TsCANApi.Interop 编程指导 V0.8

1. 目录

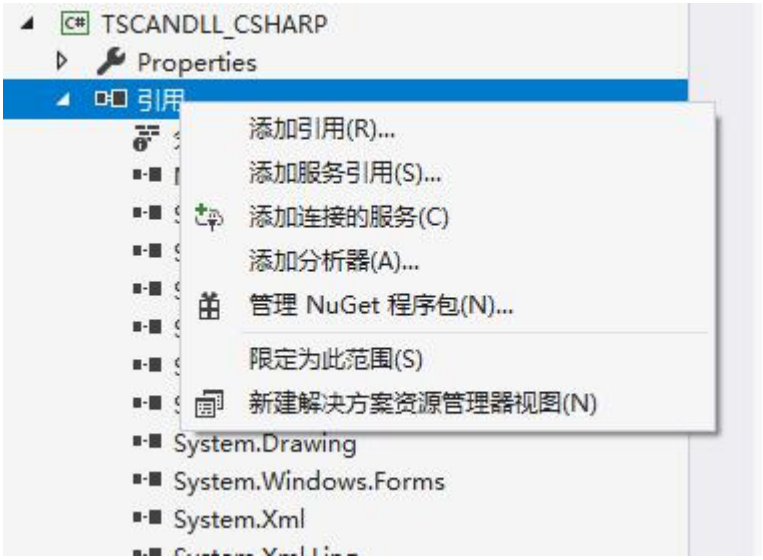
1. 什么情况下需要此文档？

用户基于 DotNet 平台的编程语言，对上海同星智能科技有限公司的 TSCAN 系列工具（TSCANMini，TSLiteMini，TSCANFDMini，TSCANLINLite，TSCANLINPro）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。

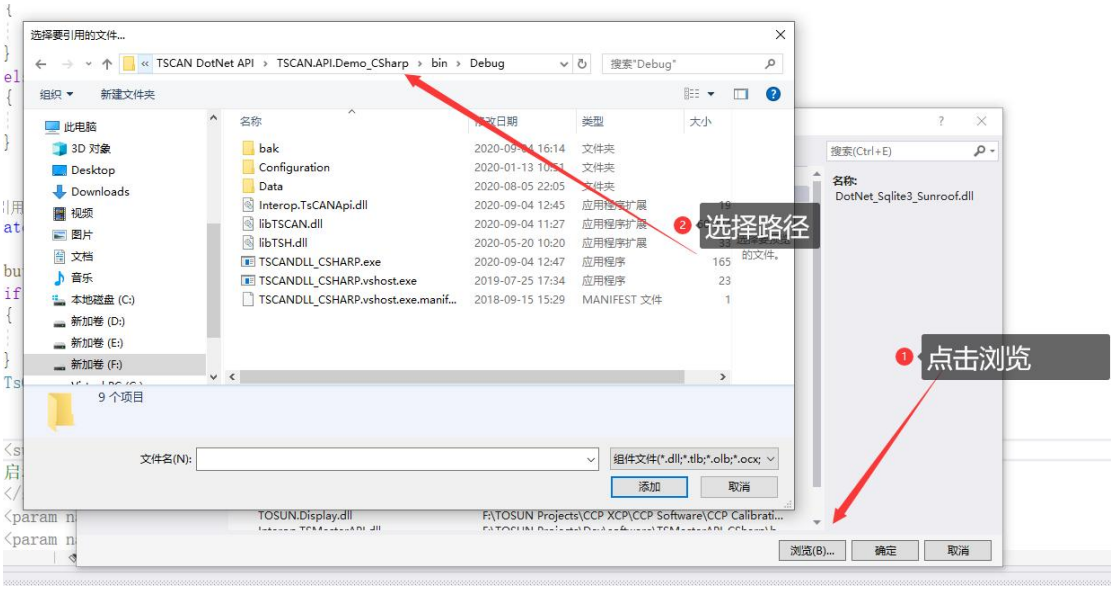
2. 添加库文件

对于 DotNet 工程，本库文件可以直接添加到工程的引用中，如下图所示：

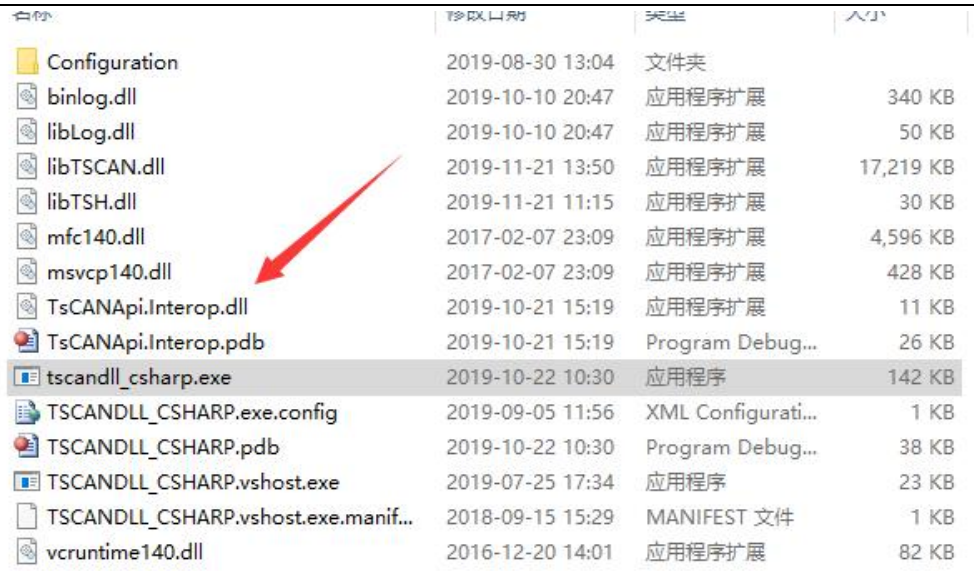
1. 第一步：选中工程引用，右键，点击添加引用：



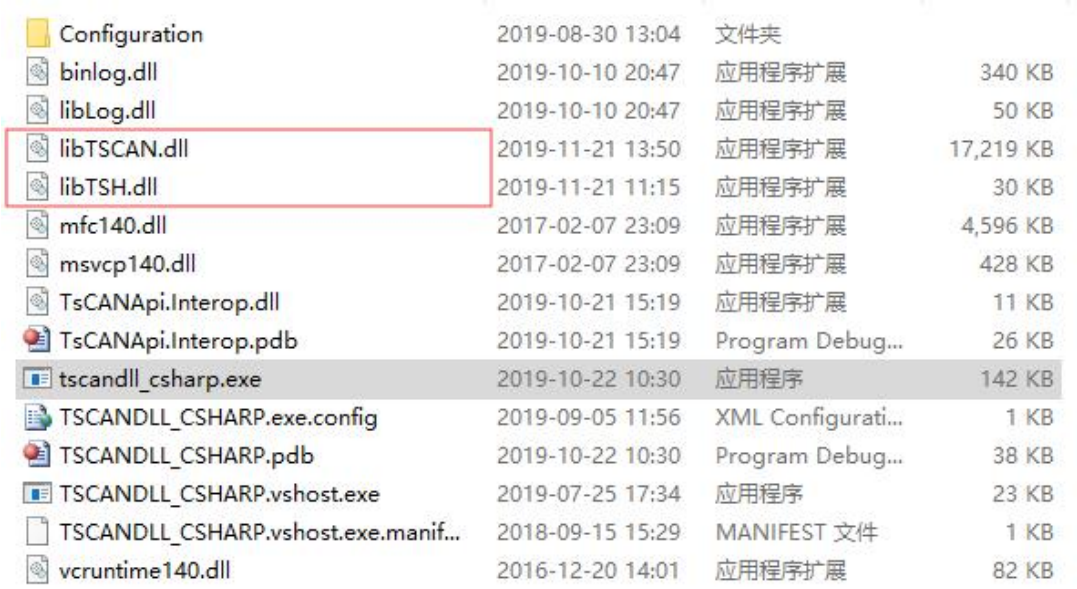
2. 第二步：在引用管理器中点击浏览，弹出文件选择框：



3. 文件目录中查找到：TsCANApi.Interop.dll，点击添加。



4. 最后在引用管理器中，点击确认，即把库文件加入到了工程中。
5. 添加底层驱动库函数: libTSCAN.dll 和 libTSH.dll,这两个文件直接放入到可执行程序目录即可，不需要添加到工程文件中，如下所示：



3. 总线数据类型定义

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

调用示例：

```
TLibCAN CANMsg = new TLibCAN(0, 0x100, true, false, false, 8,
                                    new byte[] { 0, 1, 2, 3, 4, 5, 6, 7 });
```

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;
```

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

调用示例：

```
TLibCANFD CANFDMsg = new TLibCANFD(0, 0x100, true, false, false, 8,
                                   new byte[] { 0, 1, 2, 3, 4, 5, 6, 7 });
```

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

调用示例：

```
TLIN l;
l.init_w_id(0x15, 8);
if (0 != tscan_connect()) return;
if (0 == tslin_transmit_lin_async(ADeviceHandle，&l)){
 }//ADeviceHandle：设备句柄
```

4. TLibFlexray：Flexray 总线数据类型

```
public struct TLIBFlexray
    {
        public byte FIdxChn;  // channel index starting from 0
        public byte FChannelMask; // 0: reserved, 1: A, 2: B, 3: AB
        public byte FDir;     // 0: Rx, 1: Tx, 2: Tx Request
        public byte FPayloadLength; // payload length in bytes
        public byte FActualPayloadLength;  // actual data bytes
        public byte FCycleNumber;  // cycle number: 0~63
        public byte 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)
        public byte FReserved0;
        public ushort FHeaderCRCA; // header crc A
        public ushort FHeaderCRCB; // header crc B
        public ushort FFrameStateInfo; // bit 0~15, error flags
        public ushort FSlotId; // static seg: 0~1023
        public uint FFrameFlags; // bit 0~22
        public uint FFrameCRC; // frame crc
        public ulong FReserved1; // 8 reserved bytes
        public ulong FReserved2; // 8 reserved bytes
        public ulong FTimeUs; // timestamp in us
        public byte[] FData; // 254 data bytes
    }
```

成员：

FIdxChn:通道索引
FChannelMask：通道掩码
FPayloadLength：有效负载长度(字节)
FActualPayloadLength：实际数据字节
FCycleNumber：周期数
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

调用示例：

```
TLIBFlexray tLIBFlexray = new TLIBFlexray(0, 0, 254, 1, 5, new byte[] { 1 });
for(int i = 0; i < 254; i++)
{
    tLIBFLexray.FData[i] = i;
}
tscan_transmit_flexray_async(ADeviceHandle,ref tLIBFlexray);//ADeviceHandle:设备句柄
```

4. 报文发送

1. CAN 报文发送

```
TLIBCAN can_frame = new TLIBCAN(0, 0x100, true, false, false, 8, new byte[] { 0, 1, 2,
                                3,4, 5, 6, 7 });
```

单帧异步发送：

```
tscan_transmit_can_async(ADeviceHandle,ref can_frame);//ADeviceHandle:设备句柄
```

单帧同步发送：

```
# 100 表示超时参数
tscan_transmit_can_sync(ADeviceHandle,ref can_frame,100);//ADeviceHandle:设备句柄
```

周期发送：

```
# 100ms 周期发送 ACAN 报文
tscan_add_cyclic_msg_can(ADeviceHandle,ref can_frame,100);//ADeviceHandle:设备句柄
```

删除周期发送：

```
# 删除周期发送 ACAN 报文
tscan_delete_cyclic_msg_can(ADeviceHandle,ref can_frame);//ADeviceHandle:设备句柄
```

2. CANFD 报文发送

```
TLIBCANFD can_frame = new TLIBCANFD(0, 0x100, true, false, false, 8, new byte[] { 0,
```

```
1,2,3,4,5, 6, 7 });
```

单帧异步发送：

tscan_transmit_canfd_async(ADeviceHandle,ref can_frame); //ADeviceHandle:设备句柄

单帧同步发送：

# 100 表示超时参数
tscan_transmit_canfd_sync(ADeviceHandle,ref can_frame,100);//ADeviceHandle:设备句柄

周期发送：

# 100ms 周期发送 ACAN 报文
tscan_add_cyclie_msg_canfd(ADeviceHandle,ref can_frame,100); //ADeviceHandle:设备句柄

删除周期发送：

# 删除周期发送 ACAN 报文
tscan_delete_cyclie_msg_canfd(ADeviceHandle,ref can_frame); //ADeviceHandle:设备句柄

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：

tslin_transmit_lin_async(ADeviceHandle,ref lin_frame);//ADeviceHandle:设备句柄

单帧同步发送：

# 100 表示超时参数
tslin_transmit_lin_sync(ADeviceHandle,ref lin_frame,100); //ADeviceHandle:设备句柄

4. Flexray 报文发送

```
TLIBFlexray tLIBFlexray = new TLIBFlexray(0, 1, 254, 1, 5, new byte[] { 1 });
for (u32 i = 0; i < 254; i++) {
    flexRay_frame.FData[i] = 0;
}
```

单帧异步发送：

tsflexray_transmit_async(ADeviceHandle, ref flexRay_frame);//ADeviceHandle:设备句柄

单帧同步发送：

100 表示超时参数

tsflexray_transmit_sync(ADeviceHandle,ref flexRay_frame, 100);//ADeviceHandle:设备句柄

5. 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。libTSCAN 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：

采用代理机制（C#里面类 C 函数指针），注册回调函数。需要注意的是，一定要先申请一个代理对象，然后把用户回调函数注册到该代理对象上，如下所示：

```
//
receiveCANCallBack += new TCANQueueEvent_Win32(ReceivedCANMsgCallBack);
receiveCANFDCallBack += new TCANFDQueueEvent_Win32(ReceivedCANFDMsgCallBack);
receiveLINCallBack += new TLINQueueEvent_Win32(ReceivedLINMsgCallBack);
connecttedCallBack += new TTSCANConnectedCallback_Win32(TSCANConnectedCallBack);
disconnectedCallBack += new TTSCANConnectedCallback_Win32(TSCANDisConnectedCallBack);
```

然后把代理对象调用回调函数注册到驱动中，如下所示：

```
uint ret = TsCANApi.RegisterCANRecvCallback_Win32(deviceHandle, receiveCANCallBack);
```

以上代码，有两点原因：

- 不采用代理对象，直接把函数比如 ReceivedCANMsgCallBack 注册到驱动中，程序也能够执行。但是因为.Net 是动态内存管理机制，运行一会儿过后，在 C#端的函数指针地址已经发生该表，注册在内部的函数指针无法跟着改变，会触发访问异常。
- 采用代理机制，意味着同一个代理代理对象，可以注册多个回调函数。比如 receiveCANCallBack，可以采用 += new 方式注册多个函数。用户可以把不同功能的处理函数注册到指针上。

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：

```
/// <summary>
    /// Classic CAN接收函数回调函数：需要注意的是，这个函数是在多线程中调用的。类似于串口控件的OnDataReceived事件机制，开发人员使用时候注意做好线程保护和同步。
    /// </summary>
    /// <param name="AData"></param>
     void ReceivedCANMsgCallBack(ref TCAN AData)
     {
        ///用户可以在此添加处理CAN报文AData的逻辑
        if (AData.FIsTx)
        {
          //表示这个报文是从本设备发出去的
        }
        if (AData.FIsExt)
        {
          //是否扩展帧
        }
        if (AData.FIsRemote)
        {
          //是否远程帧，否则就是数据帧
        }
        //依次类推
        DispMsg(AData.GetString());
    }
    receiveCANCallBack += new TCANQueueEvent_Win32(ReceivedCANMsgCallBack);
```

CANFD 回调：

```
     void ReceivedCANFDMsgCallBack(ref TLIBCANFD AData)
     {
        ///用户可以在此添加处理CAN报文AData的逻辑
        if (AData.FIsTx)
```

```
        {
          //表示这个报文是从本设备发出去的
        }
        if (AData.FIsExt)
        {
          //是否扩展帧
        }
        if (AData.FIsRemote)
        {
          //是否远程帧，否则就是数据帧
        }
        //依次类推
        DispMsg(AData.GetString());
    }
    receiveCANFDCallBack += new TCANFDQueueEvent_Win32(ReceivedCANFDMsgCallBack);
```

LIN 回调：

```
    void ReceivedLINMsgCallBack(ref TLIBLIN AData)
    {
        ///用户可以在此添加处理CAN报文AData的逻辑
        if (AData.FIsTx)
        {
          //表示这个报文是从本设备发出去的
        }
        if (AData.FIsExt)
        {
          //是否扩展帧
        }
        if (AData.FIsRemote)
        {
          //是否远程帧，否则就是数据帧
        }
        //依次类推
        DispMsg(AData.GetString());
    }
    receiveLINCallBack += new TLINQueueEvent_Win32(ReceivedLINMsgCallBack);
```

Flexray 回调：

```
    void ReceivedFlexrayMsgCallBack(ref TLIBFlexray AData)
    {
        ///用户可以在此添加处理CAN报文AData的逻辑
        if (AData.FIsTx)
        {
          //表示这个报文是从本设备发出去的
        }
        if (AData.FIsExt)
```

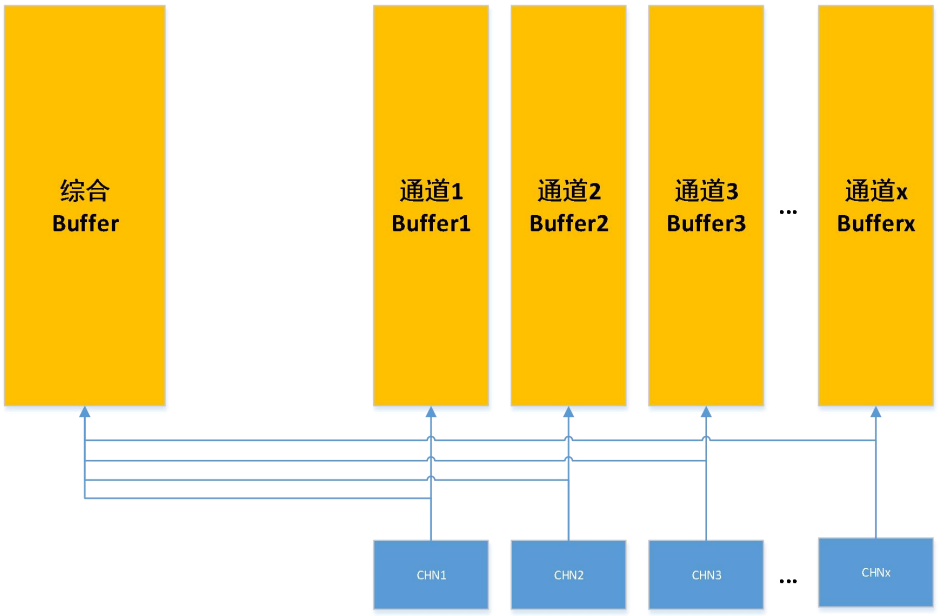
```
            {
              //是否扩展帧
            }
            if (AData.FIsRemote)
            {
              //是否远程帧，否则就是数据帧
            }
            //依次类推
            DispMsg(AData.GetString());
        }
receiveFlexrayCallBack += new TFlexrayQueueEvent_Win32(ReceivedFlexrayMsgCallBack);
```

/// 每当设备发送/收到一帧报文，就会调用此函数。参数：ref TCAN AData 就是对应的这一帧报文。用户可以根据报文的属性来决定如何处理这一帧报文。数据结构定义见章节：数据类型的定义。

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。

CAN 报文获取

获取 Rx 报文：

```
TLIBCAN canBuffer[100] ;
S32 revCnt  = 100;          #revCnt 的大小为 canBuffer 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：

```
TLIBCAN canBuffer[100] ;
S32 revCnt  = 100;          //revCnt 的大小为 canBuffer 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; //revCnt 的大小为 canBuffer 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 canBuffer 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 canBuffer 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 linBuffer 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);
```

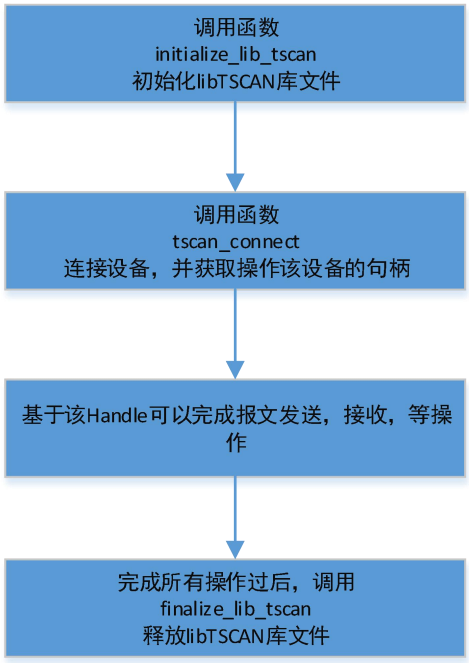
获取 fifo Rx 报文数量：

```
#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：

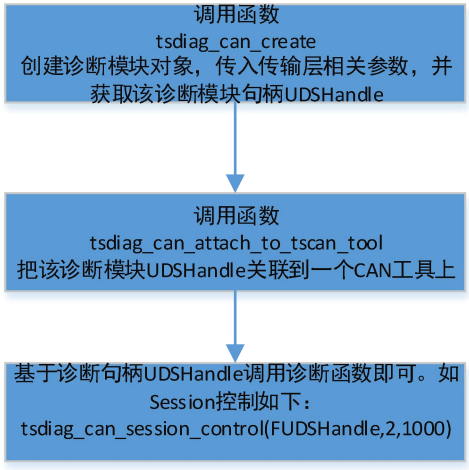
```
#读取通道 0 fifo Rx 的报文数量
tsfifo_clear_flexray_receive_buffers(0);
```

6. libTSCANAPI 调用流程



7. UDS 诊断接口说明

在完成 CAN 工具其他基本配置的基础上，诊断函数使用流程如下：



8. 接口函数介绍

1. initialize_lib_tscan

函数名称	void initialize_lib_tscan(const bool AEnableFIFO, bool AEnableErrorFrame, const bool AEnableTurbe)
功能介绍	初始化 tscan 库模块。必须调用此函数初始化 CAN 模块过后，才能够调用其他 API 函数。该函数和 finalize_lib_tscan 是成对使用的。
调用位置	使用 TSCAN 工具之前，先调用此函数连接设备
输入参数	AEnableFIFO:开启接收 FIFO，推荐设置为 true; AEnableErrorFrame:开启接受错误帧; AEnableTurbo: 是否开启性能模式，推荐设置为 false;
返回值	无
示例	initialize_lib_tscan(true, true, false);

2. finalize_lib_tscan

函数名称	void finalize_lib_tscan(void)
功能介绍	释放 can 模块
调用位置	在退出程序之前，释放 CAN 模块，和创建函数 initialize_lib_tscan 函数配对使用。
输入参数	无
返回值	无
示例	finalize_lib_tscan();

3. tscan_connect

函数名称	UInt32 tscan_connect(string ADeviceSerial, ref IntPtr ADeviceHandle)
功能介绍	连接 TSCAN 工具，并获取该工具的唯一句柄
调用位置	使用 TSCAN 工具之前，先调用此函数连接设备
输入参数	ADeviceSerial:设备序列号 ADeviceHandle:设备句柄
返回值	==0:连接成功 ==5:设备已经连接
示例	IntPtr handle = (IntPtr)0; tscan_connect(ADeviceSerial, ref Handle);

4. tscan_disconnect_by_handle

函数名称	UInt32 tscan_disconnect_by_handle(IntPtr ADeviceHandle)

功能介绍	根据设备句柄，断开该 TSCAN 设备
调用位置	不需要使用设备，调用此函数断开设备连接
输入参数	ADeviceHandle:设备句柄
返回值	==0:断开设备成功 其他值：断开设备失败
示例	IntPtr Handle = (IntPtr)0; tscan_disconnect_by_handle(ref Handle);

5. tscan_scan_devices

函数名称	UInt32 tscan_scan_devices(ref uint ADeviceCount)
功能介绍	扫描当前电脑上存在的 TSCAN 设备数目
调用位置	当用户想知道当前 PC 上 TSCAN 设备数的场合
输入参数	ADeviceCount：引用类型的设备数
返回值	==0：获取成功 其他值：注册失败
示例	UInt32 ADeviceCount = 0; tscan_scan_devices(ref ADeviceCount);

6. tscan_get_device_info

函数名称	UInt32 tscan_get_device_info(Uint ADeviceIndex, ref string AFManufacturer, ref string AFProduct, ref string AFSerial)
功能介绍	根据设备编号获取设备的信息
调用位置	查询设备的详细信息
输入参数	AdeviceIndex：设备索引值 AFManufacturer：生产商名称 AFProduct:设备名称 AFSerial：设备序列号
返回值	==0：查询成功 其他值：注册失败
示例	string AFManufacturer; string AFProduct; string AFSerial; tscan_get_device_info(1, ref AFManufacturer, ref AFProduct, ref AFSerial);

7. tscan_config_can_by_baudrate

函数名称	public static UInt tscan_config_can_by_baudrate(IntPtr ADeviceHandle, CHANNEL_INDEX AChnIdx, Double ARateKbp, Uint A120OhmConnected)
功能介绍	设置 CAN 通道的波特率参数
调用位置	在 CAN 工具连接成功后，调用此函数设置 CAN 工具的波特率
输入参数	ADeviceHandle：设备句柄； AChnIdx:CAN 通道编号 ARateKbp:波特率值，如：1000kbps,500kbps,250kbps,125kbps A120OhmConnected：是否使能终端电阻：1：使能；0：不使能
返回值	==0：函数执行成功 其他值：执行失败
示例	tscan_config_can_by_baudrate(ADeviceHandle, 0, 1000,1);

8. tscan_get_error_description

函数名称	public Uint tscan_get_error_description(Uint ACode, ref IntPtr ADesc)
功能介绍	根据函数执行的返回值编码 ACode，查询该函数的执行结果
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:存储返回的错误详细信息的地址
返回值	==0：函数执行成功 其他值：执行失败
示例	IntPtr ADesc = (IntPtr)0; tscan_get_error_description(23,ref ADesc);

9. tscan_register_event_can

函数名称	UInt32 tscan_register_event_can(IntPtr ADeviceHandle, TCANQueueEvent_Win32 ACallback)
功能介绍	注册 CAN 数据包接收回调函数
调用位置	在 CAN 工具连接成功后，调用此函数注册接收数据的函数
输入参数	ADeviceHandle：设备句柄； ACallback：接收数据处理函数委托
返回值	==0：注册成功 其他值：注册失败
示例	receiveCANCallBack += new TCANQueueEvent_Win32(ReceivedCANMsgCallBack); IntPtr ADeviceHandle = (IntPtr)0;

	tscan_register_event_can(ADeviceHandle, receiveCANCallBack);

10. tscan_unregister_event_can

函数名称	Uint tscan_unregister_event_can(IntPtr ADeviceHandle, TCANQueueEvent_Win32 ACallback)
功能介绍	反注册 CAN 数据接收函数
调用位置	在不需要接收 CAN 报文的时候，调用此函数
输入参数	ADeviceHandle：设备句柄； ACallback：接收数据处理函数委托
返回值	==0：注册成功 其他值：注册失败

示例	receiveCANCallBack += new TCANQueueEvent_Win32(ReceivedCANMsgCallBack);
	IntPtr ADeviceHandle = (IntPtr)0;
	tscan_unregister_event_can(ADeviceHandle ,receiveCANCallBack);

11. tscan_transmit_can_sync

函数名称	Uint tscan_transmit_can_sync(IntPtr ADeviceHandle, ref TCAN ACAN, Uint ATimeoutMS)
功能介绍	以同步的方式发送 CAN 报文
调用位置	发送 CAN 报文
输入参数	ADeviceHandle：设备句柄 ACAN：CAN 数据包 ATimeOutMS：超时判断参数
返回值	==0：注册成功 其他值：注册失败
示例	IntPtr ADeviceHandle = (IntPtr)0; tscan_transmit_can_sync(ADeviceHandle,ref ACAN,100);

12. tscan_transmit_can_async

函数名称	UInt32 tscan_transmit_can_async(IntPtr ADeviceHandle,ref 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	ADeviceHandle：设备句柄 ACAN：CAN 数据包
返回值	==0：注册成功 其他值：注册失败
示例	TLIBCAN ACAN = new TLIBCAN();

	tscan_transmit_can_async(1,ref ACAN);

13. tscan_add_cyclic_msg_can

函数名称	Uint tscan_add_cyclic_msg_can(IntPtr ADeviceHandle, ref TLIBCAN ACAN, float APeriodMS)
功能介绍	周期性发送 CAN 报文
调用位置	在需要周期性发送 CAN 报文的场合
输入参数	ADeviceHandle：设备句柄 ACAN：CAN 数据包 APeriodMS：周期值
返回值	==0：注册成功 其他值：注册失败
示例	TLIBCAN ACAN = new TLIBCAN(); tscan_add_cyclic_msg_can(1,ref ACAN,100);

14. tscan_delete_cyclic_msg_can

函数名称	UInt32 tscan_delete_cyclic_msg_can(IntPtr ADeviceHandle, ref TLIBCAN ACAN)
功能介绍	停止周期性发送 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	ADeviceHandle：设备句柄 ACAN：CAN 数据包
返回值	==0：注册成功 其他值：注册失败
示例	TLIBCAN ACAN = new TLIBCAN(); tscan_delete_cyclic_msg_can(1,ref ACAN);

15. tscan_get_bus_status

函数名称	double tscan_get_bus_status(IntPtr ADeviceHandle, CHANNEL_INDEX AChnBase0, TSTATISTICTYPE AIndex);
功能介绍	获取总线状态
调用位置	
输入参数	ADeviceHandle：设备句柄 AChnBase0： AIndex：
返回值	==0：注册成功 其他值：注册失败
示例	IntPtr ADeviceHandle = (IntPtr)0; tscan_get_bus_status(ADeviceHandle,AChnBase0,AIndex);

16. tscan_set_auto_calc_bus_statistics

函数名称	void tscan_set_auto_calc_bus_statistics(bool Value);
功能介绍	
调用位置	
输入参数	Value:
返回值	==0：注册成功 其他值：注册失败
示例	tscan_set_auto_calc_bus_statistics(true);

17. tscan_clear_can_bus_statistic

函数名称	void tscan_clear_can_bus_statistic();
功能介绍	清除 CAN 总线状态
调用位置	
输入参数	无
返回值	==0：成功 其他值：失败
示例	tscan_clear_can_bus_statistic();

18. tscan_disconnect_all_devices

函数名称	uint tscan_disconnect_all_devices();
功能介绍	断开所有设备
调用位置	**测试结束时**
输入参数	无
返回值	==0：成功 其他值：失败
示例	tscan_disconnect_all_devices();

19. tscan_disconnect_by_serial

函数名称	uint tscan_disconnect_by_serial(string ADeviceSerial);
功能介绍	通过序列号断开设备
调用位置	**测试结束时**
输入参数	ADeviceSerial：设备序列号
返回值	==0：成功 其他值：失败
示例	tscan_disconnect_all_devices(ADeviceSerial);

20. tscan_register_event_canfd

函数名称	uint tscan_register_event_canfd(IntPtr ADeviceHandle,

	TCANFDQueueEvent_Win32 ACallback);
功能介绍	注册 canfd 事件
调用位置	**注册 CANfd 回调函数**
输入参数	ADeviceHandle：设备句柄 ACallback:回调函数
返回值	==0：成功 其他值：失败
示例	IntPtr ADeviceHandle= (IntPtr)0; tscan_register_event_canfd(ADeviceHandle,ACallback);

21. tscan_unregister_event_canfd

函数名称	uint tscan_unregister_event_canfd(IntPtr ADeviceHandle, TCANFDQueueEvent_Win32 ACallback);
功能介绍	**注销 canfd 事件**
调用位置	**注销 canfd 回调函数时**
输入参数	ADeviceHandle：设备句柄 ACallback:回调函数
返回值	==0：成功 其他值：失败
示例	IntPtr ADeviceHandle= (IntPtr)0; tscan_unregister_event_canfd(ADeviceHandle, ACallback);

22. tslin_register_event_lin

函数名称	uint tslin_register_event_lin(IntPtr ADeviceHandle, TLINQueueEvent_Win32 ACallback);
功能介绍	**注册 lin 事件**
调用位置	**注册 lin 回调函数时**
输入参数	ADeviceHandle：设备句柄 ACallback:回调函数
返回值	==0：成功 其他值：失败
示例	IntPtr ADeviceHandle= (IntPtr)0; tslin_register_event_lin(ADeviceHandle, ACallback);

23. tslin_unregister_event_lin

函数名称	uint tslin_unregister_event_lin(IntPtr ADeviceHandle, TLINQueueEvent_Win32 ACallback);
功能介绍	**注销 lin 事件**
调用位置	**注销 lin 回调函数时**
输入参数	ADeviceHandle：设备句柄 ACallback:回调函数
返回值	==0：成功

	其他值：失败
示例	IntPtr ADeviceHandle= (IntPtr)0; tslin_unregister_event_lin(ADeviceHandle, ACallback);

24. tsfifo_receive_can_msgs

函数名称	uint tsfifo_receive_can_msgs(IntPtr ADeviceHandle, IntPtr ACANBuffer, ref int ABufferSize, CHANNEL_INDEX AChn, READ_TX_RX_DEF ATxRx);
功能介绍	接收 can 报文消息缓存
调用位置	
输入参数	ADeviceHandle：设备句柄 ACANBuffer：can 报文缓冲区 ABufferSize：缓冲区大小 AChn：通道 ATxRx：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
返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	int ABufferSize = 0;
	tsfifo_receive_can_msgs(ADeviceHandle,ACANBuffer,ref ABufferSize, 1, 1);

25. tsfifo_clear_can_receive_buffers

函数名称	uint tsfifo_clear_can_receive_buffers(IntPtr ADeviceHandle, CHANNEL_INDEX AChnIndex);
功能介绍	接收 can 报文消息缓存
调用位置	
输入参数	ADeviceHandle：设备句柄 ACANBuffer：can 报文缓冲区 ABufferSize：缓冲区大小 AChn：通道 ATxRx：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
返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	int ABufferSize = 0;
	tsfifo_receive_can_msgs(ADeviceHandle,ACANBuffer,ref ABufferSize, 1, 1);

26. tsfifo_read_can_buffer_frame_count

函数名称	int tsfifo_read_can_buffer_frame_count(IntPtr ADeviceHandle, CHANNEL_INDEX AChannel, ref int ACount);
功能介绍	读取 can 缓冲区报文数量
调用位置	
输入参数	ADeviceHandle：设备句柄 AChannel：通道 ACount：返回的报文数量
返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	int ACount = 0;
	tsfifo_read_can_buffer_frame_count(ADeviceHandle,1, ref ACount);

27. tsfifo_read_can_tx_buffer_frame_count

函数名称	int tsfifo_read_can_tx_buffer_frame_count(IntPtr ADeviceHandle, CHANNEL_INDEX AChannel, ref int ACount);
功能介绍	读取 can Tx 缓冲区报文数量
调用位置	
输入参数	ADeviceHandle：设备句柄 AChannel：通道 ACount：返回的报文数量
返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	int ACount = 0;
	tsfifo_read_can_tx_buffer_frame_count(ADeviceHandle,1, ref ACount);

28. tsfifo_read_can_rx_buffer_frame_count

函数名称	int tsfifo_read_can_rx_buffer_frame_count(IntPtr ADeviceHandle, CHANNEL_INDEX AChannel, ref int ACount);
功能介绍	读取 can rx 缓冲区报文数量
调用位置	
输入参数	ADeviceHandle：设备句柄 AChannel：通道 ACount：返回的报文数量
返回值	==0：成功 其他值：失败
示例	intPtr ADeviceHandle= (IntPtr)0; int ACount = 0;

	tsfifo_read_can_rx_buffer_frame_count(ADeviceHandle,1, ref ACount);

29. tsfifo_read_canfd_buffer_frame_count

函数名称	int tsfifo_read_canfd_buffer_frame_count(IntPtr ADeviceHandle, CHANNEL_INDEX AChannel, ref int ACount);
功能介绍	读取 canfd 缓冲区报文数量
调用位置	
输入参数	ADeviceHandle：设备句柄 AChannel:通道 ACount：返回的报文数量
返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	int ACount = 0;
	tsfifo_read_can_rx_buffer_frame_count(ADeviceHandle,1, ref ACount);

30. tsfifo_read_canfd_tx_buffer_frame_count

函数名称	int tsfifo_read_canfd_tx_buffer_frame_count(IntPtr ADeviceHandle, CHANNEL_INDEX AChannel, ref int ACount);
功能介绍	读取 canfd tx 缓冲区报文数量
调用位置	
输入参数	ADeviceHandle：设备句柄 AChannel：通道 ACount：返回的报文数量
返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	int ACount = 0;
	tsfifo_read_can_tx_buffer_frame_count(ADeviceHandle,1, ref ACount);

31. tsfifo_read_canfd_rx_buffer_frame_count

函数名称	int tsfifo_read_canfd_rx_buffer_frame_count(IntPtr ADeviceHandle, CHANNEL_INDEX AChannel, ref int ACount);
功能介绍	读取 canfd rx 缓冲区报文数量
调用位置	
输入参数	ADeviceHandle：设备句柄 AChannel：通道 ACount：返回的报文数量
返回值	==0：成功

	其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	int ACount = 0;
	tsfifo_read_can_rx_buffer_frame_count(ADeviceHandle,1, ref ACount);

32. tsfifo_read_lin_buffer_datacount

函数名称	uint tsfifo_read_lin_buffer_datacount(IntPtr ADeviceHandle, CHANNEL_INDEX AChannel, ref int ACount);
功能介绍	读取 lin 缓冲区报文数量
调用位置	
输入参数	ADeviceHandle：设备句柄 AChannel：通道 ACount：返回的报文数量
返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	int ACount = 0;
	tsfifo_read_lin_buffer_datacount(ADeviceHandle,1, ref ACount);

33. tsfifo_read_lin_tx_buffer_frame_count

函数名称	uint tsfifo_read_lin_tx_buffer_frame_count(IntPtr ADeviceHandle, CHANNEL_INDEX AChannel, ref int ACount);
功能介绍	读取 lin tx 缓冲区报文数量
调用位置	
输入参数	ADeviceHandle：设备句柄 AChannel：通道 ACount：返回的报文数量
返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	int ACount = 0;
	tsfifo_read_lin_tx_buffer_datacount(ADeviceHandle,1, ref ACount);

34. tsfifo_read_lin_rx_buffer_frame_count

函数名称	uint tsfifo_read_lin_rx_buffer_frame_count(IntPtr ADeviceHandle, CHANNEL_INDEX AChannel, ref int ACount);
功能介绍	读取 lin rx 缓冲区报文数量
调用位置	
输入参数	ADeviceHandle：设备句柄 AChannel：通道 ACount：返回的报文数量

返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	int ACount = 0;
	tsfifo_read_lin_rx_buffer_datacount(ADeviceHandle,1, ref ACount);

35. tsfifo_read_fastlin_buffer_datacount

函数名称	uint tsfifo_read_fastlin_buffer_datacount(IntPtr ADeviceHandle, CHANNEL_INDEX AChannel, ref int ACount);
功能介绍	读取 fastlin 缓冲区报文数量
调用位置	
输入参数	ADeviceHandle：设备句柄 AChannel：通道 ACount：返回的报文数量
返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	int ACount = 0;
	tsfifo_read_fastlin_buffer_datacount(ADeviceHandle,1,ref ACount);

36. tsfifo_read_fastlin_tx_buffer_frame_count

函数名称	uint tsfifo_read_fastlin_tx_buffer_frame_count(IntPtr ADeviceHandle, CHANNEL_INDEX AChannel, ref int ACount);
功能介绍	读取 fastlin tx 缓冲区报文数量
调用位置	
输入参数	ADeviceHandle：设备句柄 AChannel：通道 ACount：返回的报文数量
返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	int ACount = 0;
	tsfifo_read_lin_tx_buffer_datacount(ADeviceHandle,1, ref ACount);

37. tsfifo_read_fastlin_rx_buffer_frame_count

函数名称	uint tsfifo_read_fastlin_rx_buffer_frame_count(IntPtr ADeviceHandle, CHANNEL_INDEX AChannel, ref int ACount);
功能介绍	读取 fastlin rx 缓冲区报文数量
调用位置	
输入参数	ADeviceHandle：设备句柄 AChannel：通道

	ACount：返回的报文数量
返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	int ACount = 0;
	tsfifo_read_lin_rx_buffer_datacount(ADeviceHandle,1, ref ACount);

38. tscan_transmit_canfd_sync

函数名称	uint tscan_transmit_canfd_sync(IntPtr ADeviceHandle, ref TLIBCANFD ACANFD, uint ATimeoutMS);
功能介绍	异步发送 canfd 报文
调用位置	
输入参数	ADeviceHandle：设备句柄 ACANFD：canfd 报文结构体指针 ATimeoutMS：事件戳
返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	TLIBCANFD tLibCANFD = new TLIBCANFD();
	tscan_transmit_canfd_sync(ADeviceHandle,ref ACANFD, 100);

39. tscan_transmit_canfd_async

函数名称	uint uint tscan_transmit_canfd_async(IntPtr ADeviceHandle, ref TLIBCANFD ACANFD);
功能介绍	同步发送 canfd 报文
调用位置	
输入参数	ADeviceHandle：设备句柄 ACANFD：canfd 报文结构体指针
返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	TLIBCANFD tLibCANFD = new TLIBCANFD();
	tscan_transmit_canfd_async(ADeviceHandle,ref ACANFD);

40. tscan_add_cyclic_msg_canfd

函数名称	uint tscan_add_cyclic_msg_canfd(IntPtr ADeviceHandle, ref TLIBCANFD ACANFD, float APeriodMS);
功能介绍	添加 canfd 周期报文
调用位置	
输入参数	ADeviceHandle：设备句柄 ACANFD：canfd 报文结构体指针

	APeriodMS：周期时间
返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	TLIBCANFD tLibCANFD = new TLIBCANFD();
	tscan_transmit_canfd_async(ADeviceHandle,ref ACANFD, 100.0);

41. tscan_delete_cyclic_msg_canfd

函数名称	uint tscan_delete_cyclic_msg_canfd(IntPtr ADeviceHandle, ref TLIBCANFD ACANFD);
功能介绍	**删除 canfd 周期报文**
调用位置	
输入参数	ADeviceHandle：设备句柄 ACANFD：canfd 报文结构体指针
返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	TLIBCANFD tLibCANFD = new TLIBCANFD();
	tscan_transmit_canfd_async(ADeviceHandle,ref ACANFD);

42. tsfifo_receive_canfd_msgs

函数名称	uint tsfifo_receive_canfd_msgs(IntPtr ADeviceHandle, IntPtr ACANFDBuffers, ref int ACANFDBufferSize, CHANNEL_INDEX AChn, READ_TX_RX_DEF ATxRx);
功能介绍	**接收缓冲区 canfd 报文**
调用位置	
输入参数	ADeviceHandle：设备句柄 ACANFDBuffers：CANFD 报文缓冲区 ACANFDBufferSize：缓冲区大小 AChn：通道 ATxRx：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
返回值	==0：成功 其他值：失败
示例	intPtr ADeviceHandle= (IntPtr)0; tsfifo_receive_canfd_msgs(ADeviceHandle,ACANFDBuffers,ref ACANFDBufferSize,1, 1);

43. tsfifo_clear_canfd_receive_buffers

函数名称	uint tsfifo_clear_canfd_receive_buffers(IntPtr ADeviceHandle, CHANNEL_INDEX AChnIndex);

功能介绍	**清除 canfd 缓冲区**
调用位置	
输入参数	ADeviceHandle：设备句柄 AChnIndex：通道索引
返回值	==0：成功 其他值：失败
示例	intPtr ADeviceHandle= (IntPtr)0; tsfifo_clear_canfd_receive_buffers(ADeviceHandle, 0);

44. tsfifo_add_can_canfd_pass_filter

函数名称	uint tsfifo_add_can_canfd_pass_filter(IntPtr ADeviceHandle, CHANNEL_INDEX AChnIdx, int AIdentifier, bool AIsStd);
功能介绍	**添加 can 和 canfd 过滤**
调用位置	
输入参数	ADeviceHandle：设备句柄 AChnIndex：通道索引 AIdentifier：标识符 AIsStd：是否为标准帧
返回值	==0：成功 其他值：失败
示例	intPtr ADeviceHandle= (IntPtr)0; tsfifo_add_can_canfd_pass_filter(ADeviceHandle,0,0x123,true);

45. tsfifo_add_lin_pass_filter

函数名称	uint tsfifo_add_lin_pass_filter(IntPtr ADeviceHandle, CHANNEL_INDEX AChnIdx, int AIdentifier);
功能介绍	**添加 lin 过滤**
调用位置	
输入参数	ADeviceHandle：设备句柄 AChnIdx：通道索引 AIdentifier：标识符
返回值	==0：成功 其他值：失败
示例	intPtr ADeviceHandle= (IntPtr)0; tsfifo_add_lin_pass_filter(ADeviceHandle,0,0x123);

46. tsfifo_delete_can_canfd_pass_filter

函数名称	uint tsfifo_delete_can_canfd_pass_filter(IntPtr ADeviceHandle, CHANNEL_INDEX AChnIdx, int AIdentifier);
功能介绍	**删除 can 和 canfd 过滤器**
调用位置	

输入参数	ADeviceHandle：设备句柄 AChnIdx：通道索引 AIdentifier：标识符
返回值	==0：成功 其他值：失败
示例	intPtr ADeviceHandle= (IntPtr)0; tsfifo_add_can_canfd_pass_filter(ADeviceHandle,0,0x123);

47. tsfifo_delete_lin_pass_filter

函数名称	uint tsfifo_delete_lin_pass_filter(IntPtr ADeviceHandle, CHANNEL_INDEX AChnIdx, int AIdentifier);
功能介绍	**删除 lin 过滤器**
调用位置	
输入参数	ADeviceHandle：设备句柄 AChnIdx：通道索引 AIdentifier：标识符
返回值	==0：成功 其他值：失败
示例	intPtr ADeviceHandle= (IntPtr)0; tsfifo_delete_lin_pass_filter(ADeviceHandle,0,0x123);

48. tslin_transmit_lin_sync

函数名称	uint tslin_transmit_lin_sync(IntPtr ADeviceHandle, ref TLIBLIN ALIN, uint ATimeoutMS);
功能介绍	**同步发送 lin 报文**
调用位置	
输入参数	ADeviceHandle：设备句柄 ALIN：lin 报文结构体指针 ATimeoutMS：超时时间
返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	TLIBLIN tLibLIN = new TLIBLIN();
	tslin_transmit_lin_sync(ADeviceHandle,ref tLibLIN ,100);

49. tslin_transmit_lin_async

函数名称	uint tslin_transmit_lin_async(IntPtr ADeviceHandle, ref TLIBLIN ALIN);
功能介绍	**异步发送 lin 报文**
调用位置	
输入参数	ADeviceHandle：设备句柄 ALIN：lin 报文结构体指针

返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	TLIBLIN tLibLIN = new TLIBLIN();
	tslin_transmit_lin_async(ADeviceHandle,ref tLibLIN);

50. tsfifo_receive_lin_msgs

函数名称	uint tsfifo_receive_lin_msgs(IntPtr ADeviceHandle, IntPtr ALINBuffer, ref int ABufferSize, CHANNEL_INDEX AChn, READ_TX_RX_DEF ATxRx);
功能介绍	接收 lin 报文
调用位置	
输入参数	ADeviceHandle：设备句柄 ALINBuffer：lin 缓冲区 ABufferSize：缓冲区大小 AChn：通道 ATxRx：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
返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	Int ABufferSize = 0;
	tsfifo_receive_lin_msgs(ADeviceHandle,ALINBuffer,ref ABufferSize,0,1);

51. tsfifo_clear_lin_receive_buffers

函数名称	uint tsfifo_clear_lin_receive_buffers(IntPtr ADeviceHandle, CHANNEL_INDEX AChn);
功能介绍	接收 lin 报文
调用位置	
输入参数	ADeviceHandle：设备句柄 AChn：通道
返回值	==0：成功 其他值：失败
示例	intPtr ADeviceHandle= (IntPtr)0; tsfifo_clear_lin_receive_buffers(ADeviceHandle,0);

52. tsfifo_receive_fastlin_msgs

函数名称	uint tsfifo_receive_fastlin_msgs(IntPtr ADeviceHandle, IntPtr AFastLINBuffer, ref int ABufferSize, CHANNEL_INDEX AChn, READ_TX_RX_DEF ATxRx);

功能介绍	接收 fastlin 报文
调用位置	
输入参数	ADeviceHandle：设备句柄 ALINBuffer：fastlin 缓冲区 ABufferSize：缓冲区大小 AChn：通道 ATxRx：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
返回值	==0：成功 其他值：失败

示例	intPtr ADeviceHandle= (IntPtr)0;
	int ABufferSize = 0;
	tsfifo_receive_fastlin_msgs(ADeviceHandle,ALINBuffer,ref ABufferSize,0,1);

53. tsfifo_clear_fastlin_receive_buffers

函数名称	uint tsfifo_clear_fastlin_receive_buffers(IntPtr ADeviceHandle, CHANNEL_INDEX AChnIndex);
功能介绍	清除 fastlin 接收缓冲区
调用位置	
输入参数	ADeviceHandle：设备句柄 AChnIndex:通道索引
返回值	==0：成功 其他值：失败
示例	intPtr ADeviceHandle= (IntPtr)0; tsfifo_clear_fastlin_receive_buffers(ADeviceHandle,0);

54. tscan_register_event_connected

函数名称	uint tscan_register_event_connected(TTSCANConnectedCallback_Win32 ACallback);
功能介绍	注册连接事件
调用位置	
输入参数	ACallback：回调函数
返回值	==0：成功 其他值：失败
示例	tscan_register_event_connected(ACallback);

55. tscan_register_event_disconnected

函数名称	uint tscan_register_event_disconnected(TTSCANConnectedCallback_Win32 ACallback);
功能介绍	注册断开事件

调用位置	
输入参数	ACallback：回调函数
返回值	==0：成功 其他值：失败
示例	tscan_register_event_disconnected(ACallback);

56. tscan_unregister_event_connected

函数名称	uint tscan_unregister_event_connected(TTSCANConnectedCallback_Win32 ACallback);
功能介绍	**注销连接事件**
调用位置	
输入参数	ACallback：回调函数
返回值	==0：成功 其他值：失败
示例	tscan_unregister_event_connected(ACallback);

57. tscan_unregister_event_disconnected

函数名称	uint tscan_unregister_event_disconnected(TTSCANConnectedCallback_Win32 ACallback);
功能介绍	**注销连接事件**
调用位置	
输入参数	ACallback：回调函数
返回值	==0：成功 其他值：失败
示例	tscan_unregister_event_connected(ACallback);

58. tslin_config_baudrate

函数名称	uint tslin_config_baudrate(IntPtr ADeviceHandle, CHANNEL_INDEX AChnIdx, double ARateKbps, LIN_PROTOCOL AProtocol);
功能介绍	**配置 lin 波特率**
调用位置	
输入参数	ADeviceHandle：设备句柄 AChnIdx：通道索引 ARateKbps：波特率 AProtocol：lin 协议 包括{LIN_PROTOCOL_13, LIN_PROTOCOL_20, LIN_PROTOCOL_21, LIN_PROTOCOL_J2602}

返回值	==0：成功 其他值：失败
示例	IntPtr ADeviceHandle = (IntPtr)0; tslin_config_baudrate(ADeviceHandle,0,2000,LIN_PROTOCOL_13);

59. tslin_set_schedule_table

函数名称	uint tslin_set_schedule_table(IntPtr ADeviceHandle, CHANNEL_INDEX AChnIdx, byte ASchIndex);
功能介绍	**设置目录表**
调用位置	
输入参数	ADeviceHandle：设备句柄 AChnIdx：通道索引 ASchIndex：目录索引
返回值	==0：成功 其他值：失败
示例	IntPtr ADeviceHandle = (IntPtr)0; Byte[] ASchIndex; tslin_set_schedule_table(ADeviceHandle, C0, ASchIndex);

60. tslin_stop_lin_channel

函数名称	uint tslin_stop_lin_channel(IntPtr ADeviceHandle, CHANNEL_INDEX AChnIdx);
功能介绍	**断开 lin 通道**
调用位置	
输入参数	ADeviceHandle：设备句柄 AChnIdx：通道索引
返回值	==0：成功 其他值：失败
示例	IntPtr ADeviceHandle = (IntPtr)0; tslin_stop_lin_channel(ADeviceHandle, 0);

61. tslin_set_node_funtiontype

函数名称	uint tslin_set_node_funtiontype(IntPtr ADeviceHandle, CHANNEL_INDEX AChnIdx, T_LIN_NODE_FUNCTION AFunctionType);
功能介绍	**设置 lin 节点函数类型**
调用位置	
输入参数	ADeviceHandle：设备句柄 AChnIdx：通道索引 AFunctionType:函数类型 0：T_MasterNode 1：T_SlaveNode 2：T_MonitorNode
返回值	==0：成功 其他值：失败
示例	IntPtr ADeviceHandle = (IntPtr)0;

	tslin_set_node_funtiontype(ADeviceHandle, 0, T_MASTER_NODE);

62. tslin_clear_schedule_tables

函数名称	uint tslin_clear_schedule_tables(IntPtr ADeviceHandle, CHANNEL_INDEX AChnIdx);
功能介绍	清除 lin 进度表
调用位置	
输入参数	ADeviceHandle：设备句柄 AChnIdx：通道索引
返回值	==0：成功 其他值：失败
示例	IntPtr ADeviceHandle = (IntPtr)0; tslin_clear_schedule_tables(ADeviceHandle, 0);

63. tstp_lin_reset

函数名称	uint tstp_lin_reset(IntPtr ADeviceHandle, uint AChnIdx);
功能介绍	Lin 诊断请求
调用位置	
输入参数	ADeviceHandle：设备句柄 AChnIdx：通道索引
返回值	==0：成功 其他值：失败
示例	IntPtr ADeviceHandle = (IntPtr)0; tstp_lin_reset(ADeviceHandle,0);

64. tstp_lin_slave_response_intervalms

函数名称	uint tstp_lin_slave_response_intervalms(IntPtr ADeviceHandle, CHANNEL_INDEX AChnIdx, byte AData);
功能介绍	控制 intervalms 响应
调用位置	
输入参数	ADeviceHandle：设备句柄 AChnIdx：通道索引 AData:数据信息
返回值	==0：成功 其他值：失败
示例	IntPtr ADeviceHandle = (IntPtr)0; tstp_lin_slave_response_intervalms(ADeviceHandle,0, AData);

65. tsdiag_lin_read_data_by_identifier

函数名称	uint tstp_lin_slave_response_intervalms(IntPtr ADeviceHandle, CHANNEL_INDEX AChnIdx, byte AData);
功能介绍	控制 intervalms 响应
调用位置	
输入参数	ADeviceHandle：设备句柄 AChnIdx：通道索引 AData:数据信息
返回值	==0：成功 其他值：失败
示例	IntPtr ADeviceHandle = (IntPtr)0; tstp_lin_slave_response_intervalms(ADeviceHandle,0, AData);

66. tsdiag_lin_session_control

函数名称	int tsdiag_lin_session_control(IntPtr ADeviceHandle, CHANNEL_INDEX AChnIdx, byte ANAD, byte ASession, uint ATimeoutMS);
功能介绍	Lin session 控制器诊断
调用位置	
输入参数	ADeviceHandle：设备句柄 AChnIdx：通道索引 ANAD:数据信息 ASession： ATimeoutMS：超时时间
返回值	==0：成功 其他值：失败
示例	IntPtr ADeviceHandle = (IntPtr)0; tsdiag_lin_session_control(ADeviceHandle,0, ANAD,Session,100);

67. tsdiag_can_create

函数名称	UInt32 tsdiag_can_create(ref int pDiagModuleIndex, CHANNEL_INDEX AChnIndex, byte ASupportFDCAN, byte AMaxDLC, uint ARequestID, bool ARequestIDIsStd, uint AResponseID, bool AResponseIDIsStd, uint AFunctionID, bool AFunctionIDIsStd);
功能介绍	创建 can 诊断模块
调用位置	
输入参数	pDiagModuleIndex:诊断模块索引 AChnIndex: 通道索引 ASupportFDCAN:支持的 CANFD AMaxDLC:最大 DLC 长度 ARequestID:请求 ID ARequestIDIsStd:请求 ID 是否标准

	AResponseID:响应 ID AResponseIDIsStd:响应 ID 是否标准 AFunctionID:功能 ID AFunctionIDIsStd：功能 ID 是否标准
返回值	==0:创建成功 其他值:创建失败
示例	int pDiaModuleIndex = 0; tsdiag_can_create(ref pDiagModuleIndex,0,ASupportFDCAN, AMaxDLC, ARequestID, true, AResponseID,true, AFunctionID, true);

68. tsdiag_can_delete

函数名称	int tsdiag_can_delete(int ADiagModuleIndex);
功能介绍	**卸载 can 诊断**
调用位置	
输入参数	DiagModuleIndex：诊断模块索引
返回值	==0：成功 其他值：失败
示例	tsdiag_can_delete(0);

69. tsdiag_can_delete_all

函数名称	void tsdiag_can_delete_all();
功能介绍	**卸载所有 can 诊断**
调用位置	
输入参数	无
返回值	==0：成功 其他值：失败
示例	tsdiag_can_delete_all();

70. tsdiag_can_attach_to_tscan_tool

函数名称	int tsdiag_can_attach_to_tscan_tool(int ADiagModuleIndex, IntPtr ACANToolHandle);
功能介绍	**固定 can 诊断模块到 tscan 工具上**
调用位置	
输入参数	ADiagModuleIndex：诊断模块索引 ACANToolHandle：can 工具句柄
返回值	==0：成功 其他值：失败
示例	IntPtr ACANToolHandle = (IntPtr)0;

	tsdiag_can_attach_to_tscan_tool(0, ACANToolHandle);

71. tstp_can_register_tx_completed_recall

函数名称	int tstp_can_register_tx_completed_recall(int ADiagModuleIndex, T_N_USData_TranslateCompleted_Recall ATxcompleted)
功能介绍	注册完整的 can 回归传输
调用位置	
输入参数	ADiagModuleIndex:uds 模块的句柄 ATxcompleted:传输完成
返回值	==0：执行成功 其他值：查询错误码
示例	T_N_USData_TranslateCompleted_Recall ATxcompleted; tstp_can_register_tx_completed_recall(udsHandle, ATxcompleted)

72. tstp_can_register_rx_completed_recall

函数名称	int tstp_can_register_rx_completed_recall(int ADiagModuleIndex, T_N_USData_TranslateCompleted_Recall ARxcompleted)
功能介绍	注册 CAN 完整的回归接收
调用位置	
输入参数	ADiagModuleIndex:uds 模块的句柄 ARxcompleted:接收完成
返回值	==0：执行成功 其他值：查询错误码
示例	T_N_USData_TranslateCompleted_Recall ARxcompleted; tstp_can_register_rx_completed_recall(udsHandle, ARxcompleted);

73. tstp_can_send_functional

函数名称	int tstp_can_send_functional(int ADiagModuleIndex, byte* AReqDataArray, int AReqDataSize)
功能介绍	功能 id 请求数据
调用位置	
输入参数	ADiagModuleIndex:诊断模块索引 AReqDataArray:指向请求数据源的数据指针 AReqDataSize:数据源数组的大小（字节数）
返回值	==0：执行成功 其他值：查询错误码

示例	byte [] reqDataArray= {0x3E, 0x80};
	if(TsMasterApi.tstp_can_send_functional(udsHandle, reqDataArray, 2) == 0x00)
	{
	Console.WriteLine("send functional payload Success!");

	}

74. tstp_can_send_request

函数名称	int tstp_can_send_request(int ADiagModuleIndex, byte* AReqDataArray, int AReqDataSize)
功能介绍	请求 id 请求数据
调用位置	
输入参数	ADiagModuleIndex:uds 模块的句柄 AReqDataArray:指向请求数据源的数据指针 AReqDataSize:数据源数组的大小（字节数）
返回值	==0：执行成功 其他值：查询错误码

示例	byte [] reqDataArray= {0x10,0x03};
	if(tstp_can_send_request(udsHandle, reqDataArray, 2) == 0x00)
	{
	Console.WriteLine("send diagnostic payload Success!");
	}

75. tstp_can_request_and_get_response

函数名称	int tstp_can_request_and_get_response(int ADiagModuleIndex, byte* AReqDataArray, int AReqDataSize, byte* AResponseDataArray, ref int AResponseDataSize)
功能介绍	请求 id 请求数据并获取响应数据
调用位置	
输入参数	ADiagModuleIndex:uds 模块的句柄 AReqDataArray:指向请求数据源的数据指针 AReqDataSize:数据源数组的大小（字节数） AResponseDataArray:指向响应数据缓冲区的数据指针 AResponseDataSize:指向整数值的数据指针，用于保存响应数据大小
返回值	==0：执行成功 其他值：查询错误码

示例	byte reqDataArray= {0x10,0x03};
	byte responseArray[10];
	int responseArraySize = 10;
	if(tstp_can_request_and_get_response(udsHandle, reqDataArray, 2, responseArray, &responseArraySize) == 0x00)
	{
	Console.WriteLine("send diagnostic payload and get response success!");
	}

76. tsdiag_can_session_control

函数名称	int tsdiag_can_session_control(int ADiagModuleIndex, byte ASubSession)
功能介绍	设置会话控制
调用位置	
输入参数	ADiagModuleIndex:uds 模块的句柄 ASubSession:子会话值
返回值	==0：执行成功 其他值：查询错误码

示例	if(tsdiag_can_session_control(udsHandle, 2) == 0x00)
	{
	Console.WriteLine("switch sesion:2 success");
	}
	else
	{
	Console.WriteLine("switch sesion:2 failed");
	}

77. tsdiag_can_routine_control

函数名称	int tsdiag_can_routine_control(int ADiagModuleIndex, byte ARoutineControlType, ushort ARoutintID)
功能介绍	设置 can 常规控制类型值
调用位置	
输入参数	ADiagModuleIndex:uds 模块的句柄 ARoutineControlType:例行控制类型 ARoutintID:例行 id
返回值	==0：执行成功 其他值：查询错误码

示例	if(tsdiag_can_routine_control(udsHandle,0x01, 0x9178) == 0x00)
	{
	Console.WriteLine("routine control [type:0x01; id:0x9178] success");
	}
	else
	{
	Console.WriteLine("routine control:%d failed", ENABLE_RX_TX);
	}

78. tsdiag_can_communication_control

函数名称	int tsdiag_can_communication_control(int ADiagModuleIndex, byte AControlType)

功能介绍	设置 can 通信控制类型
调用位置	
输入参数	ADiagModuleIndex:uds 模块的句柄 AControlType:通信控制类型
返回值	==0：执行成功 其他值：查询错误码

示例	if(tsdiag_can_communication_control(udsHandle, 0x00) == 0x00)
	{
	Console.WriteLine("communication control:0x00 success");
	}
	else
	{
	Console.WriteLine("communication control:0x00 failed");
	}

79. tsdiag_can_security_access_request_seed

函数名称	int tsdiag_can_security_access_request_seed(int ADiagModuleIndex, int ALevel, byte* ARecSeed, ref int ARecSeedSize)
功能介绍	27 seed 获取 seed
调用位置	连接硬件之后，需要创建诊断模块后，不再需要该诊断模块
输入参数	ADiagModuleIndex:uds 模块的句柄 ALevel:安全级别 ARecSeed:用于保存种子值的数据指针 ARecSeedSize:种子值的大小
返回值	==0：执行成功 其他值：查询错误码

示例	byte seedBuffer[4]; //seedBuffer to save the seed received from the DUT
	int seedBufferSize = 4; //byte num of the seed value
	if(tsdiag_can_security_access_request_seed(udsHandle, 0x01, seedBuffer,ref seedBufferSize) == 0x00)
	{
	Console.WriteLine("get seed success!");
	}
	else
	{
	Console.WriteLine("get seed failed!");
	}

80. tsdiag_can_security_access_send_key

函数名称	int tsdiag_can_security_access_send_key(int ADiagModuleIndex, int ALevel, byte* AKeyValue, int AKeySize)

功能介绍	27 key 发送 key
调用位置	
输入参数	ADiagModuleIndex:uds 模块的句柄 ALevel:安全级别 AKeyValue:用于保存键值的数据指针 AKeySize:键值大小
返回值	==0：执行成功 其他值：查询错误码

示例	byte keyValue[4]; //Calculated from seed value
	if(tsdiag_can_security_access_send_key(udsHandle, 0x02, keyValue, 4) == 0x00)
	{
	Console.WriteLine("unlock the ecu success!");
	}
	else
	{
	Console.WriteLine("unlock the ecu failed");
	}

81. tsdiag_can_request_download

函数名称	int tsdiag_can_request_download(int ADiagModuleIndex, uint AMemAddr, uint AMemSize)
功能介绍	客户端使用请求下载服务来启动从客户端到服务器的 can 数据传输
调用位置	
输入参数	ADiagModuleIndex:uds 模块的句柄 AMemAddr:存储器地址 AMemSize:存储器大小
返回值	==0：执行成功 其他值：查询错误码

示例	if(tsdiag_can_request_download(udsHandle, 0x08000000, 0x00010000) == 0x00)
	{
	Console.WriteLine("require download success!");
	}
	else
	{
	Console.WriteLine("require download failed!");
	}

82. tsdiag_can_request_upload

函数名称	int tsdiag_can_request_upload(int ADiagModuleIndex, uint AMemAddr,

	uint AMemSize)
功能介绍	请求上传 CAN 数据包
调用位置	
输入参数	ADiagModuleIndex:uds 模块的句柄 AMemAddr:存储器地址 AMemSize:存储器大小
返回值	==0：执行成功 其他值：查询错误码

示例	if(tsdiag_can_request_upload(udsHandle, 0x08000000, 0x00010000) == 0x00)
	{
	Console.WriteLine("require upload success!");
	}
	else
	{
	Console.WriteLine("require upload failed!");
	}

83. tsdiag_can_transfer_data

函数名称	int tsdiag_can_transfer_data(int ADiagModuleIndex, byte* ASourceDatas, int ADataSize, int AReqCase)
功能介绍	启动传输数据包
调用位置	连接硬件之后，需要创建诊断模块后，不再需要该诊断模块
输入参数	ADiagModuleIndex:uds 模块的句柄 ASourceDatas:指向数据缓冲区的数据指针 ADataSize:数据集大小（字节数） AReqCase:req 情况
返回值	==0：执行成功 其他值：查询错误码

示例	byte hexSourceData[0x10000]; //buffer saved hex data
	if(tsdiag_can_transfer_data(udsHandle, hexSourceData, 0x10000, 0x00) == 0x00)
	{
	Console.WriteLine("transfer data success!");
	}
	else
	{
	Console.WriteLine("transfer data failed!");
	}

84. tsdiag_can_request_transfer_exit

函数名称	int tsdiag_can_request_transfer_exit(int ADiagModuleIndex)
功能介绍	请求停止并退出 can 数据传输
调用位置	连接硬件之后，需要创建诊断模块后，不再需要该诊断模块
输入参数	ADiagModuleIndex:uds 模块的句柄
返回值	==0：执行成功 其他值：查询错误码

示例	if(tsdiag_can_request_transfer_exit(udsHandle) == 0x00)
	{
	Console.WriteLine("request transfer data exit success!");
	}
	else
	{
	Console.WriteLine("request transfer data exit failed!");
	}

85. tsdiag_can_write_data_by_identifier

函数名称	int tsdiag_can_write_data_by_identifier(int ADiagModuleIndex, ushort ADataIdentifier, byte* AWriteData, int AWriteDataSizem)
功能介绍	通过标识符写入数据
调用位置	连接硬件之后，需要创建诊断模块后，不再需要该诊断模块
输入参数	ADiagModuleIndex:uds 模块的句柄 ADataIdentifier:要写入的数据的标识符 AWriteData:指向数据缓冲区的数据指针 AWriteDataSizem:要写入的数据编号
返回值	==0：执行成功 其他值：查询错误码

示例	byte writeData[10];
	if(tsdiag_can_write_data_by_identifier(udsHandle, 0xF198, writeData, 10) == 0x00)
	{
	Console.WriteLine("write data by id:0xF198 success!",);
	}
	else
	{
	Console.WriteLine("write data by id:0xF198 failed!",);
	}

86. tsdiag_can_read_data_by_identifier

函数名称	int tsdiag_can_read_data_by_identifier(int ADiagModuleIndex, ushort ADataIdentifier, byte* AReturnArray, ref int AReturnArraySize)
功能介绍	通过标识符读取数据
调用位置	连接硬件之后，需要创建诊断模块后，不再需要该诊断模块
输入参数	ADiagModuleIndex:uds 模块的句柄 ADataIdentifier:要读取的数据的标识符 AReturnArray:指向数据缓冲区的数据指针 AReturnArraySize:从 dut 读取的实际数据编号
返回值	==0：执行成功 其他值：查询错误码

示例	byte readData[20]; //Buffer to save datas read from ECU
	int readDataSize = 20;
	if(tsdiag_can_read_data_by_identifier(udsHandle, 0xF198, readData, ref readDataSize) == 0x00)
	{
	Console.WriteLine("read data by id:DATA_ID success!");
	}
	else
	{
	Console.WriteLine("read data by id:DATA_ID failed!");
	}

87. GetDeviceInfo

函数名称	uint GetDeviceInfo(uint ADeviceIndex, ref string AFManufacturer, ref string AFProduct, ref string AFSerial)
功能介绍	获取设备信息
调用位置	连接硬件之后，需要创建诊断模块后，不再需要该诊断模块
输入参数	ADeviceIndex:设备索引 AFManufacturer:生产者 AFProduct:产品 AFSerial:序列号
返回值	==0：执行成功 其他值：查询错误码
示例	string AFManufacturer; string AFProduct; string AFSerial; GetDeviceInfo(0, ref string AFManufacturer, ref AFProduct, ref AFSerial);

88. tsdiag_can_security_access_send_key_dontnet

函数名称	int tsdiag_can_security_access_send_key_dontnet(int ADiagModuleIndex, int ALevel, uint AKeyValue)
功能介绍	27 key 发送 key
调用位置	连接硬件之后，需要创建诊断模块后，不再需要该诊断模块
输入参数	ADiagModuleIndex:uds 模块的句柄 ALevel:安全级别 AKeyValue:用于保存键值的数据指针 AKeySize:键值大小
返回值	==0：执行成功 其他值：查询错误码

示例	byte keyValue[4]; //Calculated from seed value
	if(tsdiag_can_security_access_send_key_dontnet(udsHandle, 0x02, keyValue, 4) == 0x00)
	{
	Console.WriteLine("unlock the ecu success!");
	}
	else
	{
	Console.WriteLine("unlock the ecu failed");
	}

89. TSCAN_GetTSCANErrorDescription

函数名称	string TSCAN_GetTSCANErrorDescription(uint ACode)
功能介绍	获取 TSCAN 错误描述
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
返回值	返回的 string 类型的错误信息
示例	TSCAN_GetTSCANErrorDescription(23);

90. ReceiveCANMsg

函数名称	bool ReceiveCANMsg(IntPtr ADeviceHandle, out TLIBCAN ACANMsg, int ATimeoutMs, CHANNEL_INDEX AChn = CHANNEL_INDEX.CHN1, READ_TX_RX_DEF AOnlyRx = READ_TX_RX_DEF.ONLY_RX_MESSAGES)
功能介绍	接收 can 报文
调用位置	

输入参数	ADeviceHandle:设备句柄 ACANMsg：can 结构体指针 ATimeoutMs：超时时间 AChn ：通道 AOnlyRx ：仅发送
返回值	==true:接收成功 ==false:接收失败
示例	IntPtr ADeviceHandle = (IntPtr)0; TLIBCAN ACANMsg = new TLIBCAN(); ReceiveCANMsg(ADeviceHandle,out ACANMsg,100,CHANNEL_INDEX.CHN1, READ_TX_RX_DEF.ONLY_RX_MESSAGES);

91. ReceiveCANMsgList

函数名称	int ReceiveCANMsgList(IntPtr ADeviceHandle, ref TLIBCAN[] ACANMsgBuffer, int ACANBufferSize, CHANNEL_INDEX AChn = CHANNEL_INDEX.CHN1, READ_TX_RX_DEF ATxRx = READ_TX_RX_DEF.ONLY_RX_MESSAGES)
功能介绍	接收 can 报文列表
调用位置	
输入参数	ADeviceHandle:设备句柄 ACANMsgBuffer：can 报文缓冲区 ACANBufferSize：缓冲区大小 AChn ：通道 ATxRx ：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
返回值	==true:接收成功 ==false:接收失败
示例	IntPtr ADeviceHandle = (IntPtr)0; TLIBCAN[] ACANMsgBuffer; ReceiveCANMsgList(ADeviceHandle,ref ACANMsgBuffer,ACANBufferSize, CHANNEL_INDEX.CHN1, READ_TX_RX_DEF.ONLY_RX_MESSAGES);

92. ReceiveCANFDMsg

函数名称	bool ReceiveCANFDMsg(IntPtr ADeviceHandle, out TLIBCANFD ACANMsg, int ATimeoutMs, CHANNEL_INDEX AChn = CHANNEL_INDEX.CHN1, READ_TX_RX_DEF ARxTx = READ_TX_RX_DEF.ONLY_RX_MESSAGES)
功能介绍	接收 canfd 报文列表
调用位置	
输入参数	ADeviceHandle:设备句柄 ACANMsg：can 报文 ATimeoutMs：超时时间 AChn ：通道

	ATxRx ： 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
返回值	==true:接收成功 ==false:接收失败
示例	IntPtr ADeviceHandle = (IntPtr)0; TLIBCANfd ACANMsg = new TLIBCANFD(); ReceiveCANFDMsg(ADeviceHandle,out ACANMsg,100, CHANNEL_INDEX.CHN1, READ_TX_RX_DEF.ONLY_RX_MESSAGES);

93. ReceiveCANFDMsgList

函数名称	int ReceiveCANFDMsgList(IntPtr ADeviceHandle, ref TLIBCANFD[] ACANMsgBuffer, int ACANBufferSize, CHANNEL_INDEX AChn = CHANNEL_INDEX.CHN1, READ_TX_RX_DEF ARxTx = READ_TX_RX_DEF.ONLY_RX_MESSAGES)
功能介绍	接收 canfd 报文列表
调用位置	
输入参数	ADeviceHandle:设备句柄 ACANMsgBuffer：can 报文缓冲区 ACANBufferSize：缓冲区大小 AChn ： 通道 ATxRx ： 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
返回值	==true:接收成功 ==false:接收失败
示例	IntPtr ADeviceHandle = (IntPtr)0; TLIBCAN[] ACANMsgBuffer; ReceiveCANFDMsg(ADeviceHandle,ref ACANMsgBuffer,ACANBufferSize, CHANNEL_INDEX.CHN1, READ_TX_RX_DEF.ONLY_RX_MESSAGES);

94. ReceiveLINMsg

函数名称	bool ReceiveLINMsg(IntPtr ADeviceHandle, out TLIBLIN ALINMsg, int ATimeoutMs, CHANNEL_INDEX AChn = CHANNEL_INDEX.CHN1, READ_TX_RX_DEF ATxRx = READ_TX_RX_DEF.ONLY_RX_MESSAGES)
功能介绍	接收 lin 报文
调用位置	
输入参数	ADeviceHandle:设备句柄 ALINMsg：lin 报文 ATimeoutMs：超时时间 AChn ： 通道 ATxRx ： 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

返回值	==true:接收成功 ==false:接收失败
示例	IntPtr ADeviceHandle = (IntPtr)0; ReceiveLINMsg(ADeviceHandle,out ALINMsg, 100,CHANNEL_INDEX.CHN1, READ_TX_RX_DEF.ONLY_RX_MESSAGES)

95. ReceiveLINMsgList

函数名称	int ReceiveLINMsgList(IntPtr ADeviceHandle, ref TLIBLIN[] ALINMsgBuffer, int ALINBufferSize, CHANNEL_INDEX AChn = CHANNEL_INDEX.CHN1, READ_TX_RX_DEF ATxRx = READ_TX_RX_DEF.ONLY_RX_MESSAGES)
功能介绍	接收 lin 报文
调用位置	
输入参数	ADeviceHandle:设备句柄 ALINMsgBuffer：can 报文缓冲区 ALINBufferSize：缓冲区大小 AChn ：通道 ATxRx ：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
返回值	==0：接收成功 其他值：返回错误信息
示例	IntPtr ADeviceHandle = (IntPtr)0; ReceiveLINMsg(ADeviceHandle,ref ALINMsgBuffer, 100,CHANNEL_INDEX.CHN1, READ_TX_RX_DEF.ONLY_RX_MESSAGES)

96. ReceiveFastLINMsg

函数名称	bool ReceiveFastLINMsg(IntPtr ADeviceHandle, out TLIBLIN ALINMsg, int ATimeoutMs, CHANNEL_INDEX AChn = CHANNEL_INDEX.CHN1, READ_TX_RX_DEF AOnlyRx = READ_TX_RX_DEF.ONLY_RX_MESSAGES)
功能介绍	接收 fastlin 报文
调用位置	
输入参数	ADeviceHandle:设备句柄 ALINMsg：lin 报文 ATimeoutMs：超时时间 AChn ：通道 ATxRx ：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
返回值	==true:接收成功 ==false:接收失败
示例	IntPtr ADeviceHandle = (IntPtr)0; ReceiveFastLINMsg(ADeviceHandle, out ALINMsg, 100,CHANNEL_INDEX.CHN1, READ_TX_RX_DEF.ONLY_RX_MESSAGES)

97. ReceiveFastLINMsgList

函数名称	int ReceiveFastLINMsgList(IntPtr ADeviceHandle, ref TLIBLIN[] ALINMsgBuffer, int ALINBufferSize, CHANNEL_INDEX AChn = CHANNEL_INDEX.CHN1, READ_TX_RX_DEF ATxRx = READ_TX_RX_DEF.ONLY_RX_MESSAGES)
功能介绍	接收 fastlin 报文
调用位置	
输入参数	ADeviceHandle:设备句柄 ALINMsgBuffer：can 报文缓冲区 ALINBufferSize：缓冲区大小 AChn ：通道 ATxRx ：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
返回值	==0：接收成功 其他值：返回错误信息
示例	IntPtr ADeviceHandle = (IntPtr)0; ReceiveFastLINMsg(ADeviceHandle,ref ALINMsgBuffer, 100,CHANNEL_INDEX.CHN1, READ_TX_RX_DEF.ONLY_RX_MESSAGES)；

98. tscan_ configure_ canfd_regs

函数名称	UInt32 tscan_configure_canfd_regs(IntPtr ADevicelandle, CHANNEL_ INDEX AChnIdx, double AArbRateKbps, int ASEGI, int ASEG2, int APrescaler, int AS,J2, double ADataRateKbps, int ADataSEG1, int ADataSEG2, int ADataPrescaler, int ADataSJ2, TLIBCANFDControllerType AControllerType, TLIBCANFDControllerMode AControllerMode, boolA120OhmConnected)
功能介绍	配置 CANFD 采样点以及波特率
调用位置	连接硬件之后，需要创建诊断模块之后
输入参数	ADevicellandle:硬件连接生成的句柄 AChnIdx: can 通道索引 AArbRateKbps:仲裁段波特率 ASEG1:仲裁段相位缓冲段 1 ASEG1:仲裁段相位缓冲段 2 APrescaler:仲裁段分频系数 ASJ2:仲裁段 BTL 数量 ADataRateKbps: 数据段波特率 ADataSEG1: 数据段相位缓冲段 1 ADataSEGI: 数据段相位缓冲段 2 ADataPrescaler: 数据段分频系数 ADataSJ2:数据段 BTL 数量 AControllerTyper: 控制器类型 0: can 1:isocanfd 2:non-isocanfd AControllerMode:控制器模式 0:正常模式 1:关闭应答 2;限制模式

	120OhmConnected: 是否激活 120Ω终端电阻
返回值	==0：注册完成 其他值：注册失败
示例	tscan_configure_canfd_regs (Handle, 0, 500, 63, 16, 1, 80, 2000, 15, 4, 1,20,1, 0, true)

99. tscan_ configure can_ regs

函数名称	UInt32 tscan_configure_can_regs (IntPtr ADev iceHandle, CHANNEL_ INDEX AChnIdx, double ABaudrateKbps,int ASEG1, int ASEG2, int APrescaler, int ASJ2, bool AOnlyListen, bool A120)
功能介绍	配置 CAN 采样点以及波特率
调用位置	连接硬件之后，
输入参数	ADev iceHandle:硬件连接生成的句柄 AChnIdx: can 通道 ABaudrateKbps:波特率 ASEG1:相位缓冲段 1 ASEG1:相位缓冲段 2 APrescaler:分频系数 ASJ2: BTL 数量. AOnlyListen:是否为只听模式 A120:是否激活 120 Ω终端电阻
返回值	==0:注册成功 其他值:注册失败
示例	tscan_configure_can_regs (Handle, 0, 500, 63, 16, 1, 80, false, true);