libTSCANAPI 编程指导

文档修订历史：

文件版本	日期	更新内容	备注
V1.00	2023.7.7	创建文档	

1. 目录

1. 什么情况下需要此文档？

用户基于 Python 语言对上海同星智能科技有限公司的 TS 系列工具（CAN/CANFD/LIN/FlexRay/ETH）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。

注：libTSCANAPI windows /linux 平台通用，即一套代码可以在 windows 以及 linux 中实现控制同星硬件代码不需要做一行更改（仅 libTSCANAPI 代码相关部分）

2. 添加库文件

Python 语言开发者只需要通过 pip 进行库的安装，安装如下：

Pip install libTSCANAPI

3. 总线数据类型定义

备注：

可能存在部分 Python 开发者不理解下列定义，这是使用 python 中 ctypes 库，将 python 与 c 语言进行联合编程，而定义出来的。对于 Python 而言，TLIBCAN 就是一个类，里面的成员可以直接使用，而对于 C 而，TLIBCAN 就是结构体（TLIBLIN TLIBCANFD TLIBFlexray 同理），在此特意声明。

1. TLIBCAN：CAN 总线数据类型

```
class TLIBCAN(Structure):
    '''
    CAN 报文结构体
    关联函数：
    tsapp_transmit_can_async  发送报文
    tsfifo_receive_can_msgs   接收报文
    '''
    _pack_ = 1
    _fields_ = [("FIdxChn", c_uint8),
                ("FProperties", c_uint8),
                ("FDLC", c_uint8),
                ("FReserved", c_uint8),
                ("FIdentifier", c_int32),
                ("FTimeUs", c_int64),
                ("FData", c_uint8 * 8),
                ]
```

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：

```
ACAN = TLIBCAN(
                    FIdxChn=0,
                    FDLC=8,
                    FIdentifier=0x123,
                    FProperties=1,
                    FData=[1,2,3,4,5,6,7,8]
                    )
```

2. TLIBCANFD: CANFD 总线数据类型

```
class TLIBCANFD(Structure):
    '''
    CANFD 报文结构体
    关联函数：
    tsapp_transmit_canfd_async 发送报文
    tsfifo_receive_canfd_msgs  接收报文
    '''
    _pack_ = 1
    _fields_ = [("FIdxChn", c_uint8),
                ("FProperties", c_uint8),
                ("FDLC", c_uint8),
```

```
("FFDProperties", c_uint8),
("FIdentifier", c_int32),
("FTimeUs", c_uint64),
("FData", c_uint8 * 64),
]
```

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：

```
ACANFD = TLIBCANFD(
                    FIdxChn=0,
                    FDLC=8,
                    FIdentifier=0x123,
                    FProperties=1,
                    FFDProperties = 1,
                    FData=[1,2,3,4,5,6,7,8]
                    )
```

注：实际 TLIBCANFD 向下兼容 CAN，使用 TLIBCANFD 定义 CAN 数据类型，只需要将 FFDProperties 置为 0 即可，当然调用的发送函数仍然是 CANFD 系列函数，最终发出的报文类型依据 FFDProperties 而定。

3. TLIBLIN：LIN 总线数据类型

```
class TLIBLIN(Structure):
    _pack_ = 1
    _fields_ = [("FIdxChn", c_uint8),
                ("FErrStatus", c_uint8),
                ("FProperties", c_uint8),
                ("FDLC", c_uint8),
                ("FIdentifier", c_uint8),
                ("FChecksum", c_uint8),
                ("FStatus", c_uint8),
                ("FTimeUs", c_int64),
                ("FData", c_uint8 * 8),
                ]
```

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：

```
ALIN= TLIBLIN(
                    FIdxChn=0,
                    FDLC=8,
                    FIdentifier=0x11,
                    FProperties=1,
                    FData=[1,2,3,4,5,6,7,8]
                    )
```

4. TLIBFlexray：Flexray 总线数据类型

```
class TLIBFlexray(Structure):
    _pack_ = 1
    _fields_ = [("FIdxChn", c_uint8),
                ("FChannelMask", c_uint8),
                ("FDir", c_uint8),
                ("FPayloadLength", c_uint8),
                ("FActualPayloadLength", c_uint8),
                ("FCycleNumber", c_uint8),
                ("FCCType", c_uint8),
                ("FReserved0", c_uint8),
                ("FHeaderCRCA", c_uint16),
                ("FHeaderCRCB", c_uint16),
                ("FFrameStateInfo", c_uint16),
                ("FSlotId", c_uint16),
                ("FFrameFlags", c_uint32),
                ("FFrameCRC", c_uint32),
                ("FReserved1", c_uint64),
                ("FReserved2", c_uint64),
                ("FTimeUs", c_uint64),
                ("FData", c_uint8 * 254),
                ]
```

成员：

FIdxChn:帧通道，注意 CHANNEL_INDEX. CHN1 = 0,实际上是从 0 开始计算的。
FChannelMask:Flexray A B 通道
FDir：方向，TX or RX

FPayloadLength :帧长度
FActualPayloadLength：真实帧长度
FCycleNumber：帧发送 接收 Cycle
FCCType：报文类型
FReserved0：预留
FHeaderCRCA：帧 CRC
FHeaderCRCB：帧 CRC
FFrameStateInfo：帧状态
FSlotId：帧 id
FFrameFlags：帧标志
FFrameCRC ：帧 CRC
FReserved1：预留
FReserved2：预留
FTimeUs：帧时间
FData：帧数据

其中，属性字节定义如下：

【1】 FChannelMask：帧 flexray 通道，该字段值定义如下：

Value	意义
0	预留
1	A
2	B
3	AB

【2】 FDir：帧方向，该字段值定义如下：

Bit	意义
0	0:Rx 1:Tx

【3】 FCCType：帧类型，该字段值定义如下：

Value	意义
0	Architecture independent（正常帧）
1	Invalid CC type
2	Cyclone
3	BUSDOCTOR
4	Cyclone II
5	Vector VN interface
6	VN-Sync-Pulse(only in Status Event, for debugging purposes only)

【4】 FFrameFlags：帧标志，该字段值定义如下：

Bit	意义（bit 置 1）
0	Null frame
1	Data segment contains valid data
2	Sync bit
3	Startup flag
4	Payload preamble bit
5	Reserved bit

6	Error flag(error frame or invalid frame)
7	Reserved
15	Async.monitoring has generated this event
16	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	Raw frame(only valid if PDUs are used in the configuration).A raw frame may contain PDUs in its payload
20	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

调用示例：

```
AFlexray= TLIBFlexray(
                    FIdxChn=0,
                    FSlotId=8,
                    FChannelMask = 1,
                    FCycleNumber=1,
                    FActualPayloadLength=8,
                    FData=[1,2,3,4,5,6,7,8]
                    )
```

4. 报文发送

1. CAN 报文发送

```
ACAN = TLIBCAN(
                    FIdxChn=0,
                    FDLC=8,
                    FIdentifier=0x123,
                    FProperties=1,
                    FData=[1,2,3,4,5,6,7,8]
                    )
```

单帧异步发送：

```
tsapp_transmit_can_async(HwHandle,ACAN) #HwHandle 为硬件句柄，后续说明
```

单帧同步发送：

```
# 100 表示超时参数
tsapp_transmit_can_sync(HwHandle,ACAN,100) #HwHandle 为硬件句柄，后续说明
```

周期发送：

```
# 100ms 周期发送 ACAN 报文
tsapp_add_cyclie_msg_can(HwHandle,ACAN,100) #HwHandle 为硬件句柄，后续说明
```

删除周期发送：

```
# 删除周期发送 ACAN 报文
tsapp_delete_cyclie_msg_can(HwHandle,ACAN) #HwHandle 为硬件句柄，后续说明
```

2. CANFD 报文发送

```
ACANFD = TLIBCANFD(
                    FIdxChn=0,
                    FDLC=8,
                    FIdentifier=0x123,
                    FProperties=1,
                    FFDProperties=1,
                    FData=[1,2,3,4,5,6,7,8]
)
```

单帧异步发送：

```
tsapp_transmit_canfd_async(HwHandle,ACANFD) #HwHandle 为硬件句柄，后续说明
```

单帧同步发送：

```
# 100 表示超时参数
tsapp_transmit_canfd_sync(HwHandle,ACANFD,100) #HwHandle 为硬件句柄，后续说明
```

周期发送：

```
# 100ms 周期发送 ACAN 报文
tsapp_add_cyclie_msg_canfd(HwHandle,ACANFD,100) #HwHandle 为硬件句柄，后续说明
```

删除周期发送：

```
# 删除周期发送 ACAN 报文
tsapp_delete_cyclie_msg_canfd(HwHandle,ACANFD) #HwHandle 为硬件句柄，后续说明
```

3. LIN 报文发送

```
ALIN = TLIBLIN(
                    FIdxChn=0,
                    FDLC=8,
                    FIdentifier=0x11,
                    FProperties=1,
                    FData=[1,2,3,4,5,6,7,8]
                    )
#下列 2 个函数，具体参数说明在第 8 章有详细介绍
  tsapp_configure_baudrate_lin(HwHandle,0,19.2,1) #设置 lin 波特率
  tslin_set_node_functiontype(HwHandle,0,T_LIN_NODE_FUNCTION.T_MASTER_NODE)
```

单帧异步发送：

```
tsapp_transmit_lin_async(HwHandle,ALIN) #HwHandle 为硬件句柄，后续说明
```

单帧同步发送：

```
# 100 表示超时参数
tsapp_transmit_lin_sync(HwHandle,ALIN,100) #HwHandle 为硬件句柄，后续说明
```

4. Flexray 报文发送

```
AFlexray= TLIBFlexray(
                    FIdxChn=0,
                    FSlotId=8,
                    FChannelMask = 1,
                    FCycleNumber=1,
```

```
FActualPayloadLength=8,
FData=[1,2,3,4,5,6,7,8]
)
```

单帧异步发送：

tsapp_transmit_flexray_async(HwHandle,AFlexray) #HwHandle 为硬件句柄，后续说明

单帧同步发送：

```
# 100 表示超时参数
tsapp_transmit_flexray_sync(HwHandle,AFlexray,100) #HwHandle 为硬件句柄，后续说明
```

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/LIN/Flexray 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。libTSCANAPI 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（Python 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：

```
if 'windows' in _os.lower():
    OnTx_RxFUNC_CAN = WINFUNCTYPE(None,PCAN)
    OnTx_RxFUNC_Flexray = WINFUNCTYPE(None, PFlexray)
    OnTx_RxFUNC_LIN = WINFUNCTYPE(None,PLIN)
    OnTx_RxFUNC_CANFD = WINFUNCTYPE(None,PCANFD)
    OnTx_RxFUNC_CAN_WHandle = WINFUNCTYPE(None, ps64,PCAN)
    OnTx_RxFUNC_Flexray_WHandle = WINFUNCTYPE(None,ps64, PFlexray)
    OnTx_RxFUNC_LIN_WHandle = WINFUNCTYPE(None, ps64,PLIN)
    OnTx_RxFUNC_CANFD_WHandle = WINFUNCTYPE(None, ps64,PCANFD)
    On_Connect_FUNC = WINFUNCTYPE(None,ps64)
    On_disConnect_FUNC = WINFUNCTYPE(None, ps64)
else:
    OnTx_RxFUNC_CAN = CFUNCTYPE(None, PCAN)
    OnTx_RxFUNC_Flexray = CFUNCTYPE(None, PFlexray)
    OnTx_RxFUNC_LIN = CFUNCTYPE(None, PLIN)
    OnTx_RxFUNC_CANFD = CFUNCTYPE(None, PCANFD)
    OnTx_RxFUNC_CAN_WHandle = CFUNCTYPE(None, ps64,PCAN)
    OnTx_RxFUNC_Flexray_WHandle = CFUNCTYPE(None,ps64, PFlexray)
    OnTx_RxFUNC_LIN_WHandle = CFUNCTYPE(None, ps64,PLIN)
    OnTx_RxFUNC_CANFD_WHandle = CFUNCTYPE(None, ps64,PCANFD)
    On_Connect_FUNC = CFUNCTYPE(None, ps64)
    On_disConnect_FUNC = CFUNCTYPE(None, ps64)
```

然后把代理对象调用回调函数注册到驱动中，如下所示：

```
ret = taspp_register_event_canfd(HwHandle, ONCANFDEvent);
```

。

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：
    def On_CAN_Event(obj,msg):
        if  msg.contents.is_tx():
            pass #表明该报文为设备发出报文
        else:
            pass#表明该报文为设备接收报文
        if  msg.contents.is_ext():
            pass #表明为扩展帧
    ONCANEvent = OnTx_RxFUNC_CAN_WHandle(On_CAN_Event)
    ret = taspp_register_event_can_whandle(HwHandle, ONCANEvent )

CANFD 回调：
    def On_CANFD_Event(obj,msg):
        if  msg.contents.is_tx():
            pass #表明该报文为设备发出报文
        else:
```

```
                pass#表明该报文为设备接收报文
            if  msg.contents.is_ext():
                pass #表明为扩展帧
        ONCANFDEvent = OnTx_RxFUNC_CANFD_WHandle(On_CANFD_Event)
        ret = taspp_register_event_canfd_whandle(HwHandle, ONCANFDEvent )
```

LIN 回调：

```
        def On_LIN_Event(obj,msg):
            if  msg.contents.is_tx():
                pass #表明该报文为设备发出报文
            else:
                pass#表明该报文为设备接收报文
        ONLINEvent = OnTx_RxFUNC_LIN_WHandle(On_LIN_Event)
        ret = taspp_register_event_lin_whandle(HwHandle, ONLINEvent )
```

Flexray 回调：

```
        def On_Flexray_Event(obj,msg):
            if  msg.contents.is_tx():
                pass #表明该报文为设备发出报文
            else:
                pass#表明该报文为设备接收报文
        ONFlexrayEvent = OnTx_RxFUNC_Flexray_WHandle(On_Flexray_Event)
        ret = taspp_register_event_flexray_whandle(HwHandle, 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。

CAN 报文获取

获取 Rx 报文：

```
TCANBuffer  =  (TLIBCAN*100)()
BufferSize  =  s32(100)          #buffersize 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 buffersize 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(self.HwHandle,TCANBuffer,BufferSize,chnidx,READ_TXRX_DEF.ONLY_RX_MESSAGES)

获取 Rx Tx 报文：

TCANBuffer = (TLIBCAN*100)()

BufferSize = s32(100) #buffersize 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 buffersize 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(self.HwHandle,TCANBuffer,BufferSize,chnidx,READ_TXRX_DEF.TX_RX_MESSAGES)

获取 fifo 报文数量：

```
#读取通道 0 fifo 的报文数量
ACount = s32(0)
tsfifo_read_can_buffer_frame_count(self.HwHandle,0,ACount)
Print(ACount )
```

获取 fifo Tx 报文数量：

```
#读取通道 0 fifo Tx 的报文数量
ACount = s32(0)
tsfifo_read_can_tx_buffer_frame_count(self.HwHandle,0,ACount)
Print(ACount )
```

获取 fifo Rx 报文数量：

```
#读取通道 0 fifo Rx 的报文数量
ACount = s32(0)
tsfifo_read_can_rx_buffer_frame_count(self.HwHandle,0,ACount)
Print(ACount )
```

清空 fifo 报文：

```
#读取通道 0 fifo Rx 的报文数量
tsfifo_clear_can_receive_buffers(self.HwHandle,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：

TCANFDBuffer = (TLIBCAND*100)()
BufferSize = s32(100) #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 buffersize 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(self.HwHandle,TCANFDBuffer,BufferSize,chnidx,READ_TXRX_DEF.ONLY_RX_MESSAGES)

获取 Rx Tx 报文：

TCANDBuffer = (TLIBCANFD*100)()
BufferSize = s32(100) #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 buffersize 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(self.HwHandle,TCANFDBuffer,BufferSize,chnidx,READ_TXRX_DEF.TX_RX_MESSAGES)

获取 fifo 报文数量：

#读取通道 0 fifo 的报文数量
ACount = s32(0)
tsfifo_read_canfd_buffer_frame_count(self.HwHandle,0,ACount)
Print(ACount)

获取 fifo Tx 报文数量：

#读取通道 0 fifo Tx 的报文数量
ACount = s32(0)
tsfifo_read_canfd_tx_buffer_frame_count(self.HwHandle,0,ACount)
Print(ACount)

获取 fifo Rx 报文数量：

#读取通道 0 fifo Rx 的报文数量
ACount = s32(0)
tsfifo_read_canfd_rx_buffer_frame_count(self.HwHandle,0,ACount)
Print(ACount)

清空 fifo 报文：

#读取通道 0 fifo Rx 的报文数量

tsfifo_clear_canfd_receive_buffers(self.HwHandle,0)

LIN 报文获取

获取 Rx 报文：

TLINBuffer = (TLIBLIN*100)()

BufferSize = s32(100) #buffersize 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 buffersize 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(self.HwHandle,TLINBuffer,BufferSize,chnidx,READ_TXRX_DEF.ONLY_RX_MESSAGES)

获取 Rx Tx 报文：

TLINBuffer = (TLIBLIN*100)()

BufferSize = s32(100) #buffersize 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 buffersize 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(self.HwHandle,TLINBuffer,BufferSize,chnidx,READ_TXRX_DEF.TX_RX_MESSAGES)

获取 fifo 报文数量：

#读取通道 0 fifo 的报文数量

ACount = s32(0)

tsfifo_read_lin_buffer_frame_count(self.HwHandle,0,ACount)

Print(ACount)

获取 fifo Tx 报文数量：

#读取通道 0 fifo Tx 的报文数量

ACount = s32(0)

tsfifo_read_lin_tx_buffer_frame_count(self.HwHandle,0,ACount)

Print(ACount)

获取 fifo Rx 报文数量：

#读取通道 0 fifo Rx 的报文数量

ACount = s32(0)

tsfifo_read_lin_rx_buffer_frame_count(self.HwHandle,0,ACount)

Print(ACount)

清空 fifo 报文：

```
#读取通道 0 fifo Rx 的报文数量
tsfifo_clear_lin_receive_buffers(self.HwHandle,0)
```

Flexray 报文获取

获取 Rx 报文：

```
TFlexrayBuffer = (TLIBFlexray*100)()
BufferSize = s32(100)          #buffersize 的大小为 TFlexrayBuffer 的长度，可以小，当一定不能比 TCANBuffer 大
#注：每次传入 tsfifo_receive_can_msgs 的 buffersize 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(self.HwHandle,TFlexrayBuffer,BufferSize,chnidx,READ_TXRX_DEF.ONLY_RX_MESSAGES)
```

获取 Rx Tx 报文：

```
TFlexrayBuffer = (TLIBFlexray*100)()
BufferSize = s32(100)          #buffersize 的大小为 TFlexrayBuffer 的长度，可以小，当一定不能比 TCANBuffer 大
#注：每次传入 tsfifo_receive_can_msgs 的 buffersize 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(self.HwHandle,TFlexrayBuffer,BufferSize,chnidx,READ_TXRX_DEF.TX_RX_MESSAGES)
```

获取 fifo 报文数量：

```
#读取通道 0 fifo 的报文数量
ACount = s32(0)
tsfifo_read_flexray_buffer_frame_count(self.HwHandle,0,ACount)
Print(ACount )
```

获取 fifo Tx 报文数量：

```
#读取通道 0 fifo Tx 的报文数量
ACount = s32(0)
tsfifo_read_flexray_tx_buffer_frame_count(self.HwHandle,0,ACount)
Print(ACount )
```

获取 fifo Rx 报文数量：

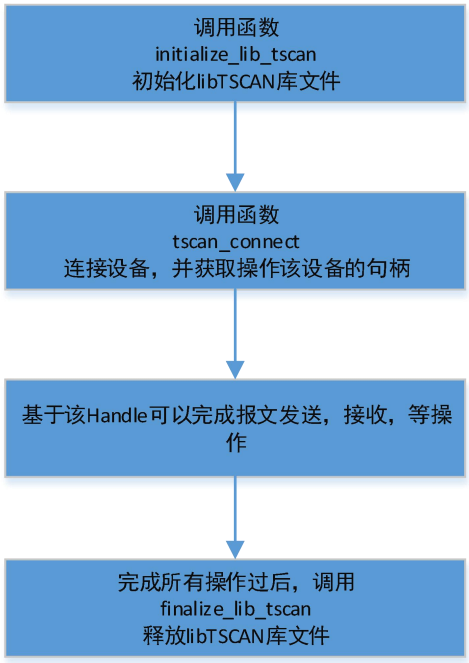
```
#读取通道 0 fifo Rx 的报文数量
```

```
ACount = s32(0)
tsfifo_read_flexray_rx_buffer_frame_count(self.HwHandle,0,ACount)
Print(ACount )
```

清空 fifo 报文：

```
#读取通道 0 fifo Rx 的报文数量
tsfifo_clear_flexray_receive_buffers(self.HwHandle,0)
```

6. libTSCANAPI 调用流程

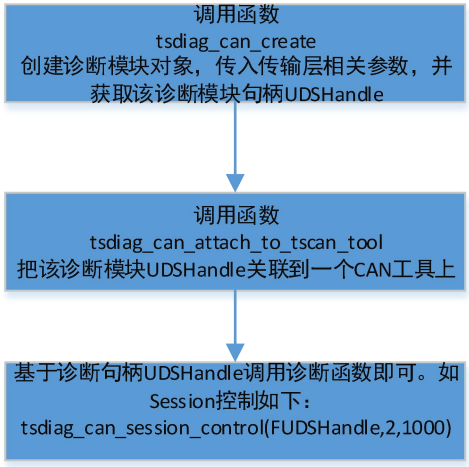


注：图中的 tscan_connect,在 libTSCANAPI 中为 tsapp_connect()

```
HwHandle = siez_t(0)
tsapp_connect(b'',HwHandle )  #该 HwHandle 便是后续一切操作硬件的句柄
```

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：



在 libTSCANAPI 中，创建诊断模块只需要直接调用 tsdiag_can_create 函数即可，诊断模块中的其他函数将于第 8 节中进行介绍说明。

8. 接口函数介绍

1. initialize_lib_tscan

函数名称	**initialize_lib_tscan(AEnableFIFO:c_bool,AEnableErrorRecv:c_bool,UseConnecttime:c_bool):->None**
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; AEnableErrorRecv: 是否开启错误帧接收，推荐设置为 True; UseConnecttime:是否使用硬件连接 USB 时间，推荐设置为 Flase;
返回值	无
示例	**initialize_lib_tscan(True,True,False)**

2. finalize_lib_tscan

函数名称	**finalize_lib_tscan()->None**
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. tsapp_connect

函数名称	**tsapp_connect(ASerial:c_char_p,WhHandle:c_sizt_t):->int**
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	**ASerial:设备序列号，指定连接设备是需要填写，当只有一个设备可为 None** **WhHandle:返回设备句柄，后续所有操作的唯一句柄**
返回值	==0:连接成功 ==5:设备已经连接 Other:错误码
示例	HwHandle = c_size_t(0) tsapp_connect(b'',HwHandle)

4. tsapp_disconnect_by_handle

函数名称	**tsapp_disconnect_by_handle(WhHandle:c_size_t):->int**
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	设备句柄
返回值	==0:断开设备成功 其他值：错误码
示例	tsapp_connect_by_handle(HwHandle)

5. tscan_scan_devices

函数名称	**tscan_scan_devices(ADeviceCount:c_int32)**
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,执行成功后返回在线设备数量
返回值	==0：获取成功 其他值：注册失败
示例	ACount = c_int32(0) tscan_scan_devices(ACount) print("online devices count = ",ACount)

6. tscan_get_device_info

函数名称	**tscan_get_device_info(** **ADeviceIndex:c_int32,** **AFManufacturer:c_char_p,** **AFProduct:c_char_p,** **AFSerial:c_char_p):->int**

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：设备串行号
返回值	==0：查询成功 其他值：错误码
示例	ACount = c_int32(0) tscan_scan_devices(ACount) print("online devices count = ",ACount) for i in range(ACount.value): AFManufacturer = c_char_p() AFProduct= c_char_p() AFSerial= c_char_p() tscan_get_device_info(i,AFManufacturer ,AFProduct,AFSerial) print(AFManufacturer ,AFProduct,AFSerial)

7. tsapp_config_can_by_baudrate

函数名称	**tsapp_config_can_by_baudrate(HwHandle:c_size_t, AChnIdx:c_int32, ARateKbp:c_double, A120OhmConnected:c_int32):->int**
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	HwHandle：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	tsapp_config_can_by_baudrate(HwHandle,0,500,1)

8. tsapp_config_canfd_by_baudrate

函数名称	**tsapp_config_canfd_by_baudrate(HwHandle:c_size_t, AChnIdx:c_int32, ARateKbp:c_double,ADataKbp:c_double,AControllerType:c_int32,AControllerMode:c_int32, A120OhmConnected:c_int32):->int**
功能介绍	设置 CANFD 通道的波特率参数
调用位置	在 CANFD 工具连接成功后，调用此函数设置 CANFD 工具的波特率
输入参数	HwHandle：设备句柄； AChnIdx:CAN 通道编号 ARateKbp:波特率值，如：1000kbps,500kbps,250kbps,125kbps ADataKbp:数据段波特率，如 2000kbps,4000kbps

	AControllerType:控制器类型 0：can 1:isocanfd 2:nonisocanfd AControllerMode:控制器模式 0：正常模式 1：关闭 ACK 2:限制模式 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	tsapp_config_canfd_by_baudrate(HwHandle,0,500,2000,1,0,1)

9. tsapp_configure_baudrate_lin

函数名称	**tsapp_configure_baudrate_lin(HwHandle:c_size_t,AChnIdx:c_int32,AKbp:c_double,ALINProtocol:c_int32):->int**
功能介绍	设置 LIN 通道的波特率参数
调用位置	在 LIN 工具连接成功后，调用此函数设置 LIN 工具的波特率
输入参数	HwHandle：设备句柄； AChnIdx:LIN 通道编号 AKbp:波特率值，如：10.417,19.2 ALINProtocol: 0:1.3 1:2.0 2:2.1
返回值	==0：函数执行成功 其他值：执行失败
示例	tsapp_configure_baudrate_lin(HwHandle,0,19.2,1)

10. tslin_set_node_functiontype

函数名称	**tslin_set_node_functiontype(HwHandle:c_size_t,AChnIdx:c_int32,linMode:c_uint8):->int**
功能介绍	设置 LIN 通道的主从节点模式
调用位置	在 LIN 工具连接成功后，调用此函数设置 LIN 工具的主从节点模式
输入参数	HwHandle：设备句柄； AChnIdx:LIN 通道编号 linMode:LIN 主从节点模式
返回值	==0：函数执行成功 其他值：执行失败
示例	tslin_set_node_functiontype(HwHandle,0,T_LIN_NODE_FUNCTION.T_MASTER_NODE)

11. tscan_get_error_description

函数名称	**tscan_get_error_description(ACode:c_int32, ADescription:c_char_p):->int**
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 ADescription: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	HwHandle = c_size_t(0) Ret = tsapp_connect(b'',HwHandle) result= c_char_p() tscan_get_error_description(ret,result) print(result)

12. tsapp_register_event_can_whandle

函数名称	**tsapp_register_event_can_whandle(HwHandle:c_size_t, ACallback:OnTx_RxFUNC_CAN_WHandle):->int**
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	HwHandle：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	def On_CAN_Event(obj,msg):
	if msg.contents.is_tx():
	pass #表明该报文为设备发出报文
	else:
	pass#表明该报文为设备接收报文
	if msg.contents.is_ext():
	pass #表明为扩展帧
	ONCANEvent= OnTx_RxFUNC_CAN_WHandle(On_CAN_Event)
	ret = taspp_register_event_can_whandle(HwHandle, ONCANEvent)

13. tsapp_unregister_event_can

函数名称	**tsapp_unregister_event_can_whandle(HwHandle:c_size_t, ACallback:OnTx_RxFUNC_CAN_WHandle):->int**
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	HwHandle：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	def On_CAN_Event(obj,msg): if msg.contents.is_tx():

	pass #表明该报文为设备发出报文
	else:
	pass#表明该报文为设备接收报文
	if msg.contents.is_ext():
	pass #表明为扩展帧
	ONCANEvent= OnTx_RxFUNC_CAN_WHandle(On_CAN_Event)
	ret = taspp_register_event_can_whandle(HwHandle, ONCANEvent)
	ret = taspp_unregister_event_can_whandle(HwHandle, ONCANEvent)

14. tsapp_transmit_can_sync

函数名称	**tsapp_transmit_can_sync(HwHandle:c_size_t, ACAN:TLIBCAN,ATimeOutMS:c_uint32):->int**
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	HwHandle：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	ACAN = TLIBCAN(FIdxChn=0, FDLC=8, FIdentifier=0x123, FProperties=1, FData=[1,2,3,4,5,6,7,8]) tsapp_transmit_can_sync(HwHandle,ACAN,100)

15. tsapp_transmit_can_async

函数名称	**tsapp_transmit_can_async(HwHandle:c_size_t, ACAN:TLIBCAN):->int**
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	HwHandle：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	ACAN = TLIBCAN(FIdxChn=0, FDLC=8, FIdentifier=0x123,

	FProperties=1, FData=[1,2,3,4,5,6,7,8]) tsapp_transmit_can_async(HwHandle,ACAN)

16. tsapp_add_cyclic_msg_can

函数名称	**tsapp_add_cyclie_msg_can(HwHandle:c_size_t, ACAN:TLIBCAN, APeriodMS:c_float)**
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	HwHandle：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	ACAN = TLIBCAN(FIdxChn=0, FDLC=8, FIdentifier=0x123, FProperties=1, FData=[1,2,3,4,5,6,7,8]) tsapp_add_cyclie_msg_can(HwHandle,ACAN,100)

17. tsapp_delete_cyclic_msg_can

函数名称	**tsapp_delete_cyclie_msg_can(HwHandle:c_size_t, ACAN:TLIBCAN)**
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	HwHandle：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	ACAN = TLIBCAN(FIdxChn=0, FDLC=8, FIdentifier=0x123, FProperties=1, FData=[1,2,3,4,5,6,7,8])

	tsapp_delete_cyclie_msg_can(HwHandle,ACAN)

18. tsapp_register_event_canfd_whandle

函数名称	**tsapp_register_event_canfd_whandle(HwHandle:c_size_t, ACallback:OnTx_RxFUNC_CANFD_WHandle):->int**
功能介绍	注册 CANFD 数据包接收回调函数
调用位置	在 CANFD 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	HwHandle：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	def On_CAN_Event(obj,msg):
	if msg.contents.is_tx():
	pass #表明该报文为设备发出报文
	else:
	pass#表明该报文为设备接收报文
	if msg.contents.is_ext():
	pass #表明为扩展帧
	ONCANFDEvent=
	OnTx_RxFUNC_CANFD_WHandle(On_CAN_Event)
	ret=taspp_register_event_canfd_whandle(HwHandle,
	ONCANFDEvent)

19. tsapp_unregister_event_canfd_whandle

函数名称	**tsapp_unregister_event_canfd_whandle(HwHandle:c_size_t, ACallback:OnTx_RxFUNC_CANFD_WHandle):->int**
功能介绍	反注册 CANFD 数据接收函数
调用位置	在不需要接收 CANFD 报文的时候，调用此函数
输入参数	HwHandle：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	def On_CAN_Event(obj,msg):
	if msg.contents.is_tx():
	pass #表明该报文为设备发出报文
	else:
	pass#表明该报文为设备接收报文
	if msg.contents.is_ext():

	pass #表明为扩展帧 ONCANFDEvent= OnTx_RxFUNC_CANFD_WHandle(On_CAN_Event) ret=taspp_register_event_canfd_whandle(HwHandle, ONCANFDEvent) ret=taspp_unregister_event_canfd_whandle(HwHandle, ONCANFDEvent)

20. tsapp_transmit_canfd_sync

函数名称	**tsapp_transmit_canfd_sync(HwHandle:c_size_t, ACANFD:TLIBCANFD,ATimeOutMS:c_uint32):->int**
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	HwHandle：设备句柄 ACANFD：CANFD 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	ACANFD = TLIBCANFD(FIdxChn=0, FDLC=8, FIdentifier=0x123, FProperties=1, FFDProperties=1, FData=[1,2,3,4,5,6,7,8]) tsapp_transmit_canfd_sync(HwHandle,ACANFD,100)

21. tsapp_transmit_canfd_async

函数名称	**tsapp_transmit_canfd_async(HwHandle:c_size_t, ACANFD:TLIBCANFD):->int**
功能介绍	以异步的方式发送 CAND 报文
调用位置	发送 CANFD 报文
输入参数	HwHandle：设备句柄 ACANFD：CANFD 数据包
返回值	==0：注册成功 其他值：注册失败
示例	ACANFD = TLIBCANFD(FIdxChn=0, FDLC=8,

	FIdentifier=0x123, FProperties=1, FFDProperties=1, FData=[1,2,3,4,5,6,7,8]) tsapp_transmit_canfd_async(HwHandle,ACANFD)

22. tsapp_add_cyclic_msg_canfd

函数名称	**tsapp_add_cyclie_msg_canfd(HwHandle:c_size_t, ACANFD:TLIBCANFD, APeriodMS:c_float)**
功能介绍	周期性发送 CANFD 报文
调用位置	在需要周期性发送 CANFD 报文的场合
输入参数	HwHandle：设备句柄 ACANFD：CANFD 数据包 APeriodMS：周期值
返回值	==0：注册成功 其他值：注册失败
示例	ACANFD = TLIBCANFD(FIdxChn=0, FDLC=8, FIdentifier=0x123, FProperties=1, FFDProperties=1, FData=[1,2,3,4,5,6,7,8]) tsapp_add_cyclie_msg_canfd(HwHandle,ACANFD,100)

23. tsapp_delete_cyclic_msg_canfd

函数名称	**tsapp_delete_cyclie_msg_canfd(HwHandle:c_size_t, ACANFD:TLIBCANFD)**
功能介绍	停止周期性发送 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	HwHandle：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	ACANFD = TLIBCANFD(FIdxChn=0,

	FDLC=8, FIdentifier=0x123, FProperties=1, FFDProperties=1, FData=[1,2,3,4,5,6,7,8]) tsapp_delete_cyclie_msg_canfd(HwHandle,ACANFD)

24. tsapp_register_event_lin_whandle

函数名称	**tsapp_register_event_lin_whandle(HwHandle:c_size_t, ACallback:OnTx_RxFUNC_LIN_WHandle):->int**
功能介绍	注册 LIN 数据包接收回调函数
调用位置	在 LIN 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	HwHandle：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	def On_LIN_Event(obj,msg):
	if msg.contents.is_tx():
	pass #表明该报文为设备发出报文
	else:
	pass#表明该报文为设备接收报文
	ONLINEvent = OnTx_RxFUNC_LIN_WHandle(On_LIN_Event)
	ret = taspp_register_event_lin_whandle(HwHandle, ONLINEvent)

25. tsapp_unregister_event_lin_whandle

函数名称	**tsapp_unregister_event_lin_whandle(HwHandle:c_size_t, ACallback:OnTx_RxFUNC_LIN_WHandle):->int**
功能介绍	反注册 LIN 数据接收函数
调用位置	在不需要接收 LIN 报文的时候，调用此函数
输入参数	HwHandle：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	def On_LIN_Event(obj,msg):
	if msg.contents.is_tx():
	pass #表明该报文为设备发出报文
	else:

	pass#表明该报文为设备接收报文 ONLINEvent = OnTx_RxFUNC_LIN_WHandle(On_LIN_Event) ret = taspp_register_event_lin_whandle(HwHandle, ONLINEvent) ret = taspp_unregister_event_lin_whandle(HwHandle, ONLINEvent)

26. tsapp_transmit_lin_sync

函数名称	**tsapp_transmit_lin_sync(HwHandle:c_size_t, ALIN :TLIBLIN,ATimeOutMS:c_uint32):->int**
功能介绍	以同步的方式发送 LIN 报文
调用位置	发送 LIN 报文
输入参数	HwHandle：设备句柄 ALIN：LIN 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	ALIN = TLIBLIN(FIdxChn=0, FDLC=8, FIdentifier=0x11, FProperties=1, FData=[1,2,3,4,5,6,7,8]) tsapp_transmit_lin_sync(HwHandle,ALIN,100)

27. tsapp_transmit_lin_async

函数名称	**tsapp_transmit_lin_async(HwHandle:c_size_t, ALIN:TLIBLIN):->int**
功能介绍	以异步的方式发送 LIN 报文
调用位置	发送 LIN 报文
输入参数	HwHandle：设备句柄 ALIN：LIN 数据包
返回值	==0：注册成功 其他值：注册失败
示例	ALIN = TLIBLIN(FIdxChn=0, FDLC=8, FIdentifier=0x11, FProperties=1, FData=[1,2,3,4,5,6,7,8]) tsapp_transmit_lin_async(HwHandle,ALIN)

28. tsapp_register_event_flexray_whandle

函数名称	**tsapp_register_event_flexray_whandle(HwHandle:c_size_t, ACallback:OnTx_RxFUNC_Flexray_WHandle):->int**
功能介绍	注册 **Flexray** 数据包接收回调函数
调用位置	在 **Flexray** 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	HwHandle：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	def On_flexray_Event(obj,msg):
	if msg.contents.is_tx():
	pass #表明该报文为设备发出报文
	else:
	pass#表明该报文为设备接收报文
	ONFlexrayEvent = OnTx_RxFUNC_Flexray_WHandle(On_**LIN**_Event)
	ret=taspp_register_event_flexray_whandle(HwHandle, ONFlexrayEvent)

29. tsapp_unregister_event_flexray_whandle

函数名称	**tsapp_unregister_event_flexray_whandle(HwHandle:c_size_t, ACallback:OnTx_RxFUNC_Flexray_WHandle):->int**
功能介绍	反注册 **Flexray** 数据接收函数
调用位置	在不需要接收 **Flexray** 报文的时候，调用此函数
输入参数	HwHandle：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	def On_flexray_Event(obj,msg):
	if msg.contents.is_tx():
	pass #表明该报文为设备发出报文
	else:
	pass#表明该报文为设备接收报文
	ONFlexrayEvent = OnTx_RxFUNC_Flexray_WHandle(On_**LIN**_Event)
	ret=taspp_register_event_flexray_whandle(HwHandle, ONFlexrayEvent)
	ret=taspp_unregister_event_flexray_whandle(HwHandle, ONFlexrayEvent)

30. tsapp_transmit_flexray_sync

函数名称	**tsapp_transmit_flexray_sync(HwHandle:c_size_t,** **AFlexray :TLIBFlexray,ATimeOutMS:c_uint32):->int**
功能介绍	以同步的方式发送 **Flexray** 报文
调用位置	发送 **Flexray** 报文
输入参数	HwHandle：设备句柄 **Flexray**：**Flexray** 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	AFlexray= TLIBFlexray(FIdxChn=0, FSlotId=8, FChannelMask = 1, FCycleNumber=1, FActualPayloadLength=8, FData=[1,2,3,4,5,6,7,8]) tsapp_transmit_flexray_sync(HwHandle,AFlexray,100)

31. tsapp_transmit_flexray_async

函数名称	**tsapp_transmit_flexray_async(HwHandle:c_size_t,** AFlexray**:TLIBFlexray):->int**
功能介绍	以异步的方式发送 Flexray 报文
调用位置	发送 Flexray 报文
输入参数	HwHandle：设备句柄 AFlexray：Flexray 数据包
返回值	==0：注册成功 其他值：注册失败
示例	AFlexray= TLIBFlexray(FIdxChn=0, FSlotId=8, FChannelMask = 1, FCycleNumber=1, FActualPayloadLength=8, FData=[1,2,3,4,5,6,7,8]) tsapp_transmit_flexray_async(HwHandle,AFlexray)

注：以下为 windows 专有 API，后续 Linux 开放，将更新该文档

32. tsdiag_can_create

函数名称	**tsdiag_can_create(HwHandle:c_size_t, pDiagModuleIndex:c_uint8,AChnidx:c_uint32,AISFD:c_uint8,AMaxDLC:c_uint8,ARequestID:c_int32,AReqIsStd:c_bool,ARespondID:c_int32,AResIsStd:c_bool,AFunctionID:c_int32,AFuncIsStd:c_bool):->int**
功能介绍	创建诊断模块
调用位置	连接硬件之后，需要创建诊断模块时
输入参数	HwHandle：设备句柄 pDiagModuleIndex：返回诊断模块的句柄 AChnidx:诊断模块绑定的 CAN 通道 AISFD:是否为 CANFD AMaxDLC:最大字节长度 ARequestID：请求 ID AReqIsStd:是否为标准帧 ARespondID：响应 ID AResIsStd:是否为标准帧 AFunctionID：功能 ID AFuncIsStd:是否为标准帧
返回值	==0：注册成功 其他值：注册失败
示例	udsHandle = c_int8(0) ChnIndex = CHANNEL_INDEX.CHN1 ASupportFD = c_byte(1) AMaxdlc = c_byte(8) reqID = c_int32(0x7e0) ARequestIDIsStd = False resID = c_int32(0x7e3) resIsStd = False AFctID = c_int32(0x7df) fctIsStd = False tsdiag_can_create(HWHandle,udsHandle,ChnIndex,ASupportFD,AMaxdlc,reqID,resIsStd,resID,resIsStd,AFctID,fctIsStd)

33. tsdiag_can_delete

函数名称	**tsdiag_can_create(pDiagModuleIndex:c_uint8):->int**
功能介绍	删除创建的诊断模块
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	HwHandle：设备句柄 pDiagModuleIndex：返回诊断模块的句柄

返回值	==0：注册成功 其他值：注册失败
示例	udsHandle = c_int8(0) ChnIndex = CHANNEL_INDEX.CHN1 ASupportFD = c_byte(1) AMaxdlc = c_byte(8) reqID = c_int32(0x7e0) ARequestIDIsStd = False resID = c_int32(0x7e3) resIsStd = False AFctID = c_int32(0x7df) fctIsStd = False tsdiag_can_create(HWHandle,udsHandle,ChnIndex,ASupportFD,AMax dlc,reqID,resIsStd,resID,resIsStd,AFctID,fctIsStd) **tsdiag_can_delete(udsHandle)**

34. tstp_can_send_functional

函数名称	**tstp_can_send_functional(pDiagModuleIndex:c_uint8,Data:bytes,Datalen:c_int32):->int**
功能介绍	功能 id 请求数据
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	pDiagModuleIndex：诊断模块句柄 Data：发送的数据 Datalen:发送数据的长度
返回值	==0：注册成功 其他值：注册失败
示例	Data = [0x10,0x03] tstp_can_send_functional(udsHandle,bytes(Data),len(Data))

35. tstp_can_send_request

函数名称	**tstp_can_send_request(pDiagModuleIndex:c_uint8,Data:bytes,Datalen:c_int32):->int**
功能介绍	请求 id 请求数据
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	pDiagModuleIndex：诊断模块句柄 Data：发送的数据 Datalen:发送数据的长度
返回值	==0：注册成功 其他值：注册失败
示例	Data = [0x10,0x03] tstp_can_send_request(udsHandle,bytes(Data),len(Data))

36. tstp_can_request_and_get_response

函数名称	**tstp_can_request_and_get_response(pDiagModuleIndex:c_uint8,Data:bytes,Datalen:c_int32,ResData:pu8,ResDatalen:s32):->int**
功能介绍	请求 id 请求数据并获取响应数据
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	pDiagModuleIndex：诊断模块句柄 Data：发送的数据 Datalen:发送数据的长度
返回值	==0：注册成功 其他值：注册失败
示例	Data = [0x10,0x03] ResData = (u8*100)() ResDatalen = s32(100) tstp_can_request_and_get_response(udsHandle,bytes(Data),len(Data),ResData, ResDatalen) print(ResData[:ResDatalen.value])

37. tsdiag_can_session_control

函数名称	**tstp_can_session_control(pDiagModuleIndex:c_uint8,AControlType:u8):->int**
功能介绍	10 服务
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	pDiagModuleIndex：诊断模块句柄 Data：发送的数据 Datalen:发送数据的长度
返回值	==0：注册成功 其他值：注册失败
示例	tstp_can_session_control(udsHandle,0x01)

38. tsdiag_can_security_access_request_seed

函数名称	**tsdiag_can_security_access_request_seed(pDiagModuleIndex:c_uint8,ALevel:s32,FSeedData:pu8,FDataLen:ps32):->int**
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	pDiagModuleIndex：诊断模块句柄 ALevel：安全等级 FSeedData:seed 数据 FDataLen :seed 长度
返回值	==0：注册成功 其他值：注册失败

示例	ResData = (u8*100)() ResDatalen = s32(100) tsdiag_can_security_access_request_seed(udsHandle,1,ResData ,ResDatalen) print(ResData[:ResDatalen.value])

39. tsdiag_can_security_access_send_key

函数名称	**tsdiag_can_security_access_send_key(pDiagModuleIndex:c_uint8,ALevel:s32,FKeyData:pu8,FDataLen:s32):->int**
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	pDiagModuleIndex：诊断模块句柄 ALevel：安全等级 FKeyData:key 数据 FDataLen :key 长度
返回值	==0：注册成功 其他值：注册失败
示例	FKeyData = [1,2,3,4] tsdiag_can_security_access_send_key(udsHandle,2,bytes(FKeyData),len(FKeyData))

40. tsdiag_can_request_download

函数名称	**tsdiag_can_request_download(pDiagModuleIndex:c_uint8,AMemaddr:u32,AMemsize:u32):->int**
功能介绍	34 服务
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	pDiagModuleIndex：诊断模块句柄 **AMemaddr**:地址 **AMemsize**:地址大小
返回值	==0：注册成功 其他值：注册失败
示例	AMemaddr = 0xA0000 AMemsize = 0x1000 tsdiag_can_request_download(udsHandle,AMemaddr ,AMemsize)

41. tsdiag_can_request_upload

函数名称	**tsdiag_can_request_upload(pDiagModuleIndex:c_uint8,AMemaddr:u32,AMemsize:u32):->int**
功能介绍	35 服务
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	pDiagModuleIndex：诊断模块句柄

	AMemaddr:地址 **AMemsize**:地址大小
返回值	==0：注册成功 其他值：注册失败
示例	AMemaddr = 0xA0000 AMemsize = 0x1000 tsdiag_can_request_upload(udsHandle,AMemaddr ,AMemsize)

42. tsdiag_can_transfer_data

函数名称	**tsdiag_can_transfer_data(pDiagModuleIndex:c_uint8,Data:pu8,Datalen:s32,AReqCase:s32):->int**
功能介绍	36 服务
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	pDiagModuleIndex：诊断模块句柄 Data:传输数据 DataSize:数据长度 AReqCase:请求 case
返回值	==0：注册成功 其他值：注册失败
示例	Data = [1,2,3,4,5,6,7,8,9] tsdiag_can_transfer_data(udsHandle,bytes(Data),len(Data),1)

43. tsdiag_can_write_data_by_identifier

函数名称	**tsdiag_can_write_data_by_identifier (pDiagModuleIndex:c_uint8,DataID:U16,Data:pu8,Datalen:s32):->int**
功能介绍	36 服务
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	pDiagModuleIndex：诊断模块句柄 DataID:数据 ID Data:传输数据 DataSize:数据长度
返回值	==0：注册成功 其他值：注册失败
示例	Data = [1,2,3,4,5,6,7,8,9] tsdiag_can_write_data_by_identifier (udsHandle,0xF190,bytes(Data),len(Data))

44. tsdiag_can_read_data_by_identifier

函数名称	**tsdiag_can_read_data_by_identifier (pDiagModuleIndex:c_uint8,DataID:U16,Data:pu8,Datalen:s32):->int**
功能介绍	36 服务

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	pDiagModuleIndex：诊断模块句柄 DataID:数据 ID Data:接收数据 DataSize:接收数据长度
返回值	==0：注册成功 其他值：注册失败
示例	Data =(u8*100) Datalen = s32(100) tsdiag_can_read_data_by_identifier (udsHandle,0xF190,Data,Datalen)