

TsCAN API编程指导 Linux C++版

V1.2



1. 目录

1. 什么情况下需要此文档?	4
2. 添加库文件	4
1. C 语言:	4
2. CPP 语言	5
3. 测试验证	5
4. 数据类型定义	6
1. TLibCAN: CAN 总线数据类型	6
2. TLibCANFD: CANFD 总线数据类型	7
3. TLibLIN: LIN 总线数据类型	7
5. 发送/接收报文	8
1. 发送报文	8
tscan_transmit_can_async	8
tscan_transmit_can_sync	9
2. 接收报文	9
基于 FIFO 方式	9
基于回调函数	10
6. 接口函数介绍	10
1. initialize_lib_tscan	10
2. finalize_lib_tscan	10
3. tscan_scan_devices	11
4. tscan_connect	11
5. tscan_disconnect_by_handle	11
6. tscan_config_can_by_baudrate	12
7. tscan_register_event_can	12
8. tscan_unregister_event_can	12
9. tsfifo_add_can_canfd_pass_filter	13
10. tsfifo_receive_can_msgs	13
11. tsfifo_clear_can_receive_buffers	14
12. tscan_transmit_can_async	14

13.	tscan_config_canfd_by_baudrate.....	15
14.	tscan_register_event_canfd.....	15
15.	tscan_unregister_event_canfd	16
16.	tsfifo_receive_canfd_msgs	16
17.	tsfifo_clear_can_receive_buffers.....	17
18.	tscan_transmit_canfd_async.....	17
19.	tslin_set_node_funtiontype.....	18
20.	tslin_config_baudrate	18
21.	tslin_register_event_lin	19
22.	tslin_unregister_event_lin	19
23.	tsfifo_add_lin_pass_filter.....	20
24.	tsfifo_receive_lin_msgs.....	20
25.	tsfifo_clear_lin_receive_buffers.....	21
26.	tslin_transmit_lin_async.....	21
7.	示例工程.....	22

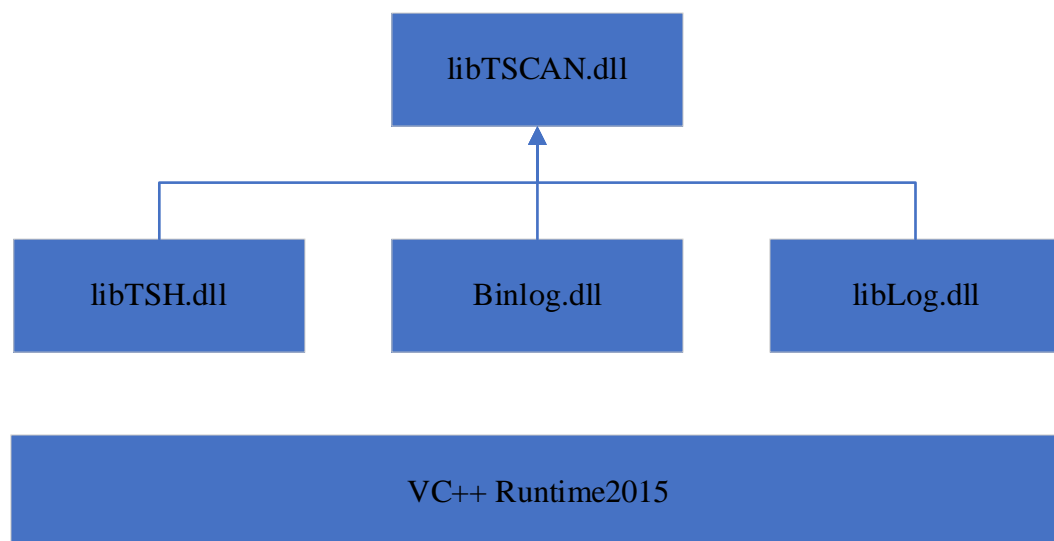
1. 什么情况下需要此文档？

用户基于C++编程语言，对上海同星智能科技有限公司的TSCAN系列工具(TC1001,TC1011,TC1012,TC1013,TC1014,TC1016)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。

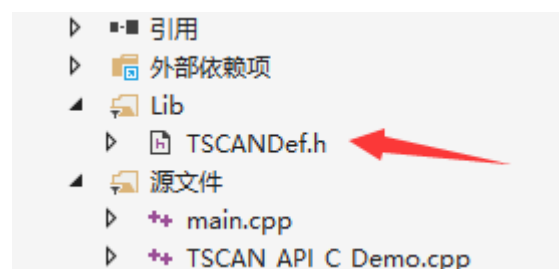
2. 添加库文件

1. C 语言：

要实现对TOSUN系列CAN/CANFD，LIN设备的操作，需要基于libTSCAN.dll动态链接库文件。该文件集成了上海同星公司对TS系列工具设备在Win32平台上的所有API接口。libTSCAN.dll的运行，除了依赖常用的C++运行库如mfc140.dll，msvc140.dll等，还需要依赖libTSH.dll，binlog.dll以及liblog.dll。其库文件依赖关系如下图所示：



要调用dll内部的接口函数，需要在工程中添加TSCANDef.h头文件。该文件中主要定义了使用API所需要用到的数据结构类型以及函数指针类型，如下所示：



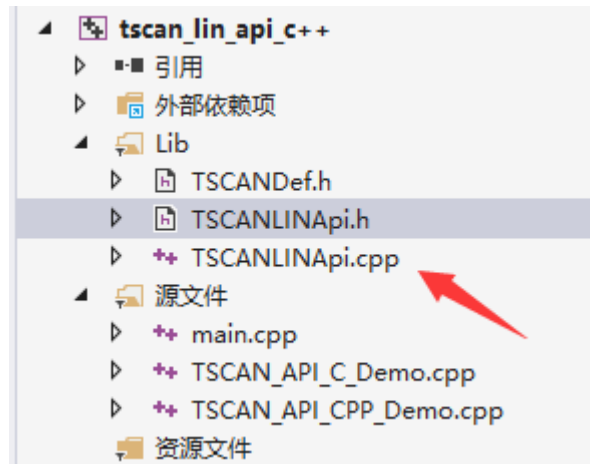
1. 引用头文件

开发人员可以根据头文件定义，直接引用lib库文件进行开发。也可以根据数据结构定

义，动态载入函数指针，详细情况见例程。

2. C++ 语言

为了方便C++人员操作硬件设备，本工程提供了基于C++类的封装TSCANLINApi。要引用此定义，需要引用三个文件：TSCANLINApi.h, TSCANLINApi.c以及TSCANDef.h.如下图所示：

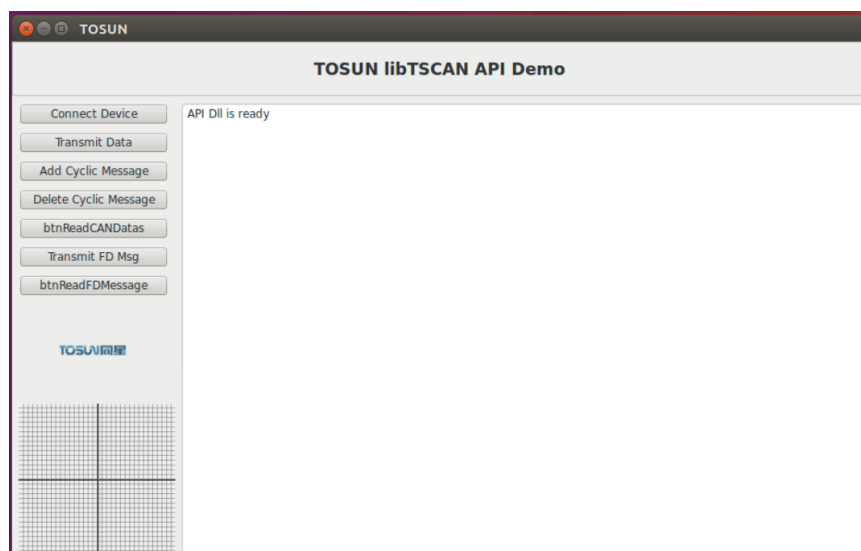


TSCANAPI C++库文件引用

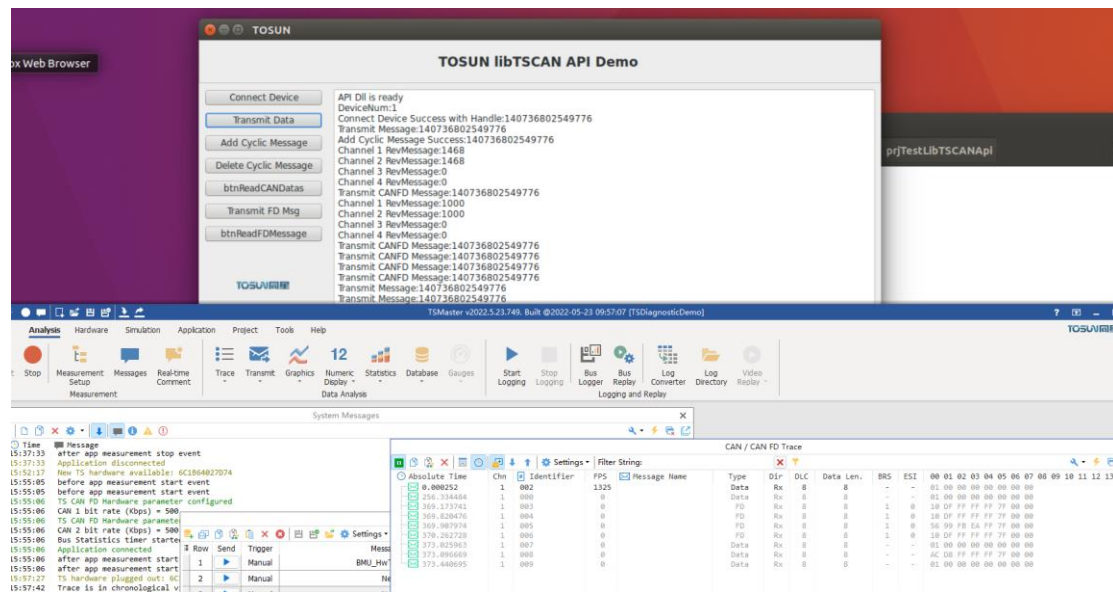
在TSCANLINApi.c中，封装了常用的操作函数，详细的调用方法见例程所示。

3. 测试验证

在驱动库的目录下面，附带了一个简单的测试UI程序prjTestLibTSCANApi。用管理员权限打开此程序，界面如下图所示所示：



用户可以插上TOSUN的设备，并验证设备连接，周期发送报文，读取报文等功能。其中设备连接过程中，默认初始化各个通道的波特率参数为：仲裁场500kBps，数据场2000kBps，使能内部120欧终端电阻。联合调试效果如下图所示：



4. 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;
```

【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. 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;
```

【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

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;
```

【1】 FProperties: 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

5. 发送/接收报文

1. 发送报文

本章节以发送 CAN 报文为例进行讲解。其他 CANFD/LIN 的 API 函数完成对应。发送函数主要包括异步发送函数 `tscan_transmit_can_async` 和同步发送函数 `tscan_transmit_can_sync`。

`tscan_transmit_can_async`

异步发送 Can 报文。异步发送, 就是把 CAN 报文推到发送 FIFO 过后, 就直接退出。该报文是否确定发送成功, 对于调用者来说在退出函数的时候是无法确认的。该函数是最常用的 API 函数, 大部分的驱动都会提供此功能函数。其调用方式如下:

```
TCAN msg; //定义一帧LIN报文
msg.FIdentifier = 0x3C; //定义数据ID为0x3C
msg.FDLC = 3; //数据长度为3
msg.FIdxChn = 0;
msg.FProperties.value = 0x00; //清除原始属性
msg.FProperties.bits.istx = 1; //设置属性为发送报文
msg.FData.d[0] = (byte)0x12;
msg.FData.d[1] = (byte)0x34;
msg.FData.d[2] = (byte)0x56;
tscan_transmit_canfd_async(ADeviceHandle, &msg);
```

表示创建一个 TCAN 类型的帧 msg, 发送通道为 0, 帧 ID 为 0x3C, 数据长度为 3 个字节。然后三个字节分别为 0x12, 0x34, 0x56。发送 msg。

tscan_transmit_can_sync

同步发送 Can 报文。同步发送，就是把 CAN 报文推到发送 FIFO 过后，调用方要等待，直到设备端成功发送了该报文，并返回成功才退出；或者一直不能返回成功，超时退出。其特点是该函数执行过后，调用方能够明确的知道该报文是否发送成功；缺点是因为是阻塞等待，所以如果在追求高负载率传输的场合，此函数就不能运用了。该函数是 TOSUN 驱动包特有的函数。其调用方式如下：

```
TCAN msg;    //定义一帧LIN报文
msg.FIdentifier = 0x3C; //定义数据ID为0x3C
msg.FDLC = 3;      //数据长度为3
msg.FIdxChn = 0;
msg.FProperties.value = 0x00; //清除原始属性
msg.FProperties.bits.istx = 1; //设置属性为发送报文
msg.FData.d[0] = (byte)0x12;
msg.FData.d[1] = (byte)0x34;
msg.FData.d[2] = (byte)0x56;
tscan_transmit_canfd_sync(ADeviceHandle,&msg,200);
```

表示创建一个 TCAN 类型的帧 msg，发送通道为 0，帧 ID 为 0x3C，数据长度为 3 个字节。然后三个字节分别为 0x12,0x34,0x56。发送 msg，超时判断参数为 200ms，如果超过 200ms 该报文没有发送成功，则返回超时失败。

2. 接收报文

TSMaste 报文接收方式有两种：基于回调函数方式和基于读取 FIFO 方式。下面分别予以介绍：

基于 FIFO 方式

采用 tsfifo_receive_canfd_msgs 函数从驱动缓存中读取当前已经收到的报文。其主要包含如下参数：

参数名称	数据方向	ADeviceHandle[IN]：设备句柄。
ADeviceHandle	输入[IN]	ADeviceHandle[IN]：设备句柄。
ACANFDBuffers	输出[OUT]	ACANFDBuffers[OUT]：报文数组首地址，该首地址表示用于存储读取的报文的首地址。
ACANFDBufferSize	输入输出 [IN, OUT]	IN:表示传入的报文数组的尺寸，驱动内部才知道一次性最多读取多少个数据，否则造成内存越界。 OUT：表示实际读取的报文数量。比如*ACANBufferSize 在传进去的时候等于 20，函数执行过后变成了 10，表示实际读取了 10 个报文。
AChn	输入[IN]	
ARxTx	输入[IN]	ARxTx[IN]：=0：只读取从其他节点接收到的报文；1：把自己发出去的和从其他节点收到的报文都读取出来。

基于回调函数

回调函数方式相当于 MCU 中的中断机制。当驱动中收到一个完整的 CAN 数据包过后，就会触发此回调函数执行。要使用回调函数，主要使用以下两个函数：注册回调函数，反注册回调函数。如下所示：

```
retValue = tscan_register_event_can(ADeviceHandle, ReceiveCANMessage);
```

当注册了 can 回调函数过后，当驱动中收到一帧 CAN 报文过后，就会从内部触发 ReceiveCANMessage 函数执行，如下所示：

```
void ReceiveCANMessage(const TLibCAN* AData)
{
    //处理收到的报文数据 AData
}
```

当不需要执行此接收回调函数的时候，需要调用 unregister 反注册此接收处理函数，如下所示：

```
retValue = tscan_unregister_event_can(ADeviceHandle, ReceiveCANMessage);
```

6. 接口函数介绍

1. initialize_lib_tscan

函数名称	void initialize_lib_tscan(bool AEnableFIFO, bool AEnableErrorFrame, bool AEnableTurbe)
功能介绍	初始化 libTSCANOnLinux 模块。
调用位置	初始化此模块过后，其他 API 函数才能被调用。
输入参数	AEnableFIFO[IN]：是否开启 FIFO 机制，建议设置为 True，否则用户无法通过 tsfifo_receive_xx 函数读取报文。 AEnableErrorFrame[IN]：是否接收错误帧。如果设置为 False，则驱动直接把错误帧抛弃掉。 AEnableTurbo[IN]：直接设置为 False 即可。
返回值	无
示例	initialize_lib_tscan(true, false, false);

2. finalize_lib_tscan

函数名称	void finalize_lib_tscan (void)
功能介绍	释放 libTSCANOnLinux 模块。
调用位置	此函数跟 initialize_lib_tscan 函数是对应的。在使用完 API 模块过后，退出程序之前，一定要调用此函数释放掉驱动模块。

输入参数	无
返回值	无
示例	<code>finalize_lib_tscan();</code>

3. tscan_scan_devices

函数名称	<code>u32 tscan_scan_devices(u32* ADeviceCount)</code>
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	<code>ADeviceCount[IN]</code> : 指针参数, 指向设备数量
返回值	<code>==0</code> : 获取成功 其他值: 获取失败
示例	<code>uint32_t ADeviceCount;</code> <code>tscan_scan_devices(&ADeviceCount);</code> //扫描设备数量, 并存储在变//量 ADeviceCount 中

4. tscan_connect

函数名称	<code>u32 tscan_connect(const char* ADeviceSerial, size_t* AHandle)</code>
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	<code>ADeviceSerial[IN]</code> : <code>!= NULL</code> , 获取指定序列号的设备 <code>==NULL</code> , 获取任意处于连接状态的设备 <code>AHandle[IN]</code> : 设备句柄
返回值	<code>==0</code> : 连接成功 <code>==5</code> : 设备已经连接
示例	<code>u32 retValue = tscan_connect("", &ADeviceHandle);</code> //连接字符串为空, 则设备连接默认设备 <code>u32 retValue = tscan_connect("512356EF32CD", &ADeviceHandle);</code> //连接字符串不为空, 则连接指定串行号设备

5. tscan_disconnect_by_handle

函数名称	<code>u32 tscan_disconnect_by_handle(const size_t ADeviceHandle)</code>
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	<code>ADeviceHandle[IN]</code> : 设备句柄
返回值	<code>==0</code> : 断开设备成功 其他值: 断开设备失败
示例	

6. tscan_config_can_by_baudrate

函数名称	u32 tscan_config_can_by_baudrate(const size_t ADeviceHandle, const APP_CHANNEL AChnIdx, const double ARateKbps, const u32 A1200hmConnected)
功能介绍	配置 CAN 总线波特率
调用位置	在调用 CAN 报文收发 API 之前，调用此 API 初始化总线波特率等。
输入参数	ADeviceHandle[IN]:设备句柄 AChnIdx[IN]:通道参数 ARateKbps[IN]:波特率参数，比如 500 代表 500kbps A1200hmConnected[IN]:是否使能 120 Ω 终端电阻
返回值	==0: 函数执行成功 其他值: 函数执行失败
示例	tscan_config_can_by_baudrate (ADeviceHandle, CHN1, 500, 1); //配置通道 1 的波特率参数为 500kbps，并且使能终端电阻

7. tscan_register_event_can

函数名称	u32 tscan_register_event_can(const size_t ADeviceHandle, const TCANQueueEvent_Win32_t 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[IN]: 设备句柄; ACallback[IN]: 接收数据的处理函数
返回值	==0: 注册成功 其他值: 注册失败
示例	<pre>retValue = tscan_register_event_can(ADeviceHandle, ReceiveCANMessage); ReceiveCANMessage 是报文处理函数，其定义如下： void ReceiveCANMessage(const TLibCAN* AData) { if(AData->FProperties.bits.istx){ qDebug()<<"tx frame with id 0x"<<QString::number(AData->FIdentifier, 16);}else{ qDebug()<<"rx frame with id 0x"<<QString::number(AData->FIdentifier, 16);}</pre>

8. tscan_unregister_event_can

函数名称	u32 tscan_unregister_event_can(const size_t ADeviceHandle, const TCANQueueEvent_Win32_t ACallback)
功能介绍	反注册 CAN 数据接收函数

调用位置	在不需要通过回调函数的形式接收 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: 反注册成功 其他值: 反注册失败
示例	<pre>retValue = tscan_unregister_event_can(ADeviceHandle, ReceiveCANMessage);</pre>

9. tsfifo_add_can_canfd_pass_filter

函数名称	u32 tsfifo_add_can_canfd_pass_filter(const size_t ADeviceHandle, const APP_CHANNEL AChnIdx, const s32 AIdentifier, const bool AIsStd)
功能介绍	添加 can/canfd 过滤报文
调用位置	用户如果只想接收特定 ID 报文的时候，需要调用此函数
输入参数	ADeviceHandle: 设备句柄; ACallback: 接收数据处理函数委托 AIdentifier: 报文 ID AIsStd: 是否标准帧
返回值	==0: 执行成功 其他值: 执行失败
示例	<pre>tsfifo_add_can_canfd_pass_filter(ADeviceHandle, CHN1, 0x123, true); //把 0x123 报文添加到过滤器中</pre>

10. tsfifo_receive_can_msgs

函数名称	u32 tsfifo_receive_can_msgs(const size_t ADeviceHandle, const TLibCAN* ACANBuffers, s32* ACANBufferSize, APP_CHANNEL AChn, TREAD_TX_RX_DEF ARxTx)
功能介绍	从 fifo 中读取收到的 CAN 报文
调用位置	用户读取收到 (包含发送出去的和接收到的) 的 CAN 报文。如果用读取 fifo 的方式读取 CAN 报文，需要在 initalize_lib_tscan 函数中第一个参数 EnableFiFo 设置为 true，否则无法从 fifo 中读取数据。
输入参数	ADeviceHandle[IN]: 设备句柄。 ACANBuffers[OUT]: 报文数组首地址，该首地址表示用于存储读取的报文的首地址。 ACANBufferSize[IN, OUT]: 该参数是一个 IN, OUT 参数。 IN: 表示传入的报文数组的尺寸，驱动内部才知道一次性最多读取多少个数据，否则造成内存越界。 OUT: 表示实际读取的报文数量。比如*ACANBufferSize 在传进去的时候等于 20，函数执行过后变成了 10，表示实际读取了 10 个报文。 AChn[IN]: 需要读取的报文通道 ARxTx[IN]: =0: 只读取从其他节点接收到的报文; 1: 把自己发出去的和从其他节点收到的报文都读取出来。
返回值	==0: 读取数据成功

	其他值：读取数据失败
示例	<pre>TLibCAN readDataBuffer[20]; //首先创建一个 20 个元素的报文数组，用于存//储从 fifo 读取的报文 int realDataSize = 20; //报文大小是 IN，OUT 参数，所以要先设置 //初始值 //Reveive data from FIFO of Driver if(tsfifo_receive_can_msgs(ADeviceHandle,readDataBuffer,&realDataSize,CHN1,1) == 0x00) { for(int i = 0; i< realDataSize; i++) { qDebug()<<"read frame from fifo with id 0x"<<QString::number(readDataBuffer[i].FIdentifier, 16); } }</pre>

11. tsfifo_clear_can_receive_buffers

函数名称	u32 tsfifo_clear_can_receive_buffers(const size_t ADeviceHandle, const APP_CHANNEL AChnIdx)
功能介绍	清除指定通道 FIFO 里面的 CAN 报文
调用位置	需要清除指定通道 FIFO 内部的报文，仿真内部缓存的报文太多影响了最新报文的接收。
输入参数	ADeviceHandle[IN]：设备句柄。 AChnIdx[IN]：CAN 报文通道，清除指定通道 FIFO 内部的报文
返回值	==0：函数执行成功 其他值：函数执行失败
示例	tsfifo_clear_can_receive_buffers(ADeviceHandle, CHN1); //清除指定通道的 FIFO

12. tscan_transmit_can_async

函数名称	u32 tscan_transmit_can_async(const size_t ADeviceHandle, const TLibCAN* ACAN)
功能介绍	异步方式发送 CAN 报文
调用位置	在需要发送 CAN 报文的场合
输入参数	ADeviceHandle[IN]：设备句柄。 ACAN[IN]：CAN 报文
返回值	==0：发送报文成功 其他值：函数执行失败
示例	<pre>TLibCAN msg; //首先定义 CAN 报文 msg.FIdentifier = 0x03; //设置报文 ID</pre>

	<pre>msg.FProperties.bits.remoteframe = 0x00; //not remote frame, standard frame msg.FProperties.bits.extframe = 0; //设置是否数据帧, 远程帧等属性 msg.FDLC = 3; //设置要发送的报文长度 msg.FIdxChn = CHN1; //设置报文发送的通道 tscan_transmit_can_async (ADeviceHandle, &msg);</pre>
--	--

13. tscan_config_canfd_by_baudrate

函数名称	u32 tscan_config_canfd_by_baudrate(const size_t ADeviceHandle, const APP_CHANNEL AChnIdx, const double AArbRateKbps, const double ADataRateKbps, const TLIBCANFDControllerType AControllerType, const TLIBCANFDControllerMode AControllerMode, const u32 A1200hmConnected)
功能介绍	配置 CANFD 总线波特率
调用位置	在调用 CANFD 报文收发 API 之前, 调用此 API 初始化总线波特率等。
输入参数	ADeviceHandle[IN]:设备句柄 AChnIdx[IN]:通道参数 AArbRateKbps[IN]:仲裁场波特率参数, 比如 500 代表 500kbps ADataRateKbps[IN]:数据场波特率参数, 比如 2000 代表 2000kbps AControllerType[IN]:控制器类型, 主要包含: lfdtCAN: 普通 CAN 模式 lfdtISOCAN: ISO-CANFD 模式 lfdtNonISOCAN: NoISO-CANFD 模式 AControllerMode[IN]:控制器模式, 主要包含: lfdmNormal: 正常工作模式 lfdmACKOff: 关闭 ACK 应答模式 lfdmRestricted: 受限模式 lfdmInternalLoopback: 设备内循环模式 lfdmExternalLoopback: 设备外循环模式 A1200hmConnected[IN]:是否使能 120Ω 终端电阻
返回值	==0:函数执行成功 其他值: 函数执行失败
示例	<pre>tscan_config_canfd_by_baudrate(ADeviceHandle, CHN1, 500,2000, lfdtISOCAN, lfdmNormal,1); //配置通道 1 的波特率参数为仲裁场 500kbps, 数据场 2000kbps, ISO-CANFD 模式, 正常工作模式, 并且使能终端 电阻。</pre>

14. tscan_register_event_canfd

函数名称	u32 tscan_register_event_canfd(const size_t ADeviceHandle, const TCANFDQueueEvent_Win32_t 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[IN]: 设备句柄; ACallback[IN]: 接收数据的处理函数
返回值	==0: 注册成功 其他值: 注册失败
示例	<pre>retValue = tscan_register_event_canfd(ADeviceHandle, ReceiveCANMessage);</pre> <p>ReceiveCANFDMessage 是报文处理函数, 其定义如下:</p> <pre>void ReceiveCANFDMessage(const TLibCANFD* AData) { if(AData->FProperties.bits.istx){ qDebug()<<"tx frame with id 0x"<<QString::number(AData->FIdentifier, 16);}else{ qDebug()<<"rx frame with id 0x"<<QString::number(AData->FIdentifier, 16);} }</pre>

15. tscan_unregister_event_canfd

函数名称	u32 tscan_unregister_event_canfd(const size_t ADeviceHandle, const TCANFDQueueEvent_Win32_t ACallback)
功能介绍	反注册 CAN 数据接收函数
调用位置	在不需要通过回调函数的形式接收 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: 反注册成功 其他值: 反注册失败
示例	<pre>retValue = tscan_unregister_event_canfd(ADeviceHandle, ReceiveCANFDMessage);</pre>

16. tsfifo_receive_canfd_msgs

函数名称	u32 tsfifo_receive_canfd_msgs(const size_t ADeviceHandle, const TLibCANFD* ACANFDBuffers, s32* ACANFDBufferSize, APP_CHANNEL AChn, TREAD_TX_RX_DEF ARXTX)
功能介绍	从 fifo 中读取收到的 CANFD 报文
调用位置	用户读取收到 (包含发送出去的和接收到的) 的 CANFD 报文。如果用读取 fifo 的方式读取 CAN 报文, 需要在 initialize_lib_tscan 函数中第一个参数 EnableFiFo 设置为 true, 否则无法从 fifo 中读取数据。
输入参数	ADeviceHandle[IN]: 设备句柄。 ACANFDBuffers[OUT]: 报文数组首地址, 该首地址表示用于存储读取的报文的首地址。 ACANFDBufferSize[IN, OUT]: 该参数是一个 IN, OUT 参数。 IN: 表示传入的报文数组的尺寸, 驱动内部才知道一次性最多读取

	<p>多少个数据，否则造成内存越界。</p> <p>OUT：表示实际读取的报文数量。比如*ACANBufferSize 在传进去的时候等于 20，函数执行过后变成了 10，表示实际读取了 10 个报文。</p> <p>ACHn[IN]：需要读取的报文通道</p> <p>ARxTx[IN]：=0：只读取从其他节点接收到的报文；1：把自己发出去的和从其他节点收到的报文都读取出来。</p>
返回值	<p>==0：读取数据成功</p> <p>其他值：读取数据失败</p>
示例	<pre>TLibCANFD readDataBuffer[20]; //首先创建一个 20 个元素的报文数组，用于存 //储从 fifo 读取的报文 int realDataSize = 20; //报文大小是 IN，OUT 参数，所以要先设置 //初始值 //Reveive data from FIFO of Driver if(tsfifo_receive_canfd_msgs(ADeviceHandle, readDataBuffer, &realDataSize, CHN1, 1) == 0x00) { for(int i = 0; i< realDataSize; i++) { qDebug()<<"read frame from fifo with id 0x"<<QString::number(readDataBuffer[i].FIdentifier, 16); } }</pre>

17. tsfifo_clear_can_receive_buffers

函数名称	u32 tsfifo_clear_canfd_receive_buffers(const size_t ADeviceHandle, const APP_CHANNEL AChnIdx)
功能介绍	清除指定通道 FIFO 里面的 CAN 报文
调用位置	需要清除指定通道 FIFO 内部的报文，仿真内部缓存的报文太多影响了最新报文的接收。
输入参数	ADeviceHandle[IN]：设备句柄。 AChnIdx[IN]：CAN 报文通道，清除指定通道 FIFO 内部的报文
返回值	==0：函数执行成功 其他值：函数执行失败
示例	tsfifo_clear_canfd_receive_buffers(ADeviceHandle, CHN1); //清除指定通道的 FIFO

18. tscan_transmit_canfd_async

函数名称	u32 tscan_transmit_canfd_async(const size_t ADeviceHandle, const TLibCANFD* ACANFD)
功能介绍	异步方式发送 CANFD 报文

调用位置	在需要发送 CANFD 报文的情况
输入参数	ADeviceHandle[IN]: 设备句柄。 ACANFD[IN]: CANFD 报文
返回值	==0: 发送报文成功 其他值: 函数执行失败
示例	<pre>TLibCANFD fdmsg; //首先定义 CANFD 报文 fdmsg.FIdentifier = 0x03; //然后设置 CAN 报文 ID fdmsg.FProperties.bits.remoteframe = 0x00; //not remote frame, standard frame fdmsg.FProperties.bits.extframe = 0; //是否扩展帧, 远程帧等属性 fdmsg.FDLC = 3; //报文数据长度 fdmsg.FIdxChn = CHN1; //报文发送到哪一个通道中 fdmsg.FFDProperties.bits.EDL = 1; //FDMode: 是否 FD 模式报文 fdmsg.FFDProperties.bits.BRS = 1; //Open baudrate speed: 是否客气波特率可变 tscan_transmit_canfd_async (ADeviceHandle, &canmsg);</pre>

19. tslin_set_node_funtiontype

函数名称	u32 tslin_set_node_funtiontype(const size_t ADeviceHandle, const APP_CHANNEL AChnIdx, const TLIN_FUNCTION_TYPE AFunctionType)
功能介绍	设置设备的功能类型: 0: 主节点 (Master Node) 1: 从节点 (Slave Node) 2: 监听节点 (Monitor Node)
调用位置	LIN 总线工作之前, 需要设置该节点的工作模式
输入参数	ADeviceHandle: 设备句柄; AChnIdx: 通道编号 AFunctionType: 功能类型
返回值	==0: 函数执行成功 其他值: 执行失败
示例	<pre>tslin_set_node_funtiontype (ADeviceHandle, CHN1, MasterNode) //设置设备的 LIN 通道 1 工作在主节点模式下</pre>

20. tslin_config_baudrate

函数名称	u32 tslin_config_baudrate(const size_t ADeviceHandle, const APP_CHANNEL AChnIdx, const double ARateKbps, TLINProtocol AProtocol)
功能介绍	配置 LIN 总线总线波特率
调用位置	在调用 LIN 报文收发 API 之前, 调用此 API 初始化总线波特率等。
输入参数	ADeviceHandle[IN]: 设备句柄 AChnIdx[IN]: 通道参数 ARateKbps[IN]: 波特率参数, 比如 20 代表 20kbps AProtocol [IN]: LIN 总线协议版本。包括如下定义:

	LIN_Protocol_13: LIN1.3 版本 LIN_Protocol_20: LIN 协议 2.0 版本 LIN_Protocol_21: LIN 协议 2.1 版本 LIN_Protocol_J2602: LIN 协议 J2602 版本
返回值	==0: 函数执行成功 其他值: 函数执行失败
示例	tscan_config_lin_by_baudrate (ADeviceHandle, CHN1, 20, LIN_Protocol_21); //配置通道 1 的波特率参数为 20kbps, 协议版本为 2.1 版本

21. tslin_register_event_lin

函数名称	u32 tslin_register_event_can(const size_t ADeviceHandle, const TLINQueueEvent_Win32_t ACallback)
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	ADeviceHandle[IN]: 设备句柄; ACallback[IN]: 接收数据的处理函数
返回值	==0: 注册成功 其他值: 注册失败
示例	<pre>retValue = tscan_register_event_lin(ADeviceHandle, ReceiveLINMessage); ReceiveLINMessage 是报文处理函数, 其定义如下: void ReceiveLINMessage(const TLibLIN* AData) { if(AData->FProperties.bits.istx){ qDebug()<<"tx frame with id 0x"<<QString::number(AData->FIdentifier, 16);}else{ qDebug()<<"rx frame with id 0x"<<QString::number(AData->FIdentifier, 16);} }</pre>

22. tslin_unregister_event_lin

函数名称	u32 tslin_unregister_event_lin(const size_t ADeviceHandle, const TLINQueueEvent_Win32_t ACallback)
功能介绍	反注册 LIN 数据接收函数
调用位置	在不需要通过回调函数的形式接收 LIN 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: 反注册成功 其他值: 反注册失败
示例	retValue = tslin_unregister_event_lin(ADeviceHandle,

	ReceiveLINMessage);
--	---------------------

23. tsfifo_add_lin_pass_filter

函数名称	u32 tsfifo_add_lin_pass_filter(const size_t ADeviceHandle, const APP_CHANNEL AChnIdx, const s32 AIdentifier)
功能介绍	添加 can/canfd 过滤报文
调用位置	用户如果只想接收特定 ID 报文的时候，需要调用此函数
输入参数	ADeviceHandle: 设备句柄; ACallback: 接收数据处理函数委托 AIdentifier: 报文 ID
返回值	==0: 执行成功 其他值: 执行失败
示例	tsfifo_add_lin_pass_filter(ADeviceHandle, CHN1, 0x12); //把 0x12 报文添加到过滤器中

24. tsfifo_receive_lin_msgs

函数名称	u32 tsfifo_receive_lin_msgs(const size_t ADeviceHandle, const TLibLIN* ALINBuffers, s32* ALINBufferSize, APP_CHANNEL AChn, TREAD_TX_RX_DEF ARXTX)
功能介绍	从 fifo 中读取收到的 CAN 报文
调用位置	用户读取收到(包含发送出去的和接收到的)的 CAN 报文。如果用读取 fifo 的方式读取 CAN 报文，需要在 initialize_lib_tscan 函数中第一个参数 EnableFiFo 设置为 true，否则无法从 fifo 中读取数据。
输入参数	ADeviceHandle[IN]: 设备句柄。 ALINBuffers[OUT]: 报文数组首地址，该首地址表示用于存储读取的报文的首地址。 ALINBufferSize[IN,OUT]: 该参数是一个 IN, OUT 参数。 IN: 表示传入的报文数组的尺寸，驱动内部才知道一次性最多读取多少个数据，否则造成内存越界。 OUT: 表示实际读取的报文数量。比如*ALINBufferSize 在传进去的时候等于 20，函数执行过后变成了 10，表示实际读取了 10 个报文。 AChn[IN]: 需要读取的报文通道 ARxTx[IN]: =0: 只读取从其他节点接收到的报文；1: 把自己发出去的和从其他节点收到的报文都读取出来。
返回值	==0: 读取数据成功 其他值: 读取数据失败
示例	TLibLIN readDataBuffer[20]; //首先创建一个 20 个元素的报文数组，用于存储从 fifo 读取的报文 int realDataSize = 20; //报文大小是 IN, OUT 参数，所以要先设置 //初始值

```
//Reveive data from FIFO of Driver
if(tsfifo_receive_lin_msgs(ADeviceHandle, readDataBuffer, &realDataSize, CHN1, 1) == 0x00)
{
    for(int i = 0; i< realDataSize; i++)
    {
        qDebug()<<"read frame from fifo with id 0x"<<QString::number(readDataBuffer[i].FIdentifier, 16);
    }
}
```

25. tsfifo_clear_lin_receive_buffers

函数名称	u32 tsfifo_clear_lin_receive_buffers(const size_t ADeviceHandle, const APP_CHANNEL AChnIdx)
功能介绍	清除指定通道 FIFO 里面的 LIN 报文
调用位置	需要清除指定通道 FIFO 内部的报文，仿真内部缓存的报文太多影响了最新报文的接收。
输入参数	ADeviceHandle[IN]：设备句柄。 AChnIdx[IN]：LIN 报文通道，清除指定通道 FIFO 内部的报文
返回值	==0：函数执行成功 其他值：函数执行失败
示例	tsfifo_clear_lin_receive_buffers(ADeviceHandle, CHN1); //清除指定通道的 FIFO

26. tslin_transmit_lin_async

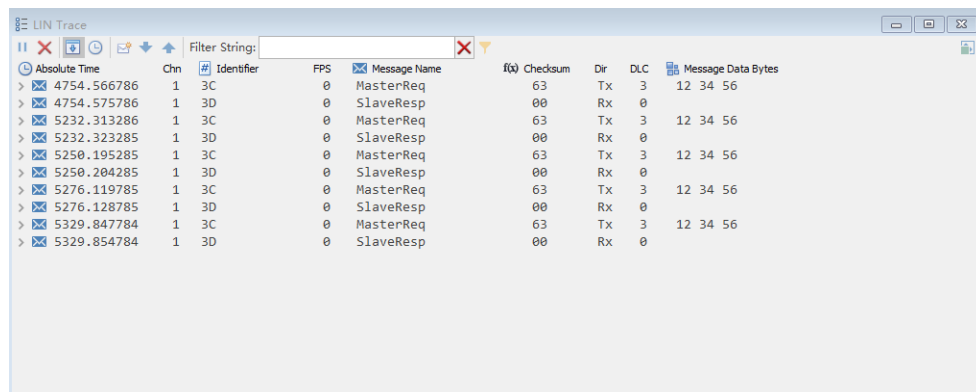
函数名称	u32 tslin_transmit_lin_async(const size_t ADeviceHandle, const TLibLIN* ALIN)
功能介绍	异步方式发送 LIN 报文
调用位置	在需要发送 LIN 报文的场合
输入参数	ADeviceHandle[IN]：设备句柄。 ALIN[IN]：LIN 报文
返回值	==0：发送报文成功 其他值：函数执行失败
示例	<pre>TLibLIN msg; //首先定义 LIN 报文 msg.FIdentifier = 0x03; //设置报文 ID msg.FDLC = 3; //设置要发送的报文长度 msg.FIdxChn = CHN1; //设置报文发送的通道 tslin_transmit_lin_async (ADeviceHandle, &msg);</pre>

7. 示例工程

本工程演示了调用API，实现加载DLL，扫描设备，连接设备，注册回调函数，设置波特率，发送LIN报文，接收LIN报文等过程，运行效果图：

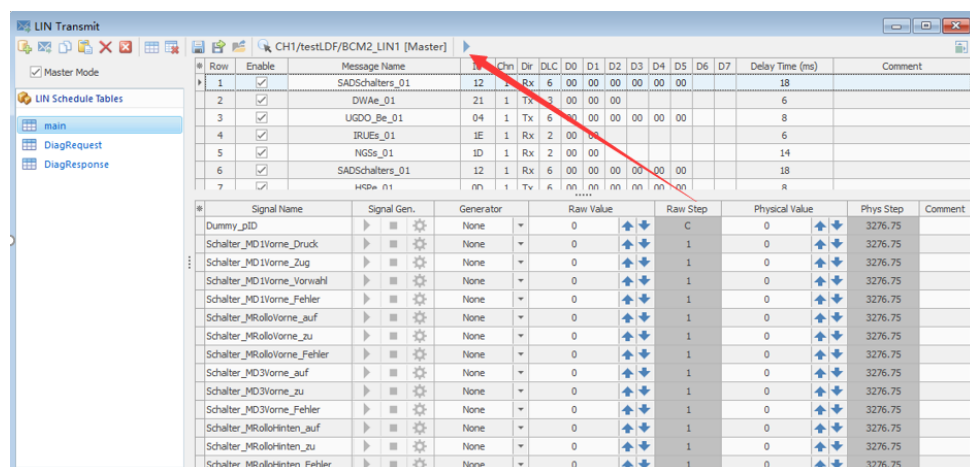
```
find libTSCAN.dll!  
TSCAN Device Count:1  
Device with handle:81784864 connected  
Device with handle:81784864 register rev callback success  
Device with handle:81784864 set baudrate success  
Device with handle:81784864 sync send can message failed  
Device with handle:81784864 async send can message success  
Device with handle:81784864 set function type success  
Device with handle:81784864 apply new ldf callback success  
Device with handle:81784864 register rev callback success  
Device with handle:81784864 set baudrate success  
Device with handle:81784864 sync send lin message success  
Receive Recall  
Translate message:Datalength:3 Datas: 12 34 56  
1 messages received  
Translate message:Datalength:3 Datas: 12 34 56  
Device with handle:81784864 async send lin message success  
Receive Recall  
Receive message:Datalength:0 Datas:  
1 messages received  
Receive message:Datalength:0 Datas:  
Device with handle:81784864 unregister rev callback success  
Disconnect device with handle:81784864 success  
end dll!
```

为了同步监测TSLIN报文通讯情况，可以打开同星公司的TSMaster软件，打开LIN Trace窗口，监测LIN报文通讯过程，如下所示：



Absolute Time	Chn	Identifier	FPS	Message Name	Checksum	Dir	DLC	Message Data Bytes
4754.566786	1	3C	0	MasterReq	63	Tx	3	12 34 56
4754.575786	1	3D	0	SlaveResp	00	Rx	0	
5232.313286	1	3C	0	MasterReq	63	Tx	3	12 34 56
5232.323285	1	3D	0	SlaveResp	00	Rx	0	
5250.195285	1	3C	0	MasterReq	63	Tx	3	12 34 56
5250.204285	1	3D	0	SlaveResp	00	Rx	0	
5276.119785	1	3C	0	MasterReq	63	Tx	3	12 34 56
5276.128785	1	3D	0	SlaveResp	00	Rx	0	
5329.847784	1	3C	0	MasterReq	63	Tx	3	12 34 56
5329.854784	1	3D	0	SlaveResp	00	Rx	0	

注意，TSMaster软件在监测过程中，尽量不要打开PC端的调度表下载仿真功能，这样会造成设备端默认维护一张调度表，影响用户程序调试。如下按钮：



Row	Enable	Message Name	Chn	Dir	DLC	D0	D1	D2	D3	D4	D5	D6	D7	Delay Time (ms)	Comment
1	<input checked="" type="checkbox"/>	SADSchalters_01	12	Rx	6	00	00	00	00	00	00			18	
2	<input checked="" type="checkbox"/>	DWAw_01	21	Tx	3	00	00	00	00	00				6	
3	<input checked="" type="checkbox"/>	UGDO_01	04	Tx	6	00	00	00	00	00				8	
4	<input checked="" type="checkbox"/>	IRUES_01	1E	Rx	2	00	00							6	
5	<input checked="" type="checkbox"/>	NGSs_01	1D	Rx	2	00	00							14	
6	<input checked="" type="checkbox"/>	SADSchalters_01	12	Rx	6	00	00	00	00	00				18	
7	<input checked="" type="checkbox"/>	MRp_01	1	Tx	6	00	00	00	00	00				18	

Signal Name	Signal Gen.	Generator	Raw Value	Raw Step	Physical Value	Phys Step	Comment
Dummy_pID	<input checked="" type="checkbox"/>	None	0	C	0	3276.75	
Schalter_MD1Vorne_Druck	<input checked="" type="checkbox"/>	None	0	1	0	3276.75	
Schalter_MD1Vorne_Zug	<input checked="" type="checkbox"/>	None	0	1	0	3276.75	
Schalter_MD1Vorne_Vorwahl	<input checked="" type="checkbox"/>	None	0	1	0	3276.75	
Schalter_MD1Vorne_Fehler	<input checked="" type="checkbox"/>	None	0	1	0	3276.75	
Schalter_MRolloVorne_auf	<input checked="" type="checkbox"/>	None	0	1	0	3276.75	
Schalter_MRolloVorne_zu	<input checked="" type="checkbox"/>	None	0	1	0	3276.75	
Schalter_MRolloVorne_Fehler	<input checked="" type="checkbox"/>	None	0	1	0	3276.75	
Schalter_MD3Vorne_auf	<input checked="" type="checkbox"/>	None	0	1	0	3276.75	
Schalter_MD3Vorne_zu	<input checked="" type="checkbox"/>	None	0	1	0	3276.75	
Schalter_MD3Vorne_Fehler	<input checked="" type="checkbox"/>	None	0	1	0	3276.75	
Schalter_MRolloHinten_auf	<input checked="" type="checkbox"/>	None	0	1	0	3276.75	
Schalter_MRolloHinten_zu	<input checked="" type="checkbox"/>	None	0	1	0	3276.75	
Schalter_MRolloHinten_Fehler	<input checked="" type="checkbox"/>	None	0	1	0	3276.75	