TSMaster_COM 接口_python 编程指导 V1.0

1. 目录

1. 什么情况下需要此文档?

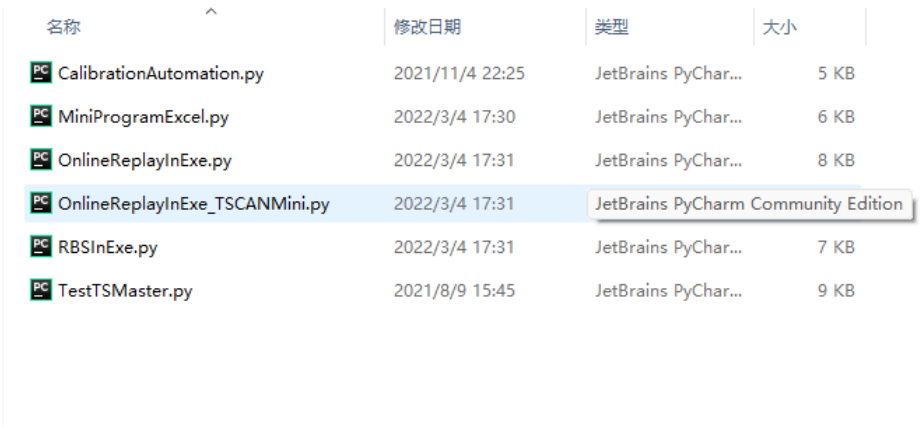
用户基于 Python 平台的编程语言，对目前市面上主流的如 Vector，TOSUN，Peak，英特佩斯等工具进行二次开发的时候，需要参考本文档，调用 com 接口来实现对设备的程序控制。

2. 使用 SDK 工程

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

C:\ProgramFiles(x86)\TOSUN\TSMaster\bin\Data\Demo\Automation\OutOfProcess\Python

进入目录后，目录包含文件如下所示：


名称
修改日期
类型
大小
CalibrationAutomation.py 2021/11/4 22:25 JetBrains PyChar... 5 KB
MiniProgramExcel.py 2022/3/4 17:30 JetBrains PyChar... 6 KB
OnlineReplayInExe.py 2022/3/4 17:31 JetBrains PyChar... 8 KB
OnlineReplayInExe_TSCANMini.py 2022/3/4 17:31 JetBrains PyCharm Community Edition
RBSInExe.py 2022/3/4 17:31 JetBrains PyChar... 7 KB
TestTSMaster.py 2021/8/9 15:45 JetBrains PyChar... 9 KB

1. 错误提示：

2. 解决办法：

3. 驱动特点

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

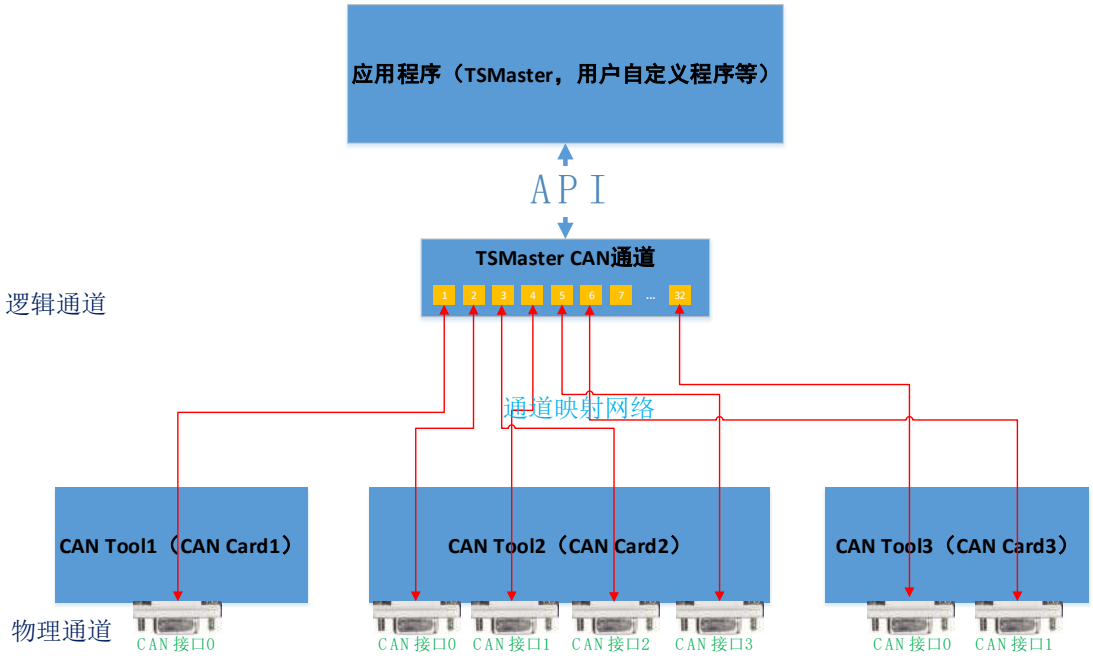


图1. TSMasterAPI 通过逻辑通道给上层提供统一的 API 接口

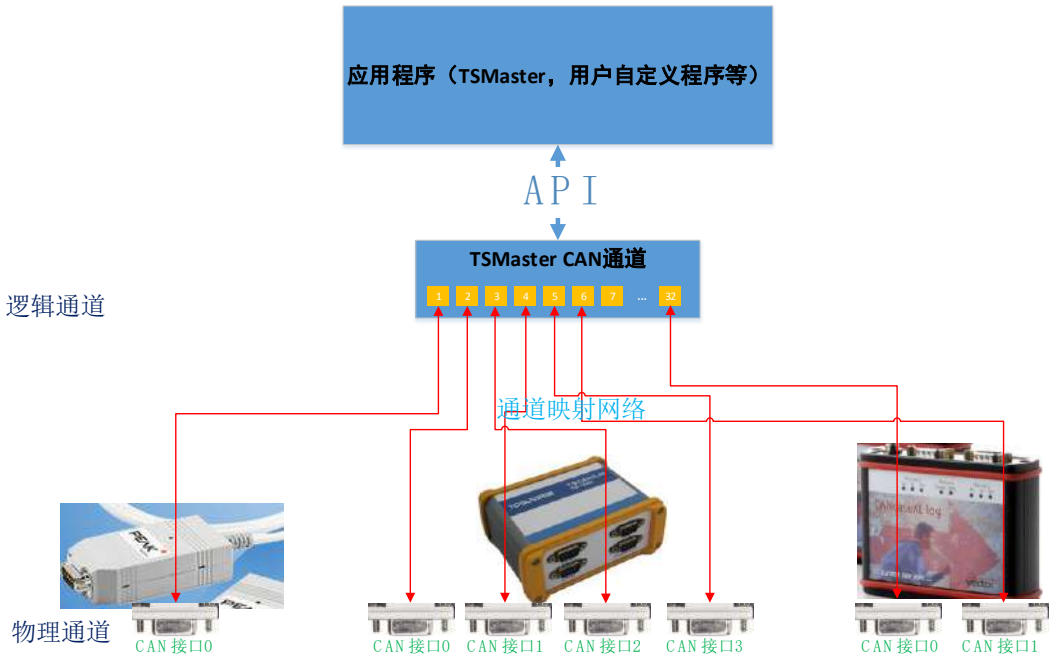


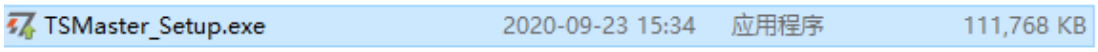
图2. 基于 TSMasterAPI 混合连接不同硬件板卡示意图

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

4. 添加 API 库文件

1. 安装驱动运行环境 TSMaster

TSMaster API 提供了对多种工具平台的支持，因此要使用 TSMaster 驱动，首先要安装 TSMaster 运行环境。该运行环境大小 100 多兆，安装文件如下：

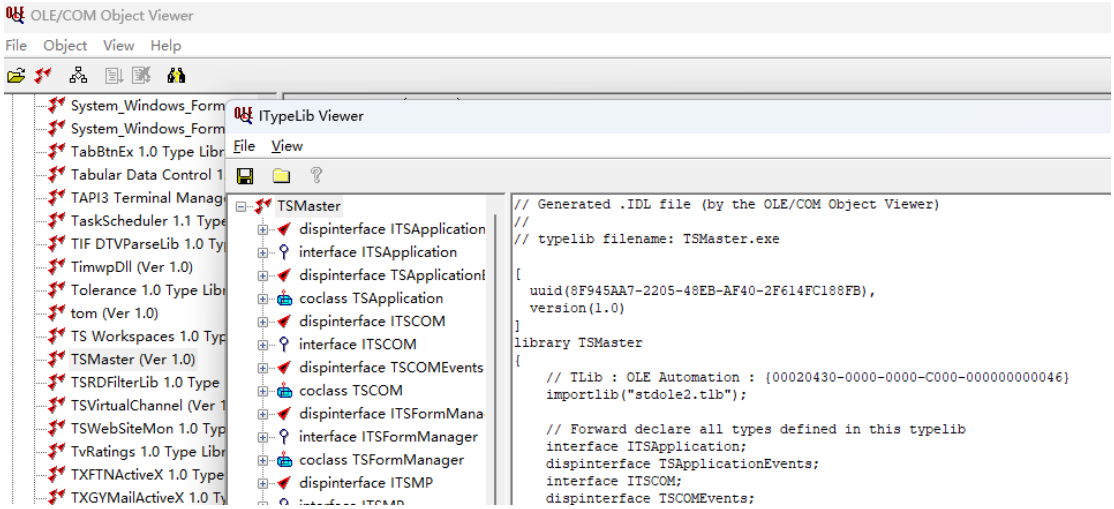


TSMaster 运行时环境为免费软件，可以直接到上海同星公司的官网上下载，下来链接为：http://www.tosun.tech/TOSUNSoftware/TSMaster_Setup.exe

下载该文件，并完成安装，然后进入下一步。

2. 添加库文件引用

安装完运行环境后，便有了运行时环境的支撑，如下图所示：



运行 oleview.exe 打开 Type Libraries，找到 Tsmaster，其中便是 TsMaster 提供的 COM 接口。

5. 数据类型定义

1. TLIBCAN：Classic CAN 数据类型

Struct 视图

```
typedef [uuid(19CEBEE3-37B4-4092-AC82-7D124EE2352C)
struct tagTCAN {

long FIdxChn;

VARIANT_BOOL FIsTX;

VARIANT_BOOL FIsRemote;

VARIANT_BOOL FIsExtendedId;

VARIANT_BOOL FIsError;

long FDLC;

long FIdentifier;

int64 FTimeUs;

SAFEARRAY(char) FDatas;
} TCAN;
```

构造函数：

用户可灵活调用。在构造函数中没有传入的参数，可以通过访问结构体对象直接修改。比如构造函数 1 没有传输数据组，但是可以在创建结构体对象过后，直接给 FData 成员赋值。

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 CAN 相关的属性，比如是否远程帧，是否扩展帧。

示例：

```
# init a TCAN record for CAN message transmission
c = win32com.client.Record("TCAN", app)
c.FIdxChn = 0
c.FIsExtendedId = 0
c.FIsRemote = False
c.FIdentifier = 0x123
c.FDLC = 8
c.FDatas = VARIANT(pythoncom.VT_ARRAY | pythoncom.VT_I1, [1, 2, 3, 4, 5, 6,
7, 8])
```

2. TLIBCANFD：CANFD 报文类型

Struct 视图

```
typedef [uuid(F8CC7BEE-63F0-4FD2-AFB1-B1C313BECE43)]
struct tagTCANFD {

long FIdxChn;

VARIANT_BOOL FIsTX;

VARIANT_BOOL FIsExtendedId;

VARIANT_BOOL FIsError;

VARIANT_BOOL FIsEDL;

VARIANT_BOOL FIsBRS;

VARIANT_BOOL FIsESI;

long FDLC;

long FIdentifier;

int64 FTimeUs;

SAFEARRAY(char) FDatas;
} TCANFD;
```

构造函数：

用户可灵活调用。在构造函数中没有传入的参数，可以通过访问结构体对象直接修改。比如构造函数 1 没有传输数据组，但是可以在创建结构体对象过后，直接给 FData 成员赋值。

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。

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。

示例：

```
# init a TCANFD record for CAN FD message transmission
cFD = win32com.client.Record("TCANFD", app)
cFD.FIdxChn = 0
cFD.FIsExtendedId = 0
cFD.FIsEDL = True
cFD.FIsBRS = True
cFD.FIsESI = False
cFD.FIdentifier = 0x345
cFD.FDLC = 15
cFD.FDatas = VARIANT(pythoncom.VT_ARRAY | pythoncom.VT_I1, [1, 2, 3, 4, 5,
6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28,
29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,45, 46, 47, 48, 49, 50,
51, 52, 53, 54, 55, 56, 57, 58,59, 60, 61, 62, 63, 64])
```

3. TLIBLIN

Struct 结构

```
typedef [uuid(F5C26CE2-3629-4BD0-A7F7-C2FAF8999736)]
struct tagTLIN {

long FIdxChn;

VARIANT_BOOL FIsTX;

VARIANT_BOOL FIsError;

long FDLC;

long FIdentifier;

long FChecksum;

int64 FTimeUs;

SAFEARRAY(char) FDatas;
} TLIN;
```

构造函数

用户可灵活调用。在构造函数中没有传入的参数，可以通过访问结构体对象直接修改。比如构造函数 1 没有传输数据组，但是可以在创建结构体对象过后，直接给 FData 成员赋值。

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。

调用示例：

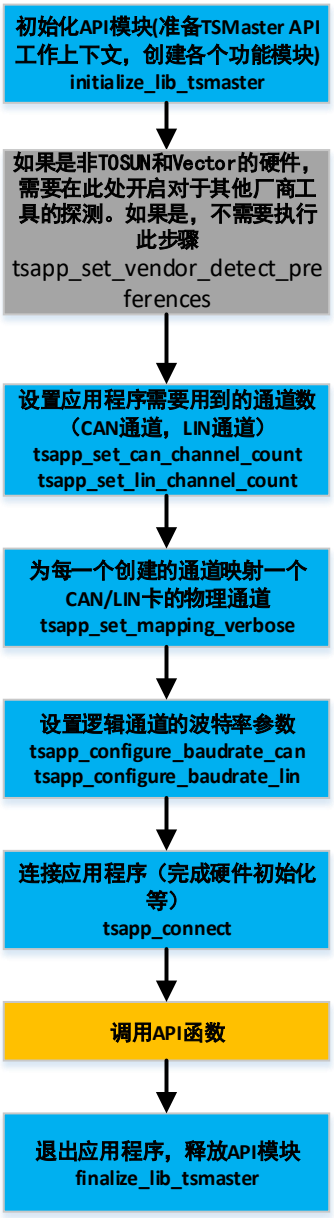
```
cLIN = win32com.client.Record("TLIN", app)
cLIN.FIdxChn = 0
cLIN.FIsTX = True
cLIN.FIsError = False
cLIN.FIdentifier = 0x3C
cLIN.FDLC = 8
cLIN.FDatas = VARIANT(pythoncom.VT_ARRAY | pythoncom.VT_I1, [1, 2, 3, 4, 5,
6, 7, 8,])
```

6. 设备初始化

1. CAN 初始化流

程图:

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



注意:

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

CAN 初始化示例代码：

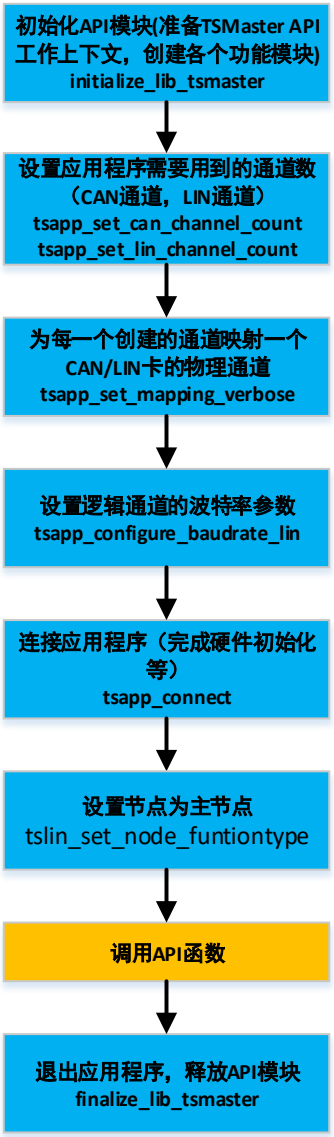
```
pythoncom.CoInitialize() # enable multithread
#打开TSMaster
app = win32com.client.Dispatch("TSMaster.TSApplication")
com = app.TSCOM() #
win32com.client.Dispatch("TSMaster.TSCOM")
formMan = app.TSFormManager()
#设置为主窗口
formMan.show_main_form()
app.set_can_channel_count(2) #设置can通道为2
app.set_lin_channel_count(0) #设置lin通道为0
r = win32com.client.Record("TTSMapping", app)
r.FAppName = APP_NAME
r.FAppChannelIndex = 0
r.FAppChannelType = 0
r.FHWIndex = 0
```

```
r.FHWDeviceType = 3
r.FHWDeviceSubType = 8  #为TC1014编号
r.FHWChannelIndex = 0
r.FHWDeviceName = "TC1014"
r.FMappingDisabled = False
app.set_mapping(r)
r.FAppChannelIndex = 1
r.FHWChannelIndex = 1
app.set_mapping(r)
app.configure_baudrate_canfd(0, 500, 2000,
constants.lfdtISOCAN, constants.lfdmNormal, True)
app.configure_baudrate_canfd(1, 500, 2000,
constants.lfdtISOCAN, constants.lfdmNormal, True)
app.connect()
#关于硬件映射编号请打开Tsmaster帮助栏的软件开发包中的
TSMasterAPI_Hardware_Map文档查找。
```

2. LIN 初始化

流程图：

LIN 通道初始化示例流程图如下：



代码示例：

下述代码为初始化为主节点的代码示例：

```
pythoncom.CoInitialize() # enable multithread
#打开TSMaster
app = win32com.client.Dispatch("TSMaster.TSApplication")
com = app.TSCOM() #
win32com.client.Dispatch("TSMaster.TSCOM")
```

```
formMan = app.TSFormManager()
#设置为主窗口
formMan.show_main_form()
app.set_can_channel_count(1) #设置can通道为2
app.set_lin_channel_count(2) #设置lin通道为0
r = win32com.client.Record("TTSMapping", app)
r.FAppName = APP_NAME
r.FAppChannelIndex = 0
r.FAppChannelType = 1
r.FHWIndex = 0
r.FHWDeviceType = 3
r.FHWDeviceSubType = 10  #为TC1026编号
r.FHWChannelIndex = 0
r.FHWDeviceName = "TC1026"
r.FMappingDisabled = False
app.set_mapping(r)
r.FAppChannelIndex = 1
r.FHWChannelIndex = 1
app.set_mapping(r)
app.configure_baudrate_lin(0, 19.2, 3)
app.connect()
```

7. CAN 报文收发

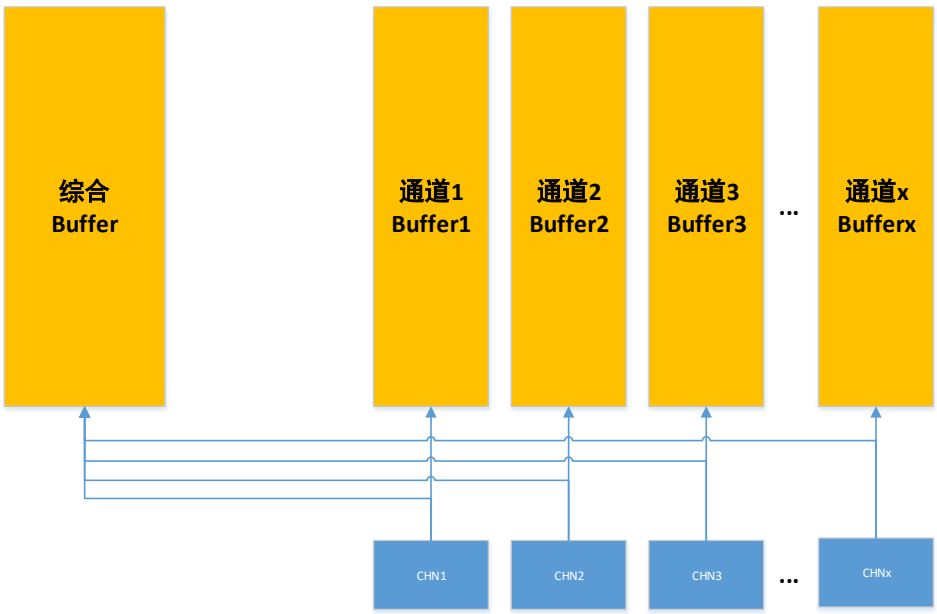
1. 报文发送

2. 报文接收

读取设备消息缓存的方式：

简介：

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



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

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

TSMaster 提供了报文读取驱动。可以选择读取指定通道的报文集合，也可以读取综合报文里面的报文集合。

fifo_receive_can_msg

- 功能：读取 Classic CAN 内部报文。
- 参数详解：

/*AIdxChnReq:请求数据通道、AIncludeTx：是否接收 TX、AIdxChnResp：回复数据通道、AIsRemote：是否是远程帧、AIsExtended：是否是扩展帧、ADLC：数据长度、AIdentifier：帧 ID、ATimestampUs：硬件时间、ADatas：帧数据*/

```
VARIANT_BOOL fifo_receive_can_msg(
        [in] long AIdxChnReq,
        [in] VARIANT_BOOL AIncludeTx,
        [out] long* AIdxChnResp,
        [out] VARIANT_BOOL* AIsRemote,
        [out] VARIANT_BOOL* AIsExtended,
        [out] long* ADLC,
        [out] long* AIdentifier,
        [out] int64* ATimestampUs,
        [out] BSTR* ADatas);
```

fifo_receive_canfd_msg

- **功能：读取 CANFD 报文**
- **参数详解：**

/*AIdxChnReq:请求数据通道、AIncludeTx：是否接收 TX、AIdxChnResp：回复数据通道、AIsRemote：是否是远程帧、AIsExtended：是否是扩展帧、AIsEDL：是否为 can、AIsBRS：速率是否可以变、ADLC：数据长度、AIdentifier：帧 ID、ATimestampUs：硬件时间、ADatas：帧数据*/

```
VARIANT_BOOL fifo_receive_canfd_msg(
        [in] long AIdxChnReq,
```

```
            [in] VARIANT_BOOL AIncludeTx,
            [out] long* AIdxChnResp,
            [out] VARIANT_BOOL* AIsRemote,
            [out] VARIANT_BOOL* AIsExtended,
            [out] VARIANT_BOOL* AIsEDL,
            [out] VARIANT_BOOL* AIsBRS,
            [out] long* ADLC,
            [out] long* AIdentifier,
            [out] int64* ATimestampUs,
            [out] BSTR* ADatas);
```

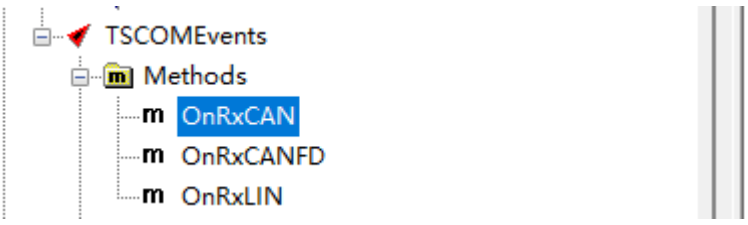
回调函数方式：

简介：

设备接收到报文过后，把报文整理成标准的 TLibCAN/TLibANFD 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：



回调函数使用

```
/// <summary>
   class TSMasterEvents:
    def OnRxCAN(self, AIdxChn, AIsTx, AIsRemote, AIsExtended, AIsError, ADLC, AIdentifier,
```

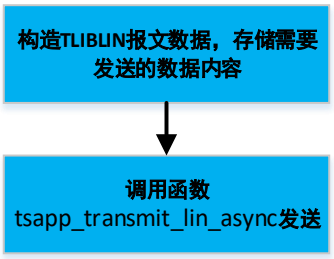
```
ATimeUs, ADatas):
        print("CAN message received:", AIdxChn, AIdentifier, ADLC, ATimeUs/1000000.0, [ADatas[0], ADatas[1], ADatas[2], ADatas[3], ADatas[4], ADatas[5], ADatas[6], ADatas[7]])
    def OnRxCANFD(self, AIdxChn, AIsTx, AIsExtended, AIsError, AIsEDL, AIsBRS, AIsESI, ADLC, AIdentifier, ATimeUs, ADatas):
        if AIsEDL:
            print("CAN FD message received:", AIdxChn, AIdentifier, ADLC, ATimeUs/1000000.0, [ADatas[0], ADatas[1], ADatas[2], ADatas[3], ADatas[4], ADatas[5], ADatas[6], ADatas[7], ADatas[8], ADatas[9], ADatas[10], ADatas[11], ADatas[12], ADatas[13], ADatas[14], ADatas[15], ADatas[16], ADatas[17], ADatas[18], ADatas[19], ADatas[20], ADatas[21], ADatas[22], ADatas[23], ADatas[24], ADatas[25], ADatas[26], ADatas[27], ADatas[28], ADatas[29], ADatas[30], ADatas[31], ADatas[32], ADatas[33], ADatas[34], ADatas[35], ADatas[36], ADatas[37], ADatas[38], ADatas[39], ADatas[40], ADatas[41], ADatas[42], ADatas[43], ADatas[44], ADatas[45], ADatas[46], ADatas[47], ADatas[48], ADatas[49], ADatas[50], ADatas[51], ADatas[52], ADatas[53], ADatas[54], ADatas[55], ADatas[56], ADatas[57], ADatas[58], ADatas[59], ADatas[60], ADatas[61], ADatas[62], ADatas[63]])
        else:
            print("CAN message received:", AIdxChn, AIdentifier, ADLC, ATimeUs/1000000.0, [ADatas[0], ADatas[1], ADatas[2], ADatas[3], ADatas[4], ADatas[5], ADatas[6], ADatas[7]])
    def OnRxLIN(self, AIdxChn, AIsTx, AIsError, ADLC, AIdentifier, AChecksum, ATimeUs, ADatas):
        print("LIN message received:")
win32com.client.WithEvents(com, TSMasterEvents)
com.enable_event_can(True)
com.enable_event_canfd(False)
com.fifo_enable_receive_fifo()
```

/// 每当设备发送/收到一帧报文，就会调用此函数。参数：ref TLibCAN 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。

8. LIN 报文收发

1. 报文发送

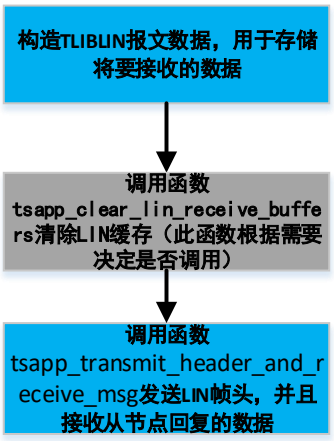
报文发送流程如下所示：



对应的代码如下：

```
cLIN = win32com.client.Record("TLIN", app)

cLIN.FIdxChn = 0

cLIN.FIsTX = True

cLIN.FIsError = False

cLIN.FIdentifier = 0x3C

cLIN.FDLC = 8

cLIN.FDatas = VARIANT(pythoncom.VT_ARRAY | pythoncom.VT_I1, [1, 2, 3, 4, 5,

6, 7, 8,])

com.transmit_lin_async(cLIN)
```

2. 报文接收




```
/*AIdxChnReq:请求数据通道、AIncludeTx：是否接收 TX、AIdxChnResp：回复数
据通道、ADLC：数据长度、AIdentifier：帧 ID、ATimestampUs：硬件时间、
ADatas：帧数据*/

VARIANT_BOOL fifo_receive_lin_msg(
             [in] long AIdxChnReq,
             [in] VARIANT_BOOL AIncludeTx,
             [out] long* AIdxChnResp,
             [out] long* ADLC,
             [out] long* AIdentifier,
             [out] int64* ATimestampUs,
             [out] BSTR* ADatas);

}
```

9. 数据库（DBC）解析

TBD

10. 接口函数介绍

1. set_current_application

函数名称	void set_current_application([in] BSTR AAppName);
功能介绍	设置当前应用程序名
调用位置	软件连接之前
输入参数	
返回值	无
示例	APP_NAME = "TSMaster" app.set_current_application(APP_NAME)

2. app.get_application_list

函数名称	BSTR get_application_list()
功能介绍	获取所有的应用程序名
调用位置	Tsmaster 连接之前
输入参数	无
返回值	无
示例	app.get_application_list()

3. set_can_channel_count

函数名称	void set_can_channel_count([in] long ACount)
功能介绍	设置 CAN 通道数量
调用位置	Tsmaster 连接之前
输入参数	整形数，最大为 32
返回值	无返回值
示例	app.set_can_channel_count(2)

4. set_can_channel_count

函数名称	void set_lin_channel_count([in] long ACount)
功能介绍	设置 CAN 通道数量
调用位置	Tsmaster 连接之前
输入参数	整形数，最大为 32
返回值	无返回值
示例	app.set_can_channel_count(0)

5. get_can_channel_count

函数名称	long get_can_channel_count();
功能介绍	获取 CAN 通道数
调用位置	

输入参数	无参
返回值	CAN 通道数
示例	app.get_can_channel_count()

6. get_can_channel_count

函数名称	**long get_can_channel_count()**
功能介绍	获取 LIN 通道数
调用位置	
输入参数	无参
返回值	CAN 通道数
示例	app.get_lin_channel_count()

7. log

函数名称	**void log([in] BSTR AMsg, [in] long ALevel);**
功能介绍	**打印信息**
调用位置	
输入参数	无参数
返回值	无返回值
示例	app.log('connecting application...', constants.LVL_HINT)

8. set_mapping

函数名称	void set_mapping([in] TTSMapping* AMapping)
功能介绍	通道映射
调用位置	Tsmaster 连接之前
输入参数	typedef [uuid(6EE71046-BB83-47A9-BC09-84A1B43C0699)] struct tagTTSMapping { BSTR FAppName; long FAppChannelIndex; TTSAppChannelType FAppChannelType; TTSBusToolDeviceType FHWDeviceType; long FHWIndex; long FHWChannelIndex; long FHWDeviceSubType; BSTR FHWDeviceName; VARIANT_BOOL FMappingDisabled; } TTSMapping;
返回值	无返回值
示例	r = win32com.client.Record("TTSMapping", app) r.FAppName = APP_NAME r.FAppChannelIndex = 0

```
r.FAppChannelType = 0
r.FHWIndex = 0
r.FHWDeviceType = 3
r.FHWDeviceSubType = 8  #为TC1014编号
r.FHWChannelIndex = 0
r.FHWDeviceName = "TC1014"
r.FMappingDisabled = False
app.set_mapping(r)
```

9. get_mapping

函数名称	**TTSMapping get_mapping([in] TTSAppChannelType AAppChannelType, [in] long AAppChannelIndex);**
功能介绍	**获取映射信息**
调用位置	在应用程序连接 CAN 工具之前，调用此函数完成硬件的绑定。
输入参数	**1:通道类型 2：通道索引**
返回值	映射信息，结构体 TTSMapping
示例	app.get_mapping(TTSAppChannelType.APP_CAN,0)

10. del_mapping

函数名称	void del_mapping([in] TTSAppChannelType AAppChannelType, [in] long AAppChannelIndex);
功能介绍	**删除指定通道的映射信息值**
调用位置	想查询应用程序指定通道当前绑定的硬件信息等内容的时候，调用此函数
输入参数	**1:通道类型 2：通道索引**
返回值	无返回值
示例	app.del_mapping(TTSAppChannelType.APP_CAN,0)

11. connect

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

12. disconnect

函数名称	**void disconnect();**
功能介绍	**断开设备连接**
调用位置	不需要使用硬件设备，调用此函数断开设备连接
输入参数	**无参数**
返回值	无返回值
示例	

13. configure_baudrate_can

函数名称	**void configure_baudrate_can(** **[in] long AIdxChn,** **[in] single ABaudrateKbps,** **[in] VARIANT_BOOL AListenOnly,** **[in] VARIANT_BOOL AInstallTermResistor120Ohm);**
功能介绍	配置 CAN 通道的波特率等硬件参数
调用位置	连接 CAN 工具之前，先调用此函数配置硬件设备参数
输入参数	/// <summary> /// 设置CANFD通道波特率等参数 /// </summary> /// <param name="AIdxChn">应用程序通道</param> /// <param name="ABaudrateKbps">波特率</param> /// <param name="AListenOnly">是否只听</param> /// <param name="AInstallTermResistor120Ohm">是否使能内部终端电阻</param>
返回值	==0:连接成功 其他：检查错误码
示例	

14. configure_baudrate_canfd

函数名称	`void configure_baudrate_canfd(`
	`[in] long AIdxChn,`
	`[in] long AArbRateKbps,`
	`[in] long ADataRateKbps,`
	`[in] TTSCANFDControllerType AControllerType,`
	`[in] TTSCANFDControllerMode AControllerMode,`
	`[in] VARIANT_BOOL AInstallTermResistor120Ohm);`

功能介绍	配置 CANFD 通道的波特率等硬件参数
调用位置	连接 CAN 工具之前，先调用此函数配置硬件设备参数
输入参数	/// <summary> /// 设置CANFD通道波特率等参数 /// </summary> /// <param name="AIdxChn">应用程序通道</param> /// <param name="AArbRateKbps">仲裁场波特率</param> /// <param name="ADataRateKbps">数据场波特率</param> /// <param name="AControllerType">控制器类型，包括： 经典CAN(lfdtCAN = 0),ISOCANFD(lfdtISOCAN = 1),NoISOCANFD(lfdtNonISOCAN = 2)</param> /// <param name="AControllerMode">控制器工作模式,包括: 正常工作模式(lfdmNormal = 0),关闭ACK模式(lfdmACKOff = 1),受限制模式(lfdmRestricted = 2)</param> /// <param name="AInstallTermResistor120Ohm">是否使能内部终端电阻</param>
返回值	无返回值
示例	app.configure_baudrate_canfd(0,500,2000,constants.lfdtISOCAN, constants.lfdmNormal, True)

15. fifo_enable_receive_fifo

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

	数。
输入参数	无
返回值	无
示例	void fifo_enable_receive_fifo()

16. fifo_disable_receive_fifo

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

17. transmit_can_async

函数名称	**`void transmit_can_async([in] PCAN ACAN);`**
功能介绍	以异步的方式发送 CAN 报文
调用位置	发送 CAN 报文
输入参数	ACAN：CAN 数据包。TLIBCAN 数据组成请查看章节：TLIBCAN：Classic CAN 数据类型。
返回值	无返回值

示例	c = win32com.client.Record("TCAN", app) c.FIdxChn = 0 c.FIsExtendedId = 0 c.FIsRemote = False c.FIdentifier = 0x123 c.FDLC = 8 c.FDatas = VARIANT(pythoncom.VT_ARRAY | pythoncom.VT_I1, [1, 2, 3, 4, 5, 6, 7, 8]) com.transmit_can_async(c)

18. transmit_can_sync

函数名称	**void transmit_can_sync(** **[in] PCAN ACAN,** **[in] long ATimeoutMs,** **[out] VARIANT_BOOL* ASuccess);**
功能介绍	发送 CAN 报文，并检测到发送成功后，才退出此函数。此函数返回成功，代表 CAN 报文一定已经成功发送到了 CAN 总线上面。
局限性	因为此函数需要设备 API 深度支持，因此，目前支持此函数接口的只有 TS 和 Vector 系列设备。
调用位置	同步发送 CAN 报文的场合
输入参数	/// <summary> /// 同步发送CAN报文 /// </summary> /// <param name="ACAN">报文数据</param> /// <param name="ATimeoutMS">同步等待的超时时间</param>
返回值	ASuccess:表示发送是否成功
示例	c = win32com.client.Record("TCAN", app) c.FIdxChn = 0 c.FIsExtendedId = 0 c.FIsRemote = False c.FIdentifier = 0x123 c.FDLC = 8 c.FDatas = VARIANT(pythoncom.VT_ARRAY | pythoncom.VT_I1, [1, 2, 3, 4, 5, 6, 7, 8]) com.transmit_can_sync(c, 20, ASuccess)

19. transmit_canfd_async

函数名称	void transmit_canfd_async([in] PCANFD ACANFD);
功能介绍	以异步的方式发送 CANFD 报文
调用位置	发送 CANFD 报文
输入参数	ACAN：CANFD 数据包。TLIBCAN 数据组成请查看章节：TLIBCANFD：CANFD 数据类型。
返回值	无返回值
示例	cFD = win32com.client.Record("TCANFD", app) cFD.FIdxChn = 0 cFD.FIsExtendedId = 0 cFD.FIsEDL = True cFD.FIsBRS = True cFD.FIsESI = False cFD.FIdentifier = 0x345 cFD.FDLC = 15 cFD.FDatas = VARIANT(pythoncom.VT_ARRAY | pythoncom.VT_I1, [1, 2, 3, 4, 5, 6, 7, 8, 9,10, 11, 12, 13, 14, 15, 16,17, 18, 19, 20, 21, 22, 23,24, 25, 26, 27, 28, 29, 30,31, 32, 33, 34, 35, 36, 37,38, 39, 40, 41, 42, 43, 44,45, 46, 47, 48, 49, 50, 51,52, 53, 54, 55, 56, 57, 58,59, 60, 61, 62, 63, 64]) com.transmit_canfd_async(cFD)

20. transmit_canfd_sync

函数名称	void transmit_canfd_sync([in] PCANFD ACANFD, [in] long ATimeoutMs, [out] VARIANT_BOOL* ASuccess);
功能介绍	发送 CANFD 报文，并检测到发送成功后，才退出此函数。此函数返回成功，代表 CANFD 报文一定已经成功发送到了 CAN 总线上面。
调用位置	发送 CANFD 报文
局限性	因为此函数需要设备 API 深度支持，因此，目前支持此函数接口的只有 TS 和 Vector 系列设备。
输入参数	ACANFD：CANFD 数据包。TLIBCANFD 数据组成请查看章节：TLIBCANFD：CANFD 数据类型。
返回值	ASuccess:表示发送是否成功

示例	cFD = win32com.client.Record("TCANFD", app) cFD.FIdxChn = 0 cFD.FIsExtendedId = 0 cFD.FIsEDL = True cFD.FIsBRS = True cFD.FIsESI = False cFD.FIdentifier = 0x345 cFD.FDLC = 15 cFD.FDatas = VARIANT(pythoncom.VT_ARRAY | pythoncom.VT_I1, [1, 2, 3, 4, 5, 6, 7, 8, 9,10, 11, 12, 13, 14, 15, 16,17, 18, 19, 20, 21, 22, 23,24, 25, 26, 27, 28, 29, 30,31, 32, 33, 34, 35, 36, 37,38, 39, 40, 41, 42, 43, 44,45, 46, 47, 48, 49, 50, 51,52, 53, 54, 55, 56, 57, 58,59, 60, 61, 62, 63, 64]) com.transmit_canfd_sync(cFD, 20, ASuccess)

21. add_cyclic_msg_can

函数名称	**void add_cyclic_msg_can(** **[in] PCAN ACAN,** **[in] single APeriodMs);**
功能介绍	增加周期发送的报文，增加完成后，TSMasterAPI 自动完成报文的周期发送。
调用位置	在需要周期性发送 CAN 报文的场合
输入参数	/// <summary> 添加周期发送报文 /// </summary> /// <param name="ACAN">CAN报文数据</param> /// <param name="APeriodMS">周期值</param>
返回值	无返回值
示例	c = win32com.client.Record("TCAN", app) c.FIdxChn = 0 c.FIsExtendedId = 0 c.FIsRemote = False c.FIdentifier = 0x123 c.FDLC = 8 c.FDatas = VARIANT(pythoncom.VT_ARRAY | pythoncom.VT_I1, [1, 2, 3, 4, 5, 6, 7, 8]) com.add_cyclic_msg_can(c, 20)

22. delete_cyclic_msg_can

函数名称	**void delete_cyclic_msg_can([in] PCAN 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	/// <summary> /// 删除周期发送报文 /// </summary> /// <param name="ACAN">需要被停止的周期报文</param> /// <returns></returns>
返回值	==0：删除成功 其他值：删除失败，查询错误码
示例	com.delete_cyclic_msg_can(c)

23. fifo_clear_can_receive_buffers

函数名称	**VARIANT_BOOL fifo_clear_can_receive_buffers([in] long AIdxChn);**
功能介绍	**清除 CAN 通道里面缓存的报文**
调用位置	**在执行交互协议之前（比如 UDS 诊断），先调用此命令把缓存中的历史数据清除掉**
输入参数	/// <summary> /// /// </summary> /// <param name="AChn">硬件通道</param> /// <returns></returns>
返回值	**是否清除成功**
示例	Com.fifo_clear_can_receive_buffers(0)

24. fifo_receive_can_msg

函数名称	VARIANT_BOOL fifo_receive_can_msg([in] long AIdxChnReq, [in] VARIANT_BOOL AIncludeTx, [out] long* AIdxChnResp, [out] VARIANT_BOOL* AIsRemote, [out] VARIANT_BOOL* AIsExtended, [out] long* ADLC, [out] long* AIdentifier, [out] int64* ATimestampUs, [out] BSTR* ADatas);
功能介绍	**读取缓存的 CAN 报文帧**
调用位置	**在需要读取报文数据包的场合**
输入参数	
返回值	是否接收成功
示例	com.fifo_receive_can_message_list(0,False,0,False,False,8,0x1,100,ADatas)

25. fifo_clear_canfd_receive_buffers

函数名称	VARIANT_BOOL fifo_clear_canfd_receive_buffers([in] long AIdxChn);
功能介绍	**清除 CANFD 通道里面缓存的报文**
调用位置	**在执行交互协议之前（比如 UDS 诊断），先调用此命令把缓存中的历史数据清除掉**
输入参数	/// <summary> /// /// </summary> /// <param name="AChn">硬件通道</param> /// <returns></returns>

返回值	**返回缓存中 CAN 报文的数量**
示例	

26. fifo_receive_canfd_msg

函数名称	**VARIANT_BOOL fifo_receive_canfd_msg(** **[in] long AIdxChnReq,** **[in] VARIANT_BOOL AIncludeTx,** **[out] long* AIdxChnResp,** **[out] VARIANT_BOOL* AIsRemote,** **[out] VARIANT_BOOL* AIsExtended,** **[out] VARIANT_BOOL* AIsEDL,** **[out] VARIANT_BOOL* AIsBRS,** **[out] long* ADLC,** **[out] long* AIdentifier,** **[out] int64* ATimestampUs,** **[out] BSTR* ADatas);**
功能介绍	**读取缓存的 CANFD 报文帧**
调用位置	在需要读取 CANFD 报文的场合
输入参数	
返回值	返回实际读取的报文数量，如果没有任何报文，则返回值为 0。如果一直无法读取报文，请检查是否通过函数 TsMasterApi.tsapp_enable_receive_fifo();开启了驱动内部 Buffer
示例	com.fifo_receive_canfd_msg(0,False,0,False,False,False,15,100,ADatas)

27. enable_event_can

函数名称	**HRESULT enable_event_can([in] VARIANT_BOOL AEnable);**
功能介绍	注册 CAN 数据包接收回调函数

调用位置	如果用户想基于接收回调机制来处理接收的报文，在连接工具成功过后，可以调用此函数注册回调函数。
输入参数	Bool,是否启动
返回值	
示例	com.enable_event_can(True)

28. transmit_lin_async

函数名称	**HRESULT transmit_lin_async([in] PLIN ALIN);**
功能介绍	异步发送 LIN 报文
调用位置	在需要发送 LIN 报文的场合
输入参数	/// <summary> /// 异步发送LIN报文 /// </summary> /// <param name="ALIN">LIN报文数据</param> /// <returns></returns>
返回值	==0：发送成功 其他值：发送失败
示例	cLIN = win32com.client.Record("TLIN", app) cLIN.FIdxChn = 0 cLIN.FIsTX = True cLIN.FIsError = False cLIN.FIdentifier = 0x3C cLIN.FDLC = 8 cLIN.FDatas = VARIANT(pythoncom.VT_ARRAY | pythoncom.VT_I1, [1, 2, 3, 4, 5, 6, 7, 8,]) com.transmit_lin_async(cLIN)

29. fifo_clear_lin_receive_buffers

函数名称	**HRESULT fifo_clear_lin_receive_buffers(** **[in] long AIdxChn,** **[out, retval] VARIANT_BOOL* ASuccess);**
功能介绍	清除 LIN 模块的接收 Buffer 缓存
调用位置	在准备开始一次发送和接收之前，调用此函数清除驱动中的 LIN 缓存中的数据，确保接收到的数据是最新的
输入参数	/// <summary> /// 清除LIN通道的缓存数据 /// </summary> /// <param name="AChn">需要清除缓存的LIN通道编号</param> /// <returns></returns>
返回值	ASuccess:表示是否清除成功
示例	com.fifo_clear_lin_receive_buffers(APP_CHANNEL.CHN1,ASuccess);

30. fifo_receive_lin_message_list

函数名称	**HRESULT fifo_receive_lin_msg(** **[in] long AIdxChnReq,** **[in] VARIANT_BOOL AIncludeTx,** **[out] long* AIdxChnResp,** **[out] long* ADLC,** **[out] long* AIdentifier,** **[out] int64* ATimestampUs,** **[out] BSTR* ADatas,** **[out, retval] VARIANT_BOOL* ASuccess);**
功能介绍	**读取缓存的 LIN 报文帧**
调用位置	**在需要读取 LIN 报文数据包的场合**
输入参数	
返回值	ASuccess:是否接受成功
示例	com.fifo_receive_lin_msg(0,False,0,8,0x3d,100,ADatas, ASuccess)

31. start_logging

函数名称	HRESULT start_logging([in] BSTR AFileName);
功能介绍	启动报文记录，记录文件格式未 blf 格式
调用位置	需要记录整个运行过程中总线上的报文通讯
输入参数	/// <summary> /// <summary> /// 启动报文记录，记录文件格式未blf格式 /// </summary> /// <param name="AFileName">记录文件名（后缀为.blf），可以为空字符串“”</param> /// <returns>==0:函数执行成功；其他：错误码</returns> 注意:如果 AFileName 传输为空字符串“”,则数据文件默认存储在 TSMaster 的安装路径下面。AFile 数据格式路径不能出错，如果路径是无效的，创建该数据文件无效，会触发模块内部异常。
返回值	==0：执行成功 其他值：查询错误码
示例	com.start_logging(“c:\\1.blf”)

32. stop_logging

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

33. unload_can_dbs

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

34. unload_can_db

函数名称	**void unload_can_db([in] long AId);**
功能介绍	卸载指定编号的数据库文件
调用位置	该函数必须在应用处于断开状态（执行 tsapp_connect 之前或者执行了 tsapp_disconnect 之后）调用。当应用处于连接状态的时候，不能删除，加载数据库文件。
输入参数	无
返回值	
示例	

35. load_can_db

函数名称	**void load_can_db(** **[in] BSTR ADBCFile,** **[in] BSTR ASupportedChannels,** **[out] long* AId);**

功能介绍	把数据库加载到指定的通道上，并获取加载数据库的编号
调用位置	该函数必须在应用处于断开状态（执行 tsapp_connect 之前或者执行了 tsapp_disconnect 之后）调用。当应用处于连接状态的时候，不能删除，加载数据库文件。
输入参数	/// <param name="ADBCAbsolutePath">ADBCAbsolutePath:DBC 绝对路径</param> /// <param name="ABindChannels">绑定的通道数组</param> /// <param name="AId">加载数据库成功过后，返回系统为该数据库分配的唯一编号</param>
返回值	
示例	

36. set_signal_value_can

函数名称	**void set_signal_value_can(** **[in] PCAN AInCAN,** **[in] BSTR AMsgName,** **[in] BSTR ASgnName,** **[in] double AValue,** **[out] SAFEARRAY(char)* ADatas);**
功能介绍	根据 Message 名称，信号名称，把信号值更新到 CAN 报文中。
调用位置	加载数据库过后，任意位置（应用连接状态，断开状态）都可以调用此函数。
输入参数	/// <param name="ACAN">原始 CAN 报文(引用)</param> /// <param name="AMsgName">报文名称</param> /// <param name="ASgnName">信号名称</param> /// <param name="AValue">信号值</param>
返回值	==0：执行成功 其他值：查询错误码
示例	msgName = "message1" singalName = "signalName" c = win32com.client.Record("TCAN", app) value = 0 ITSDB.set_signal_value_canfd(ACAN, msgName, singalName, value)

37. get_signal_value_can

函数名称	**void get_signal_value_can(** **[in] PCAN AInCAN,** **[in] BSTR AMsgName,** **[in] BSTR ASgnName,** **[out] double* AValue);**
功能介绍	根据 Message 名称，信号名称，从 CAN 报文中读取信号值
调用位置	加载数据库过后，任意位置（应用连接状态，断开状态）都可以调用此函数。
输入参数	/// <param name="ACAN">原始 CAN 报文(引用)</param> /// <param name="AMsgName">报文名称</param> /// <param name="ASgnName">信号名称</param> /// <param name="AValue">信号值(引用类型)</param>
返回值	
示例	msgName = "message1" singalName = "signalName" c = win32com.client.Record("TCAN", app) value = 0 ITSDB.get_signal_value_canfd(ACAN, msgName, singalName, value)

38. tsdb_set_signal_value_canfd

函数名称	**void set_signal_value_canfd(** **[in] PCANFD AInCANFD,** **[in] BSTR AMsgName,** **[in] BSTR ASgnName,** **[in] double AValue,** **[out] SAFEARRAY(char)* ADatas);**
功能介绍	根据 Message 名称，信号名称，从 CANFD 报文中读取信号值
调用位置	加载数据库过后，任意位置（应用连接状态，断开状态）都可以调用此函数。
输入参数	/// <param name="ACANfd">CANFD 报文（引用）</param> /// <param name="AMsgName">报文名称</param> /// <param name="ASgnName">信号名称</param> /// <param name="AValue">信号值</param>
返回值	
示例	

39. get_signal_value_canfd

函数名称	**void get_signal_value_canfd(** **[in] PCANFD AInCANFD,** **[in] BSTR AMsgName,** **[in] BSTR ASgnName,** **[out] double* AValue);**
功能介绍	根据 Message 名称，信号名称，把信号值更新到 CANFD 报文中。
调用位置	加载数据库过后，任意位置（应用连接状态，断开状态）都可以调用此函数。
输入参数	/// <param name="ACANfd">CANFD 报文（引用）</param> /// <param name="AMsgName">报文名称</param> /// <param name="ASgnName">信号名称</param> /// <param name="AValue">信号值（引用）</param>
返回值	
示例	

11. 附件

1. 示例程序

2. 错误码编码信息：

```
ERR_CODE_LIST: array [0..ERR_CODE_COUNT-1] of string = (
    'OK',                                    // IDX_ERR_OK                          0
    'Index out of range',                    // IDX_ERR_IDX_OUT_OF_RANGE            1
    'Connect failed',                        // IDX_ERR_CONNECT_FAILED              2
    'Device not found',                      // IDX_ERR_DEV_NOT_FOUND               3
    'Read status timed out',                 // IDX_ERR_READ_STATUS_TIMEDOUT        44
    'Callback already exists',               // IDX_ERR_CALLBACK_ALREADY_EXISTS     45
    'Callback not exists',                   // IDX_ERR_CALLBACK_NOT_EXISTS         46
    'File corrupted or not recognized',      // IDX_ERR_FILE_INVALID                = 47; // database file
corrupted or not recognized
    'Database unique id not found',          // IDX_ERR_DB_ID_NOT_FOUND             = 48; // database
unique id not found
```

```
    'Software API parameter invalid',          // IDX_ERR_SW_API_PAEAMETER_INVALID    = 49; // software api parameter invalid
    'Software API generic timed out',          // IDX_ERR_SW_API_GENERIC_TIMEOUT      = 50; // software api generic timed out
    'Software API set hw config. failed',      // IDX_ERR_SW_API_SET_CONF_FAILED      = 51; // software api set hw conf failed
    'Index out of bounds',                     // IDX_ERR_SW_API_INDEX_OUT_OF_BOUNDS = 52; // index out of bounds
    'RX wait timed out',                       // IDX_ERR_SW_API_WAIT_TIMEOUT         = 53; // rx wait timed out
    'Get I/O failed',                          // IDX_ERR_SW_API_GET_IO_FAILED        = 54; // get io failed
    'Set I/O failed',                          // IDX_ERR_SW_API_SET_IO_FAILED        = 55; // set io failed
    'An active replay is already running',     // IDX_ERR_SW_API_REPLAY_ON_GOING      = 56; // a replay is already on goning
    'Instance not exists',                     // IDX_ERR_SW_API_INSTANCE_NOT_EXISTS = 57; // instance not exists
    'CAN message transmit failed',             // IDX_ERR_HW_CAN_TRANSMIT_FAILED      = 58; // can transmit frame failed
    'No response from hardware',               // IDX_ERR_HW_NO_RESPONSE              = 59; // hardware no response to pc
    'CAN message not found',                   // IDX_ERR_SW_CAN_MSG_NOT_FOUND        = 60; // can cyclic message id not found
    'User CAN receive buffer empty',           // IDX_ERR_SW_CAN_RECV_BUFFER_EMPTY    = 61; // user can recv message buffer empty
    'CAN total receive count <> desired count',// IDX_ERR_SW_CAN_RECV_PARTIAL_READ    = 62; // total read count <> desired read count
    'LIN config failed',                       // IDX_ERR_SW_API_LINCONFIG_FAILED     = 63;
    'LIN frame number out of range',           // IDX_ERR_SW_API_FRAMENUM_OUTOFRANGE = 64;
    'LDF config failed',                       // IDX_ERR_SW_API_LDFCONFIG_FAILED     = 65;
    'LDF config cmd error',                    // IDX_ERR_SW_API_LDFCONFIG_CMDERR     = 66;
    'TSMaster envrionment not ready',          // IDX_ERR_SW_ENV_NOT_READY            = 67; // encryption random number not retrieved
    'reserved failed',                         // IDX_ERR_RESERVED_FAILED             = 68;
    'XL driver error',                         // IDX_ERR_XL_ERROR                    = 69;
    'index out of range',                      //   IDX_ERR_SEC_INDEX_OUTOFRANGE      = 70;
    'string length out of range',              //   IDX_SEC_ERR_STRINGLENGTH_OUTFOF_RANGE   = 71;
    'key is not initialized',                  //   IDX_SEC_ERR_KEY_IS_NOT_INITIALIZATION   = 72;
    'key is wrong',                            //   IDX_SEC_ERR_KEY_IS_WRONG          = 73;
    'write not permitted',                     //   IDX_SEC_ERR_NOT_PERMIT_WRITE      = 74;
    '16 bytes multiple',                       //   IDX_SEC_ERR_16BYTES_MULTIPLE      = 75;
    'LIN channel out of range',                // IDX_ERR_LIN_CHN_OUTOF_RANGE         = 76;
    'DLL not ready',                           // IDX_ERR_DLL_NOT_READY               = 77;
    'Feature not supported',                   //   IDX_ERR_FATURE_NOT_SUPPORTED      = 78;
    'common service error',                    // IDX_ERR_COMMON_SERV_ERROR           = 79;
    'read parameter overflow',                 // IDX_ERR_READ_PARA_OVERFLOW          = 80;
```

```
'Invalid application channel mapping',        // IDX_ERR_INVALID_CHANNEL_MAPPING
'libTSMaster generic operation failed',       // IDX_ERR_TSLIB_GENERIC_OPERATION_FAILED = 82;
'item already exists',                        // IDX_ERR_TSLIB_ITEM_ALREADY_EXISTS      = 83;
'item not found',                             // IDX_ERR_TSLIB_ITEM_NOT_FOUND           = 84;
'logical channel invalid',                    // IDX_ERR_TSLIB_LOGICAL_CHANNEL_INVALID  = 85;
'file not exists',                            // IDX_ERR_FILE_NOT_EXISTS                = 86;
'no init access, cannot set baudrate',        // IDX_ERR_NO_INIT_ACCESS                 = 87;
'the channel is inactive',                    // IDX_ERR_CHN_NOT_ACTIVE                 = 88;
'the channel is not created',                 // IDX_ERR_CHN_Not_Created                = 89;
'length of the appname is out of range',      // IDX_ERR_APPNAME_LENGTH_OUT_OF_RANGE    = 90;
'project is modified',                        // IDX_ERR_PROJECT_IS_MODIFIED            = 91;
'signal not found in database',               // IDX_ERR_SIGNAL_NOT_FOUND_IN_DB         = 92;
'message not found in database',              // IDX_ERR_MESSAGE_NOT_FOUND_IN_DB        = 93;
'TSMaster is not installed',                  // IDX_ERR_TSMASTER_IS_NOT_INSTALLED      = 94;
'Library load failed',                        // IDX_ERR_LIB_LOAD_FAILED                = 95;
'Library function not found',                 // IDX_ERR_LIB_FUNCTION_NOT_FOUND         = 96;
'Library not initialized',                    // IDX_ERR_LIB_NOT_INITIALIZED            = 97;
'PCAN generic operation error',               // IDX_ERR_PCAN_GENRIC_ERROR              = 98;
'Kvaser generic operation error',             // IDX_ERR_KVASER_GENERIC_ERROR           = 99;
'ZLG generic operation error',                // IDX_ERR_ZLG_GENERIC_ERROR              = 100;
'ICS generic operation error',                // IDX_ERR_ICS_GENERIC_ERROR              = 101;
'TC1005 generic operation error',             // IDX_ERR_TC1005_GENERIC_ERROR           = 102;
'System variable not found',                  // IDX_ERR_SYSTEM_VAR_NOT_FOUND           = 103;
'Incorrect system variable type',             // IDX_ERR_INCORRECT_SYSTEM_VAR_TYPE      = 104;
'Message not existing, update failed',        // IDX_ERR_CYCLIC_MSG_NOT_EXIST           = 105;
'Specified baudrate not available'            // IDX_ERR_BAUD_NOT_AVAIL                 = 106;
  'Device does not support sync. transmit',   // IDX_ERR_DEV_NOT_SUPPORT_SYNC_SEND      = 107;
'Wait time not satisfied',                    // IDX_ERR_MP_WAIT_TIME_NOT_SATISFIED     = 108;
'Cannot operate while app is connected',      // IDX_ERR_CANNOT_OPERATE_WHILE_CONNECTED = 109;
'Create file failed',                         // IDX_ERR_CREATE_FILE_FAILED             = 110;
'Execute python failed',                      // IDX_ERR_PYTHON_EXECUTE_FAILED          = 111;
'Current multiplexed signal is not active',// IDX_ERR_SIGNAL_MULTIPLEXED_NOT_ACTIVE   = 112;
'Get handle by logic channel failed',         // IDX_ERR_GET_HANDLE_BY_CHANNEL_FAILED   = 113;
'Cannot operate while application is ' +
'connected, please stop application first',// IDX_ERR_CANNOT_OPERATE_WHILE_APP_CONN   = 114;
'File load failed',                           // IDX_ERR_FILE_LOAD_FAILED               = 115;
'Read LIN Data Failed',                       // IDX_ERR_READ_LINDATA_FAILED            = 116;
'FIFO not enabled'                            // IDX_ERR_FIFO_NOT_ENABLED               = 117;
```

12. 常见异常解答

1. 调用 API 开发的程序无法正常打开 CAN 驱动

- **症状描述：**

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

- **原因分析：**

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

- **解决办法：**

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

2. 调用 API，总是执行不成功，返回错误码 0x114

- **症状描述：**

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

- **原因分析：**

这是因为有的 API 函数必须在应用处于断开状态（Application 处于断开）的时候才能

够执行。比如数据库的加载，如果应用程序已经处于连接状态，数据库引擎，包括过滤，信号解析这些模块都运行起来了，此时就不允许用户进行加载卸载这些操作了。

- **解决办法：**

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