行了 tsapp_disconnect 之后）调用。当应用处于连接状态的时候，不能删除，加载数据库文件。
输入参数	无
返回值	==0：执行成功 其他值：查询错误码
示例	tsdb_unload_can_dbs();

181. tsdb_get_can_db_count

函数名称	int tsdb_get_can_db_count(s32* ACount)
功能介绍	获取所有加载的 CAN 数据库数量
调用位置	
输入参数	ACount:返回 CAN 数据库数量
返回值	==0：执行成功 其他值：查询错误码
示例	S32 ACount; tsdb_get_can_db_count(&ACount)

182. tsdb_get_can_db_id

函数名称	int tsdb_get_can_db_id(const s32 AIndex, u32* AId)
功能介绍	通过索引获取 CAN 数据库 id
调用位置	
输入参数	AIndex:索引 AId:CAN 数据库 ID
返回值	==0：执行成功 其他值：查询错误码
示例	tsdb_get_can_db_id(0, &AId)

183. tsdb_get_can_db_info

函数名称	string tsdb_get_can_db_info(const u32 ADatabaseId, const s32 AType, const s32 AIndex, const s32 ASubIndex, char** AValue)

功能介绍	获取 CAN 数据库信息
调用位置	
输入参数	ADatabaseId:数据库 ID AType:类型 AIndex:数据库索引 ASubIndex:CAN 索引 Avalue:实值
返回值	==0：执行成功 其他值：查询错误码

示例	String DBCpathName;
	int DBChandle =0;
	Int ret = tsdb_load_can_db(DBCpathName, chns, &DBChandle);
	If(ret == 0)
	{
	Int canCount = tsdb_get_can_db_info(DBChandle,11,0,0);
	}

184. tsdb_get_can_db_properties_by_address

函数名称	int tsdb_get_can_db_properties_by_address(const char* AAddr, PDBProperties Avalue)
功能介绍	通过地址获取指定 CAN 数据库的属性
调用位置	
输入参数	AAddr:数据库地址，可以在数据库查看器中复制 Avalue:返回的数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBProperties p = new PDBProperties();
	if(0== tsdb_get_can_db_properties_by_address("0/network1", p)){
	// properties retrieved into p
	}

185. tsdb_get_can_db_properties_by_index

函数名称	int tsdb_get_can_db_properties_by_index(PDBProperties Avalue)
功能介绍	通过索引获取指定 CAN 数据库的属性
调用位置	
输入参数	Avalue:返回的属性数据结构，请在 TSMaster 头中找到定义

返回值	==0：执行成功 其他值：查询错误码

示例	PDBProperties p = new PDBProperties();
	// first specify indexes before calling
	p.FDBIndex = 0;
	if (0 == tsdb_get_can_db_properties_by_index(p)){
	// properties retrieved into p
	}

186. tsdb_get_can_db_ecu_properties_by_address

函数名称	int tsdb_get_can_db_ecu_properties_by_address(const char* AAddr, PDBECUProperties Avalue)
功能介绍	通过地址获取指定 CAN 数据库的 ECU 属性
调用位置	
输入参数	AAddr:数据库地址，可以在数据库查看器中复制 Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBECUProperties p = new PDBECUProperties();
	// first specify indexes before calling
	p.FDBIndex = 0;
	p.FECUIndex = 0;
	if (0 == tsdb_get_can_db_=ecu_properties_by_index("0/network1/ecu1/frame1",p){
	// properties retrieved into p
	}

187. tsdb_get_can_db_ecu_properties_by_index

函数名称	int tsdb_get_can_db_ecu_properties_by_index(PDBECUProperties Avalue)
功能介绍	通过索引获取指定 CAN 数据库的 ECU 属性
调用位置	
输入参数	Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码
示例	PDBECUProperties p = new PDBECUProperties();

	if(0==tsdb_get_can_db_ecu_properties_by_address("0/network1/ecu1", p)){
	// properties retrieved into p
	}

188. tsdb_get_can_db_frame_properties_by_db_index

函数名称	int tsdb_get_can_db_frame_properties_by_db_index(const s32 AIdxDB, const s32 AIndex, PDBFrameProperties Avalue)
功能介绍	通过数据库帧列表中指定的索引获取 CAN 数据库报文属性
调用位置	
输入参数	AIdxDB:数据库索引 AIndex:数据库报文列表索引 Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBFrameProperties fme = new PDBFrameProperties();
	if(0== db_get_can_frame_properties_by_db_index(0,0,fme)){
	// retrieve success
	}

189. tsdb_get_can_db_frame_properties_by_address

函数名称	int tsdb_get_can_db_frame_properties_by_address(const char* AAddr, PDBFrameProperties Avalue)
功能介绍	通过地址获取指定 CAN 数据库的帧属性
调用位置	
输入参数	AAddr:数据库地址，可以在数据库查看器中复制 Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBFrameProperties p = new PDBFrameProperties();
	if (0 == tsdb_get_can_db_frame_properties_by_address("0/network1/ecu1/frame1", p)){
	// properties retrieved into p
	}

190. tsdb_get_can_db_frame_properties_by_index

函数名称	int tsdb_get_can_db_frame_properties_by_index(PDBFrameProperties Avalue)

功能介绍	通过地址获取指定 CAN 数据库的帧属性
调用位置	
输入参数	Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBFrameProperties fme = new PDBFrameProperties();
	if (0 == tsdb_get_can_db_frame_properties_by_db_index(fme)){
	  // retrieve success
	}

191. tsdb_get_can_db_signal_properties_by_db_index

函数名称	int tsdb_get_can_db_signal_properties_by_db_index(const s32 AIdxDB, const s32 AIndex, PDBSignalProperties Avalue)
功能介绍	通过数据库信号列表中指定的索引获取 CAN 信号属性
调用位置	
输入参数	AIdxDB:数据库索引 AIndex:数据库信号列表索引 Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBSignalProperties sgn = new PDBSignalProperties();
	if(0== tsdb_get_can_db_signal_properties_by_db_index(0,0,sgn)){
	  // retrieve success
	}

192. tsdb_get_can_db_signal_properties_by_frame_index

函数名称	int tsdb_get_can_db_signal_properties_by_frame_index(const s32 AIdxDB, const s32 Frameidx, const s32 AIndex, PDBSignalProperties Avalue)
功能介绍	通过报文索引获取 CAN 信号属性
调用位置	
输入参数	AIdxDB:数据库索引 FrameIdx:数据库帧索引 AIndex:数据库信号列表索引 Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码
示例	PDBSignalProperties sgn = new PDBSignalProperties();

	if(0==tsdb_get_can_db_signal_properties_by_frame_index(0, 0, 0, sgn)){
	// retrieve success
	}

193. tsdb_get_can_db_signal_properties_by_address

函数名称	int tsdb_get_can_db_signal_properties_by_address(const char* AAddr, PDBSignalProperties Avalue)
功能介绍	通过地址获取指定 CAN 数据库的信号属性
调用位置	
输入参数	AAddr:数据库地址，可以在数据库查看器中复制 Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBSignalProperties p = new PDBSignalProperties();
	if (0 == tsdb_get_can_db_signal_properties_by_address("0/network1/ecu1/frame1/Signal1", p)){
	// properties retrieved into p
	}

194. tsdb_get_can_db_signal_properties_by_index

函数名称	Inttsdb_get_can_db_signal_properties_by_index(PDBSignalProperties Avalue)
功能介绍	通过索引获取指定 CAN 数据库的信号属性
调用位置	
输入参数	Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBSignalProperties p = new PDBSignalProperties();
	// first specify indexes and direction before calling
	p.FDBIndex = 0;
	p.FECUIndex = 0;
	p.FFrameIndex = 0;
	p.FSignalIndex = 0;
	p.FIsTx = 1;
	if (0 == tsdb_get_can_db_signal_properties_by_index(p)){
	// properties retrieved into p
	}

195. tsdb_load_lin_db

函数名称	int tsdb_load_lin_db(const char* ADBC, const char* ASupportedChannelsBased0, u32* AId)
功能介绍	加载 lin 数据库
调用位置	
输入参数	ADBC:DBC 绝对路径 ASupportedChannelsBased0:绑定的通道数组 AId:加载数据库成功过后，返回系统为该数据库分配的唯一编号
返回值	==0：执行成功 其他值：查询错误码
示例	U32 AId = 0; const char* ASupportedChannels = ""; tsdb_load_lin_db(ADBCAbsolutePath ,ASupportedChannelsBased0 , &AId);

196. tsdb_unload_lin_db

函数名称	int tsdb_unload_lin_db(const u32 AId)
功能介绍	通过 ID 卸载已加载的 lin 数据库
调用位置	
输入参数	AId:返回的 LIN 数据库 ID
返回值	==0：执行成功 其他值：查询错误码
示例	tsdb_unload_lin_db(0)

197. tsdb_unload_lin_dbs

函数名称	int tsdb_unload_lin_dbs()
功能介绍	卸载所有已加载的 lin 数据库
调用位置	
输入参数	无
返回值	==0：执行成功 其他值：查询错误码
示例	tsdb_unload_lin_dbs();

198. tsdb_get_lin_db_count

函数名称	int tsdb_get_lin_db_count(s32* ACount)
功能介绍	获取所有加载的 LIN 数据库数量
调用位置	
输入参数	ACount:返回的数据库计数
返回值	==0：执行成功 其他值：查询错误码

示例	int n = 0;
	if (0 == tsdb_get_lin_db_count(&n)){
	log("loaded database count :%d"n);
	}

199. tsdb_get_lin_db_id

函数名称	int tsdb_get_lin_db_id(const s32 AIndex, u32* AId)
功能介绍	获取 LIN 数据库 ID
调用位置	
输入参数	AIndex:数据库索引 AId:返回的数据库 ID
返回值	==0：执行成功 其他值：查询错误码
示例	uint AId = 0; tsdb_get_lin_db_id(0, &AId)

200. tsdb_get_lin_db_properties_by_address

函数名称	int tsdb_get_lin_db_properties_by_address(const char* AAddr, PDBProperties Avalue)
功能介绍	通过地址获取指定 LIN 数据库属性
调用位置	
输入参数	AAddr:数据库地址 Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功

	其他值：查询错误码

示例	PDBProperties p = new PDBProperties();
	if(0==tsdb_get_lin_db_properties_by_address("0/network1", p)){
	  // properties retrieved into p
	}

201. tsdb_get_lin_db_properties_by_index

函数名称	int tsdb_get_lin_db_properties_by_index(PDBProperties Avalue)
功能介绍	通过索引获取指定 LIN 数据库的属性
调用位置	
输入参数	Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBProperties p = new PDBProperties();
	// first specify indexes before calling
	p.FDBIndex = 0;
	if (0 == tsdb_get_lin_db_properties_by_index(p)){
	  // properties retrieved into p
	}

202. tsdb_get_lin_db_ecu_properties_by_address

函数名称	int tsdb_get_lin_db_ecu_properties_by_address(const char* AAddr, PDBECUProperties Avalue)
功能介绍	通过地址获取指定 LIN 数据库的 ECU 属性
调用位置	
输入参数	AAddr:数据库地址 Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBECUProperties p = new PDBECUProperties();
	if(0==tsdb_get_lin_db_ecu_properties_by_address("0/network1/ecu1", p)){
	  // properties retrieved into p
	}

203. tsdb_get_lin_db_ecu_properties_by_index

函数名称	int tsdb_get_lin_db_ecu_properties_by_index(PDBECUProperties Avalue)
功能介绍	通过索引获取指定 LIN 数据库的 ECU 属性
调用位置	
输入参数	Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBECUProperties p = new PDBECUProperties();
	// first specify indexes before calling
	p.FDBIndex = 0;
	p.FECUIndex = 0;
	if (0 == tsdb_get_lin_db_ecu_properties_by_index(p)){
	  // properties retrieved into p
	}

204. tsdb_get_lin_db_frame_properties_by_db_index

函数名称	int tsdb_get_lin_db_frame_properties_by_db_index(const s32 AIdxDB, const s32 AIndex, PDBFrameProperties Avalue)
功能介绍	通过数据库索引获取指定 LIN 数据库的帧属性
调用位置	
输入参数	AIdxDB:数据库索引 AIndex:索引 Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBFrameProperties fme = new PDBFrameProperties();
	if(0==tsdb_get_lin_db_frame_properties_by_db_index(0,0,fme)){
	  // retrieve success
	}

205. tsdb_get_lin_db_frame_properties_by_address

函数名称	int tsdb_get_lin_db_frame_properties_by_address(const char* AAddr, PDBFrameProperties Avalue)
功能介绍	通过地址获取指定 LIN 数据库的帧属性
调用位置	

输入参数	AAddr:数据库地址 Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBFrameProperties p = new PDBFrameProperties ();
	if(0==tsdb_get_lin_db_frame_properties_by_address(
	"0/network1/ecu1/frame1", p)){
	  // properties retrieved into p
	}

206. tsdb_get_lin_db_frame_properties_by_index

函数名称	int tsdb_get_lin_db_frame_properties_by_index(PDBFrameProperties Avalue)
功能介绍	通过索引获取指定 LIN 数据库的帧属性
调用位置	
输入参数	Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBFrameProperties p = new PDBFrameProperties();
	// first specify indexes and direction before calling
	p.FDBIndex = 0;
	p.FECUIndex = 0;
	p.FFrameIndex = 0;
	p.FIsTx = 1;
	if (0 == tsdb_get_lin_db_frame_properties_by_index(p)){
	  // properties retrieved into p
	}

207. tsdb_get_lin_db_signal_properties_by_db_index

函数名称	int tsdb_get_lin_db_signal_properties_by_db_index(const s32 AIdxDB, const s32 AIndex, PDBSignalProperties Avalue)
功能介绍	通过数据库索引获取指定 LIN 数据库的信号属性
调用位置	
输入参数	AIdxDB:数据库索引 AIndex:索引 Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBSignalProperties sgn = new PDBSignalProperties();
	if(0==tsdb_get_lin_db_signal_properties_by_db_index(0,0,sgn)){
	// retrieve success
	}

208. tsdb_get_lin_db_signal_properties_by_frame_index

函数名称	int tsdb_get_lin_db_signal_properties_by_frame_index(const s32 AIdxDB, const s32 Frameidx, const s32 AIndex, PDBSignalProperties Avalue)
功能介绍	通过帧索引获取指定 LIN 数据库的信号属性
调用位置	
输入参数	AIdxDB:数据库索引 FrameIdx:数据库帧索引 AIndex:帧中信号索引 Avalue:返回的属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBSignalProperties sgn;
	if(0==tsdb_get_lin_db_signal_properties_by_frame_index(0, 0, 0, sgn)){
	// retrieve success
	}

209. tsdb_get_lin_db_signal_properties_by_address

函数名称	int tsdb_get_lin_db_signal_properties_by_address(const char* AAddr, PDBSignalProperties Avalue)
功能介绍	通过地址获取指定 LIN 数据库的信号属性
调用位置	
输入参数	AAddr:数据库地址 Avalue:返回的信号属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBSignalProperties p = new PDBSignalProperties() ;
	if(0==tsdb_get_lin_signal_properties_by_address(
	"0/network1/ecu1/frame1/
	Signal1",p)){
	// properties retrieved into p
	}

210. tsdb_get_lin_db_signal_properties_by_index

函数名称	int tsdb_get_lin_db_signal_properties_by_index(PDBSignalProperties Avalue)
功能介绍	通过索引获取指定 LIN 数据库的信号属性
调用位置	
输入参数	Avalue:属性数据结构，请在 TSMaster 标头中查找定义
返回值	==0：执行成功 其他值：查询错误码

示例	PDBSignalProperties p = new PDBSignalProperties();
	// first specify indexes and direction before calling
	p.FDBIndex = 0;
	p.FECUIndex = 0;
	p.FFrameIndex = 0;
	p.FSignalIndex = 0;
	p.FIsTx = 1;
	if (0 == tsdb_get_lin_db_signal_properties_by_index(p)){
	  // properties retrieved into p
	}

211. tsfifo_add_can_canfd_pass_filter

函数名称	int tsfifo_add_can_canfd_pass_filter(const s32 AIdxDB, const s32 AId, const bool AIsStd)
功能介绍	添加 can 和 canfd 报文通过过滤器
调用位置	
输入参数	AIdxDB:数据库索引 AId:报文标识符 AIsStd:是否标准帧
返回值	==0：执行成功 其他值：查询错误码
示例	tsfifo_add_can_canfd_pass_filter(APP_CHANNEL.CHN1, 0x3D, true);

212. tsfifo_delete_can_canfd_pass_filter

函数名称	int tsfifo_delete_can_canfd_pass_filter(const s32 AIdxDB, const s32 AId)
功能介绍	卸载 can 和 canfd 报文通过过滤器
调用位置	
输入参数	AIdxDB:数据库索引 AId:报文 ID

返回值	==0：执行成功 其他值：查询错误码
示例	tsfifo_delete_can_canfd_pass_filter(APP_CHANNEL.CHN1,0x3D);

213. tsfifo_add_lin_pass_filter

函数名称	int tsfifo_add_lin_pass_filter(const s32 AIdxDB, const s32 AId)
功能介绍	添加 lin 报文通过过滤器
调用位置	
输入参数	AIdxDB:数据库索引 AId:报文 ID
返回值	==0：执行成功 其他值：查询错误码
示例	tsfifo_add_lin_pass_filter(APP_CHANNEL.CHN1, 0x3B);

214. tsfifo_delete_lin_pass_filter

函数名称	int tsfifo_delete_lin_pass_filter(const s32 AIdxDB, const s32 AId)
功能介绍	卸载 lin 报文通过过滤器
调用位置	
输入参数	AIdxDB:数据库索引 AId:报文 ID
返回值	==0：执行成功 其他值：查询错误码
示例	tsfifo_delete_lin_pass_filter(APP_CHANNEL.CHN1,0x3D);

215. tsfifo_read_can_buffer_frame_count

函数名称	int tsfifo_read_can_buffer_frame_count(const s32 AIdxChn, ps32 ACount);
功能介绍	读取 CAN 缓冲区报文数量
调用位置	
输入参数	AIdxChn:通道索引 ACount:返回的计数
返回值	==0：执行成功 其他值：查询错误码
示例	S32 n = 0; tsfifo_read_can_buffer_frame_count(APP_CHANNEL.CHN1, &n);

216. tsfifo_read_fastlin_buffer_frame_count

函数名称	int tsfifo_read_fastlin_buffer_frame_count(const s32 AIdxChn, ps32 ACount)
功能介绍	读取快速 lin 缓冲区报文数量
调用位置	
输入参数	AIdxChn:通道索引 ACount:返回的计数
返回值	==0：执行成功 其他值：查询错误码
示例	S32 n = 0; tsfifo_read_can_fastlin_frame_count(APP_CHANNEL.CHN1, &n);

217. tsfifo_read_fastlin_rx_buffer_frame_count

函数名称	int tsfifo_read_fastlin_rx_buffer_frame_count(const s32 AIdxChn, ps32 ACount);
功能介绍	读取快速 lin 接收缓冲区报文数量
调用位置	
输入参数	AIdxChn:通道索引 ACount:返回的计数
返回值	==0：执行成功 其他值：查询错误码
示例	S32 n = 0; tsfifo_read_can_fastlin_rx_frame_count(APP_CHANNEL.CHN1, &n);

218. tsfifo_read_fastlin_tx_buffer_frame_count

函数名称	int tsfifo_read_fastlin_tx_buffer_frame_count(const s32 AIdxChn, ps32 ACount);
功能介绍	读取快速 lin 发送缓冲区报文数量
调用位置	
输入参数	AIdxChn:通道索引 ACount:返回的计数
返回值	==0：执行成功 其他值：查询错误码
示例	S32 n = 0; tsfifo_read_can_fastlin_tx_frame_count(APP_CHANNEL.CHN1, &n);

219. tsdb_get_flexray_db_signal_properties_by_address

函数名称	int tsdb_get_flexray_db_signal_properties_by_address(const char* AAddr, PDBSignalProperties Avalue);

功能介绍	通过地址获取 flexray 数据库信号属性
调用位置	
输入参数	AAddr:数据库地址 Avalue:数据库信号属性结构体
返回值	==0：执行成功 其他值：查询错误码
示例	tsdb_get_flexray_db_signal_properties_by_address("0/network1/ecu1/frame1/Signal1",SignalProperties);

220. tsdb_get_flexray_db_signal_properties_by_index

函数名称	int tsdb_get_flexray_db_signal_properties_by_index(PDBSignalProperties Avalue);
功能介绍	通过索引获取 flexray 数据库信号属性
调用位置	
输入参数	Avalue:数据库信号属性结构体
返回值	==0：执行成功 其他值：查询错误码
示例	tsdb_get_flexray_db_signal_properties_by_index(SignalProperties);

221. tsapp_register_pretx_event_lin

函数名称	s32 tsapp_register_pretx_event_lin(const ps32 AObj, const TLINEvent AEvent);
功能介绍	注册 lin 预发送事件
调用位置	
输入参数	AObj:句柄，唯一标识符 AEvent：处理 LIN 数据的委托函数
返回值	==0：执行成功 其他值：查询错误码

示例	s32 AObj = 0;
	void OnLINEvent(const ps32 AObj, const PLIN ALIN)
	{
	//处理收到的报文数据 AData
	}
	tsapp_register_pretx_event_lin(&AObj,OnLINEvent);

222. tsapp_set_logger

函数名称	s32 tsapp_set_logger(const TLogger ALogger);

功能介绍	设置记录事件
调用位置	
输入参数	ALogger:记录委托函数
返回值	==0：执行成功 其他值：查询错误码

示例	void TSMasterLogger(const char* AStr, const s32 ALevel)
	{
	//
	}
	tsapp_set_logger(TSMasterLogger);

223. tsapp_unregister_events_can

函数名称	s32 tsapp_unregister_events_can(const ps32 AObj);
功能介绍	注销 can 委托事件
调用位置	
输入参数	AObj:句柄，唯一标识符
返回值	==0：执行成功 其他值：查询错误码
示例	s32 AObj = 0; tsapp_unregister_events_can(&AObj);

224. tsapp_unregister_events_lin

函数名称	s32 tsapp_unregister_events_canfd(const ps32 AObj);
功能介绍	注销 lin 委托事件
调用位置	
输入参数	AObj:句柄，唯一标识符
返回值	==0：执行成功 其他值：查询错误码
示例	s32 AObj = 0; tsapp_unregister_events_lin(&AObj);

225. tsapp_unregister_events_canfd

函数名称	s32 tsapp_unregister_events_canfd(const ps32 AObj);

功能介绍	注销 canfd 委托事件
调用位置	
输入参数	AObj:句柄，唯一标识符
返回值	==0：执行成功 其他值：查询错误码
示例	s32 AObj = 0; tsapp_unregister_events_canfd(&AObj);

226. tsapp_unregister_pretx_events_can

函数名称	s32 tsapp_unregister_pretx_events_can(const ps32 AObj);
功能介绍	注销 can 预发送委托事件
调用位置	
输入参数	AObj:句柄，唯一标识符 AEvent:can 数据处理委托函数
返回值	==0：执行成功 其他值：查询错误码
示例	s32 AObj = 0; tsapp_unregister_pretx_events_can(&AObj);

227. tsapp_unregister_pretx_events_lin

函数名称	s32 tsapp_unregister_pretx_events_lin(const ps32 AObj);
功能介绍	注销 lin 预发送委托事件
调用位置	
输入参数	AObj:句柄，唯一标识符
返回值	==0：执行成功 其他值：查询错误码
示例	s32 AObj = 0; tsapp_unregister_pretx_events_lin(&AObj);

228. tsapp_unregister_pretx_events_canfd

函数名称	s32 tsapp_unregister_pretx_events_canfd(const ps32 AObj);
功能介绍	注销 canfd 预发送委托事件
调用位置	

输入参数	AObj:句柄，唯一标识符
返回值	==0：执行成功 其他值：查询错误码
示例	s32 AObj = 0; tsapp_unregister_pretx_events_canfd(&AObj);

229. tscom_flexray_get_signal_value_in_raw_frame

函数名称	double tscom_flexray_get_signal_value_in_raw_frame(const PFlexRaySignal ASignal, pu8 data);
功能介绍	获取 flexray 信号值通过原始帧
调用位置	
输入参数	ASignal:flexray 信号结构体指针 data:数据
返回值	==0：执行成功 其他值：查询错误码
示例	FlexRaySignal flexraySignal; u8 data; tscom_flexray_get_signal_value_in_raw_frame(&flexraySignal,&data);

230. tsdb_load_flexray_db

函数名称	s32 tsdb_load_flexray_db(const char* AFRFile, const char* ASupportedChannels, pu32 AId);
功能介绍	加载 flexray 数据库
调用位置	
输入参数	AFRFile:flexray 数据库文件路径 ASupportedChannels:绑定的通道数组 AId：数据库返回 ID
返回值	==0：执行成功 其他值：查询错误码

示例	Const char* AFRFile = “”;
	Const char* ASupportedChannels = “”;
	u32 AId = 0;
	tsdb_load_flexray_db(AFRFile, ASupportedChannels, &AId);

231. tsdb_get_flexray_db_properties_by_address_verbose

函数名称	s32 tsdb_get_flexray_db_properties_by_address_verbose(const char* AAddr, ps32 ADBIndex, ps32 ASignalCount, ps32 AFrameCount, ps32 AECUCount, ps64 ASupportedChannelMask, char** AName, char** AComment);
功能介绍	通过地址获取数据库属性信息

调用位置	
输入参数	AAddr：数据库地址 ADBIndex：返回的数据库索引 ASignalCount：返回的信号数量 AFrameCount：返回的报文数量 AECUCount：返回的 ecu 数量 ASupportedChannelMask：返回的支持通道掩码 AName：数据库名称 AComment：注释
返回值	0：执行成功 其他值：失败
示例	const char* AAddr = “D:\TestDBC\...”; tsdb_get_flexray_db_properties_by_address_verbose(AAddr, &ADBIndex,&ASignalCount,&AFrameCount,&AECUCount,&ASupportedChannelMask, &AName,&AComment)

232. tsdb_get_flexray_db_properties_by_index_verbose

函数名称	s32 tsdb_get_flexray_db_properties_by_index_verbose(s32 ADBIndex, ps32 ASignalCount, ps32 AFrameCount, ps32 AECUCount, ps64 ASupportedChannelMask, char** AName, char** AComment);
功能介绍	通过索引获取数据库属性信息
调用位置	
输入参数	ADBIndex：数据库索引 ASignalCount：返回的信号数量 AFrameCount：返回的报文数量 AECUCount：返回的 ecu 数量 ASupportedChannelMask：返回的支持通道掩码 AName：数据库名称 AComment：注释
返回值	0：执行成功 其他值：失败
示例	tsdb_get_flexray_db_properties_by_index_verbose(0,& ASignalCount, &AFrameCount,&AECUCount,&ASupportedChannelMask,&AName,&AComment);

233. tsdb_get_flexray_ecu_properties_by_address_verbose

函数名称	s32 tsdb_get_flexray_ecu_properties_by_address_verbose(const char* AAddr, ps32 ADBIndex, ps32 AECUIndex, ps32 ATxFrameCount, ps32 ARxFrameCount, char** AName, char** AComment);
功能介绍	通过地址获取 flexray ecu 属性信息
调用位置	

输入参数	Adder：地址 ADBIndex：数据库索引 AECUIndex：返回的信号数量 ATxFrameCount：返回的 TX 帧数 ARxFrameCount：返回的 Rx 帧数 AName：数据库名称 AComment：注释
返回值	0：执行成功 其他值：失败
示例	const char* adder = “D:\TestDB\”； tsdb_get_flexray_ecu_properties_by_address_verbose(adder, &DBIndex,&AECUIndex,&ATxFrameCount,&ARxFrameCount,&AName, &AComment)；

234. tsdb_get_flexray_ecu_properties_by_index_verbose

函数名称	s32 tsdb_get_flexray_ecu_properties_by_index_verbose(s32 ADBIndex, s32 AECUIndex, ps32 ATxFrameCount, ps32 ARxFrameCount, char** AName, char** AComment)；
功能介绍	通过索引获取 flexray ecu 属性信息
调用位置	
输入参数	ADBIndex：数据库索引 AECUIndex：返回的信号数量 ATxFrameCount：返回的 TX 帧数 ARxFrameCount：返回的 Rx 帧数 AName：数据库名称 AComment：注释
返回值	0：执行成功 其他值：失败
示例	tsdb_get_flexray_ecu_properties_by_index_verbose(0, &AECUIndex, &ATxFrameCount, &ARxFrameCount, &AName,&AComment)

235. tsdb_get_flexray_frame_properties_by_address_verbose

函数名称	s32 tsdb_get_flexray_frame_properties_by_address_verbose(const char* AAddr, ps32 ADBIndex, ps32 AECUIndex, ps32 AFrameIndex, bool* AIsTx, ps32 AFRChannelMask, ps32 AFRBaseCycle, ps32 AFRCycleRepetition, bool* AFRIsStartupFrame, ps32 AFRSlotId, ps64 AFRCycleMask, ps32 ASignalCount, char** AName, char** AComment)；
功能介绍	通过地址获取 flexray 帧属性信息
调用位置	

输入参数	adder：地址 ADBIndex：数据库索引 AECUIndex：ecu 索引 AFrameIndex：帧索引 AIsTx：是否 TX AFRChannelMask：通道掩码 AFRBaseCycle：基本周期 AFRCycleRepetition：重复周期 AFRIsStartupFrame：是否起始帧 AFRSlotId：slotId AFRCycleMask：周期掩码 ASignalCount：信号数量 AName：数据库名称 AComment：注释
返回值	0：执行成功 其他值：失败
示例	const char* adder = “D:\TestDB\”； tsdb_get_flexray_frame_properties_by_address_verbose(adder,& ADBIndex,&AECUIndex,&AFrameIndex,&AIsTx,&AFRChannelMask,& AFRBaseCycle,&AFRCycleRepetition,&AFRIsStartupFrame,&AFRSlotId, &AFRCycleMask, &ASignalCount, &AName,&AComment);

236. tsdb_get_flexray_frame_properties_by_index_verbose

函数名称	s32 tsdb_get_flexray_frame_properties_by_index_verbose(s32 ADBIndex, s32 AECUIndex, s32 AFrameIndex, bool AIsTx, ps32 AFRChannelMask, ps32 AFRBaseCycle, ps32 AFRCycleRepetition, bool* AFRIsStartupFrame, ps32 AFRSlotId, ps64 AFRCycleMask, ps32 ASignalCount, char** AName, char** AComment);
功能介绍	通过索引获取 flexray 帧属性信息
调用位置	
输入参数	ADBIndex：数据库索引 AECUIndex：ecu 索引 AFrameIndex：帧索引 AIsTx：是否 TX AFRChannelMask：通道掩码 AFRBaseCycle：基本周期 AFRCycleRepetition：重复周期 AFRIsStartupFrame：是否起始帧 AFRSlotId：slotId AFRCycleMask：周期掩码 ASignalCount：信号数量 AFRDLC：DLC 长度 AName：数据库名称

	AComment：注释
返回值	0：执行成功 其他值：失败
示例	tsdb_get_flexray_frame_properties_by_index_verbose(0,& AECUIndex, &AFrameIndex,&AIsTx,&AFRChannelMask,&AFRBaseCycle, &AFRCycleRepetition,&AFRIsStartupFrame,&AFRSlotId,&AFRCycleMask, &ASignalCount,&AFRDLC,&AName,&AComment);

237. tsdb_get_flexray_signal_properties_by_address_verbose

函数名称	s32 tsdb_get_flexray_signal_properties_by_address_verbose(const char* AAddr, ps32 ADBIndex, ps32 AECUIndex, ps32 AFrameIndex, ps32 ASignalIndex, bool* AIsTx, PSignalType ASignalType, PFlexRayCompuMethod ACompuMethod, bool* AIsIntel, ps32 AStartBit, ps32 AUpdateBit, ps32 ALength, pdouble AFactor, pdouble AOffset, pdouble AInitValue, char** AName, char** AComment);
功能介绍	通过地址获取 flexray 信号属性信息
调用位置	
输入参数	adder：地址 ADBIndex：数据库索引 AECUIndex：ecu 索引 AFrameIndex：帧索引 ASignalIndex：信号索引 AIsTx：是否 TX ASignalType：信号类型 ACompuMethod：计算方法 AIsIntel：是否 Intel AStartBit：起始位 AUpdateBit：更新位 ALength：长度 AFactor：放大因子 AOffset：偏移量 AInitValue：初始值 AName：信号名称 AComment:注释
返回值	0：执行成功 其他值：失败
示例	const char* adder = "D:\TestDB\..."; tsdb_get_flexray_signal_properties_by_address_verbose(adder, & ADBIndex,&AECUIndex,&AFrameIndex,&ASignalIndex,&AIsTx,& ASignalType,&ACompuMethod,&AIsIntel,&AStartBit,&AUpdateBit,& ALength, &AFactor, &AOffset,&AInitValue, &AName,& AComment);

238. tsdb_get_flexray_signal_properties_by_index_verbose

函数名称	s32 tsdb_get_flexray_signal_properties_by_index_verbose(s32 ADBIndex, s32 AECUIndex, s32 AFrameIndex, s32 ASignalIndex, bool AIsTx, PSignalType ASignalType, PFlexRayCompuMethod ACompuMethod, bool* AIsIntel, ps32 AStartBit, ps32 AUpdateBit, ps32 ALength, pdouble AFactor, pdouble AOffset, pdouble AInitValue, char** AName, char** AComment);
功能介绍	通过索引获取 flexray 信号属性信息
调用位置	
输入参数	ADBIndex：数据库索引 AECUIndex：ecu 索引 AFrameIndex：帧索引 ASignalIndex：信号索引 AIsTx：是否 TX ASignalType：信号类型 ACompuMethod：计算方法 AIsIntel：是否 Intel AStartBit：起始位 AUpdateBit：更新位 ALength：长度 AFactor：放大因子 AOffset：偏移量 AInitValue：初始值 AName：信号名称 AComment:注释
返回值	0：执行成功 其他值：失败
示例	tsdb_get_flexray_signal_properties_by_index_verbose(0,&AECUIndex, &AFrameIndex, &ASignalIndex,&AIsTx, &ASignalType, &ACompuMethod, &AIsIntel, &AStartBit,&AUpdateBit, &ALength, &AFactor, &AOffset, &AInitValue, &AName,&AComment);

239. tsLog_blf_read_start_verbose

函数名称	s32 tsLog_blf_read_start_verbose(char* AFileName, ps32 AHandle, ps32 AObjCount,u16* AYear, u16* AMonth, u16* ADayOfWeek,u16* ADay, u16* AHour, u16* AMinute,u16* ASecond, u16* AMilliseconds);
功能介绍	读取已打开 BLF 文件的状态信息
调用位置	
输入参数	AFileName：文件名称 AHandle：文件句柄 AObjCount：blf 文件包含的数据个数，传入指针，在函数内部赋值

	AYear：年 AMonth：月 ADayOfWeek：周 ADay：日 AHour：小时 AMinute：分钟 ASecond：秒 AMilliseconds：微妙
返回值	0：执行成功 其他值：失败
示例	tsLog_blf_read_start_verbose(&AFileName,&AHandle,&AObjCount,&AYear, &AMonth,&ADayOfWeek,&ADay,&AHour,&AMinute,&ASecond,&AMilliseconds);

240. tsflexray_set_controller_frametrigger

函数名称	s32 tsflexray_set_controller_frametrigger(const size_t ADeviceHandle, const int ANodeIndex, const PLibFlexray_controller_config AControllerConfig, const int* AFrameLengthArray, const int AFrameNum, const PLibTrigger_def AFrameTrigger, const int AFrameTriggerNum, const int ATimeoutMs);
功能介绍	配置 flexray 报文的控制器帧触发
调用位置	
输入参数	ADeviceHandle：设备句柄 ANodeIndex：节点索引 AControllerConfig：TLIBFlexray_controller_config 结构体指针 AFrameLengthArray：帧长度数组 AFrameNum：帧数 AFrameTrigger：TLibTrigger_def 结构体指针 AFrameTriggerNum：帧触发器数 ATimeoutMs：超时时间
返回值	0：执行成功 其他值：失败

示例	size_t ADeviceHandle = 0;
	int appChannel = 0;
	TLibFlexray_controller_config controllerConfig;
	int AFrameLengthArray[32];
	TLibTrigger AFrameTrigger ;
	tsflexray_set_controller_frametrigger(ADeviceHandle, appChannel ,
	&AControllerConfig,
	AFrameLengthArray, 1,
	&AFrameTrigger, 1, 100);

10. 附件

1. 示例程序

参考章节 2

2. 错误码编码信息：

```
ERR_CODE_LIST: array [0..ERR_CODE_COUNT-1] of string = (
    'OK',                                   // IDX_ERR_OK                          0
    'Index out of range',                   // IDX_ERR_IDX_OUT_OF_RANGE            1
    'Connect failed',                       // IDX_ERR_CONNECT_FAILED              2
    'Device not found',                     // IDX_ERR_DEV_NOT_FOUND               3
    'Read status timed out',                // IDX_ERR_READ_STATUS_TIMEDOUT        44
    'Callback already exists',              // IDX_ERR_CALLBACK_ALREADY_EXISTS     45
    'Callback not exists',                  // IDX_ERR_CALLBACK_NOT_EXISTS         46
    'File corrupted or not recognized',      // IDX_ERR_FILE_INVALID                = 47; // database file
corrupted or not recognized
    'Database unique id not found',         // IDX_ERR_DB_ID_NOT_FOUND             = 48; // database unique id
not found
    'Software API parameter invalid',          // IDX_ERR_SW_API_PAEAMETER_INVALID    = 49; // software api
parameter invalid
    'Software API generic timed out',       // IDX_ERR_SW_API_GENERIC_TIMEOUT      = 50; // software api generic
timed out
    'Software API set hw config. failed',   // IDX_ERR_SW_API_SET_CONF_FAILED      = 51; // software api set hw
conf failed
    'Index out of bounds',                  // IDX_ERR_SW_API_INDEX_OUT_OF_BOUNDS = 52; // index out of bounds
    'RX wait timed out',                    // IDX_ERR_SW_API_WAIT_TIMEOUT         = 53; // rx wait timed out
    'Get I/O failed',                       // IDX_ERR_SW_API_GET_IO_FAILED        = 54; // get io failed
    'Set I/O failed',                       // IDX_ERR_SW_API_SET_IO_FAILED        = 55; // set io failed
    'An active replay is already running',  // IDX_ERR_SW_API_REPLAY_ON_GOING      = 56; // a replay is already
on goning
    'Instance not exists',                  // IDX_ERR_SW_API_INSTANCE_NOT_EXISTS = 57; // instance not exists
    'CAN message transmit failed',          // IDX_ERR_HW_CAN_TRANSMIT_FAILED      = 58; // can transmit frame
failed
    'No response from hardware',            // IDX_ERR_HW_NO_RESPONSE              = 59; // hardware no response
to pc
    'CAN message not found',                // IDX_ERR_SW_CAN_MSG_NOT_FOUND        = 60; // can cyclic message
id not found
    'User CAN receive buffer empty',        // IDX_ERR_SW_CAN_RECV_BUFFER_EMPTY    = 61; // user can recv message
buffer empty
    'CAN total receive count <> desired count',// IDX_ERR_SW_CAN_RECV_PARTIAL_READ   = 62; // total read count <>
desired read count
    'LIN config failed',                    // IDX_ERR_SW_API_LINCONFIG_FAILED     = 63;
    'LIN frame number out of range',        // IDX_ERR_SW_API_FRAMENUM_OUTOFRANGE = 64;
```

```
    'LDF config failed',                    // IDX_ERR_SW_API_LDFCONFIG_FAILED    = 65;
    'LDF config cmd error',                 // IDX_ERR_SW_API_LDFCONFIG_CMDERR    = 66;
    'TSMaster envrionment not ready',       // IDX_ERR_SW_ENV_NOT_READY           = 67; // encryption random
number not retrieved
    'reserved failed',                      // IDX_ERR_RESERVED_FAILED            = 68;
    'XL driver error',                      // IDX_ERR_XL_ERROR                   = 69;
    'index out of range',                   //  IDX_ERR_SEC_INDEX_OUTOFRANGE           = 70;
    'string length out of range',           //  IDX_SEC_ERR_STRINGLENGTH_OUTFOF_RANGE  = 71;
    'key is not initialized',               //  IDX_SEC_ERR_KEY_IS_NOT_INITIALIZATION  = 72;
    'key is wrong',                         //  IDX_SEC_ERR_KEY_IS_WRONG               = 73;
    'write not permitted',                  //  IDX_SEC_ERR_NOT_PERMIT_WRITE           = 74;
    '16 bytes multiple',                    //  IDX_SEC_ERR_16BYTES_MULTIPLE           = 75;
    'LIN channel out of range',             // IDX_ERR_LIN_CHN_OUTOF_RANGE            = 76;
    'DLL not ready',                        // IDX_ERR_DLL_NOT_READY                  = 77;
    'Feature not supported',                //  IDX_ERR_FATURE_NOT_SUPPORTED           = 78;
    'common service error',                 // IDX_ERR_COMMON_SERV_ERROR              = 79;
    'read parameter overflow',              // IDX_ERR_READ_PARA_OVERFLOW             = 80;
    'Invalid application channel mapping',  // IDX_ERR_INVALID_CHANNEL_MAPPING
    'libTSMaster generic operation failed', // IDX_ERR_TSLIB_GENERIC_OPERATION_FAILED = 82;
    'item already exists',                  // IDX_ERR_TSLIB_ITEM_ALREADY_EXISTS      = 83;
    'item not found',                       // IDX_ERR_TSLIB_ITEM_NOT_FOUND           = 84;
    'logical channel invalid',              // IDX_ERR_TSLIB_LOGICAL_CHANNEL_INVALID  = 85;
    'file not exists',                      // IDX_ERR_FILE_NOT_EXISTS                = 86;
    'no init access, cannot set baudrate',  // IDX_ERR_NO_INIT_ACCESS                 = 87;
    'the channel is inactive',              // IDX_ERR_CHN_NOT_ACTIVE                 = 88;
    'the channel is not created',           // IDX_ERR_CHN_Not_Created                = 89;
    'length of the appname is out of range', // IDX_ERR_APPNAME_LENGTH_OUT_OF_RANGE    = 90;
    'project is modified',                  // IDX_ERR_PROJECT_IS_MODIFIED            = 91;
    'signal not found in database',         // IDX_ERR_SIGNAL_NOT_FOUND_IN_DB         = 92;
    'message not found in database',        // IDX_ERR_MESSAGE_NOT_FOUND_IN_DB        = 93;
    'TSMaster is not installed',            // IDX_ERR_TSMASTER_IS_NOT_INSTALLED      = 94;
    'Library load failed',                  // IDX_ERR_LIB_LOAD_FAILED                = 95;
    'Library function not found',           // IDX_ERR_LIB_FUNCTION_NOT_FOUND         = 96;
    'Library not initialized',              // IDX_ERR_LIB_NOT_INITIALIZED            = 97;
    'PCAN generic operation error',         // IDX_ERR_PCAN_GENRIC_ERROR              = 98;
    'Kvaser generic operation error',       // IDX_ERR_KVASER_GENERIC_ERROR           = 99;
    'ZLG generic operation error',          // IDX_ERR_ZLG_GENERIC_ERROR              = 100;
    'ICS generic operation error',          // IDX_ERR_ICS_GENERIC_ERROR              = 101;
    'TC1005 generic operation error',       // IDX_ERR_TC1005_GENERIC_ERROR           = 102;
    'System variable not found',            // IDX_ERR_SYSTEM_VAR_NOT_FOUND           = 103;
    'Incorrect system variable type',       // IDX_ERR_INCORRECT_SYSTEM_VAR_TYPE      = 104;
    'Message not existing, update failed',  // IDX_ERR_CYCLIC_MSG_NOT_EXIST           = 105;
    'Specified baudrate not available'      // IDX_ERR_BAUD_NOT_AVAIL                 = 106;
      'Device does not support sync. transmit',  // IDX_ERR_DEV_NOT_SUPPORT_SYNC_SEND      = 107;
```

```
    'Wait time not satisfied',                 // IDX_ERR_MP_WAIT_TIME_NOT_SATISFIED     = 108;
    'Cannot operate while app is connected',   // IDX_ERR_CANNOT_OPERATE_WHILE_CONNECTED = 109;
    'Create file failed',                      // IDX_ERR_CREATE_FILE_FAILED             = 110;
    'Execute python failed',                   // IDX_ERR_PYTHON_EXECUTE_FAILED          = 111;
    'Current multiplexed signal is not active',// IDX_ERR_SIGNAL_MULTIPLEXED_NOT_ACTIVE  = 112;
    'Get handle by logic channel failed',      // IDX_ERR_GET_HANDLE_BY_CHANNEL_FAILED   = 113;
    'Cannot operate while application is ' +
    'connected, please stop application first',// IDX_ERR_CANNOT_OPERATE_WHILE_APP_CONN  = 114;
    'File load failed',                        // IDX_ERR_FILE_LOAD_FAILED               = 115;
    'Read LIN Data Failed',                    // IDX_ERR_READ_LINDATA_FAILED            = 116;
    'FIFO not enabled'                         // IDX_ERR_FIFO_NOT_ENABLED               = 117;
```

11. 常见异常解答

1. 调用 API 开发的程序无法正常打开 CAN 驱动

- **症状描述：**

TSMaster 已经安装（不代表完全正常），可以打开正常使用。但是基于 TSMaster API 开发的程序无法正常工作，总是提示打开 CAN 总线设备失败。如果打印 API 的返回值，发现都是 97，也就是代表驱动没有正常初始化。

- **原因分析：**

TSMaster 在安装的时候需要把一些配置信息和路径信息等写入到注册表中，但是很多电脑里面设置了权限管理，TSMaster 在安装的时候无法正常完成所有配置信息的写入。造成用户程序在调用 API 的时候，查找不到库文件的路径，从而无法完成 API 驱动的初始化。

- **解决办法：**

建议：TSMaster 里面增加一个模块来管理这些注册表信息是否就绪，如果未就绪，看看能否直接补齐，或者至少给与用户提示信息。

2. 调用 API，总是执行不成功，返回错误码 0x114

- **症状描述：**

比如，在程序中调用 API 执行加载数据库，卸载数据库的操作，结果总是返回失败。查看返回的错误码，可以看到为 0x114。查看错误码对照表，0x114 代表的意思是：'connected, please stop application first',// IDX_ERR_CANNOT_OPERATE_WHILE_APP_CONN = 114;

- **原因分析：**

这是因为有的 API 函数必须在应用处于断开状态（Application 处于断开）的时候才能够执行。比如数据库的加载，如果应用程序已经处于连接状态，数据库引擎，包括过滤，信号解析这些模块都运行起来了，此时就不允许用户进行加载卸载这些操作了。

- **解决办法：**

调整调用相应 API 执行的顺序。还是以数据库的加载和卸载为例。可以放在调用 tsapp_connect 函数之前，调用这些 API，就不会出现这个问题。或者等到执行了 tsapp_disconnect 断开应用程序过后，再调用这些 API 函数。