libTOSUN python 包使用说明

需要的依赖库 python-can 在使用 libTSCAN.dll 对同星硬件进行二次开发时，基于以前客户在使用 python 开发时，是基于 python-can 这个框架，因此 libTOSUN 包同时集成与 python-can 中。如果想使用 udsoncan 库，可以替换压缩包中 connections.py 文件至 udsoncan 库中的文件。

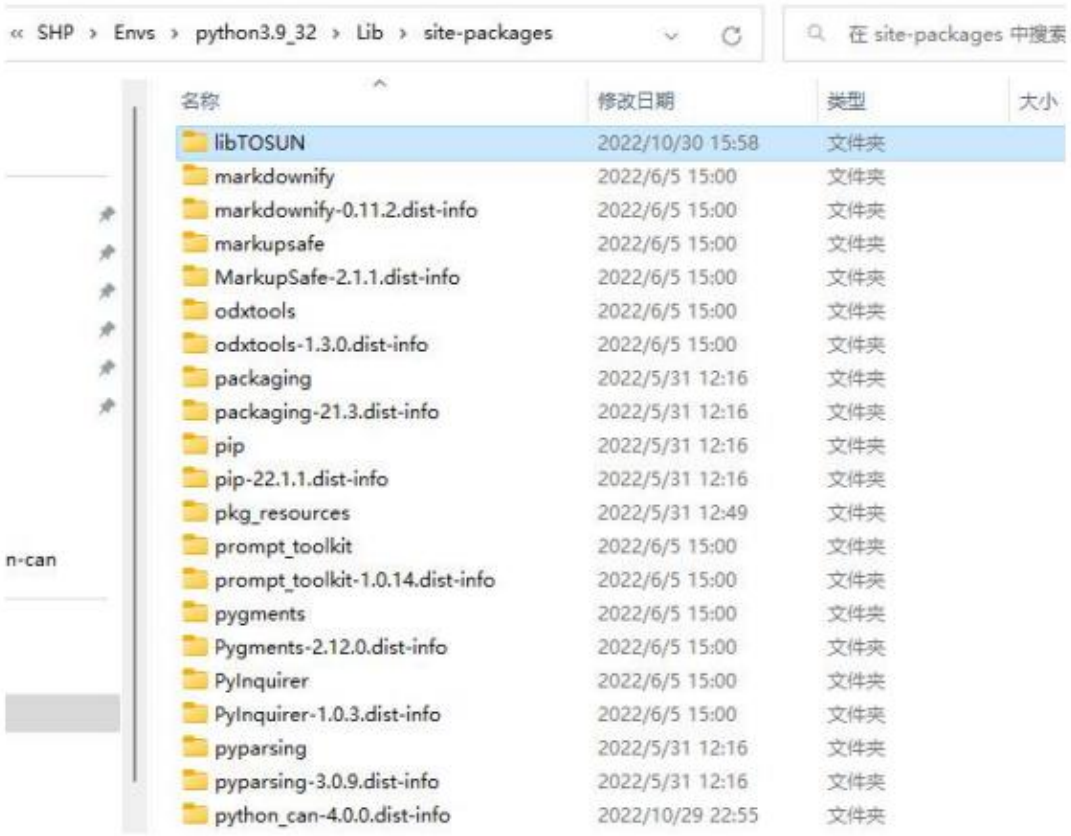
Python 环境 ：python3.7 以上 python-can 4.0.0 (python-can 4.0.0 can 属性更全更完善) 安装 pip install python-can、 pip install udsoncan(如果不使用，可以不安装）

系统环境：windows linux 同一套代码可适配两个系统

一. 库本身使用

1.python 环境配置

libTOSUN 库本身使用： 1、配置将 libTOSUN 文件夹和 can 文件夹放置在 python 下的 site-packages 文件夹里如下所示：



2.使用该库：

在 python 工程文件中 from libTOSUN.libTOSUN import * 或者 from libTOSUN import libTOSUN 含义就是导入包中 中的 libTOSUN.py 中的所有内容 接下来就是其使用： 主要依赖 TSMasterDevice 类

2.1 连接硬件以及配置

只需要在新建 TSMasterDevice 对象是，进行传递参数即可，如下图为可传

递的参数

```
def __init__(self, configs: [dict], is_recv_error: bool = False, hwserial: bytes = b'',
             is_include_tx: bool = False,
             is_start_recv: bool = False,
             dbc: str = '',
             filter:dict={}):
```

is_recv_error:是否接受错误帧，不填时默认为不接收错误帧(此处针对接受函数，不针对总线 数据)

is_include_tx:接受函数是否包含自己发送的报文，默认不包含

hwserial：硬件序列号，为空时，会自动连接硬件，如果只有一个硬件时，可不填写，如果多个硬件，需要区分时，需要填写序列号；获取序列号可通过 libTOSUN.py 中 tscan_get_device_info 获取

```
def tscan_get_device_info(ADeviceCount: c_uint64):
    AFManufacturer = POINTER(POINTER(c_char))()

    AFProduct = POINTER(POINTER(c_char))()
    AFSerial = POINTER(POINTER(c_char))()
    r = dll.tscan_get_device_info(ADeviceCount, byref(AFManufacturer), byref(AFProduct), byref(AFSerial))
    if r == 0:
        FManufacturer = string_at(AFManufacturer).decode("utf8")
        FProduct = string_at(AFProduct).decode("utf8")
        FSerial = string_at(AFSerial).decode("utf8")
    else:
        print("[illegible]")
        return 0, 0, 0
    return FManufacturer, FProduct, FSerial
```

configs:表示对硬件的设置，内容如下图：

```
self.channel_list.append(congfig['FChannel'] if 'FChannel' in congfig else index)

self.Rate_baudrate.append(congfig['rate_baudrate'] if 'rate_baudrate' in congfig else 500)

self.data_baudrate.append(congfig['data_baudrate'] if 'data_baudrate' in congfig else 2000)

self.enable_120hm.append(congfig['enable_120hm'] if 'enable_120hm' in congfig else True)
```

主要包含 4 个： FChannel：对当前硬件的哪个通道进行设置，比如 TC1016 存在 4 个 canfd 通道，此 处可选[0,3]对应[1,4]通道； rata_baudrate：仲裁段波特率 data_baudrate: 数据段波特率 enable_120hm:是否激活终端电阻并且 congfig 是一个列表字典，意味着可以同时对多个通道进行设置： 如下所示

```
configs = [{'FChannel': 0, 'rate_baudrate': 500, 'data_baudrate': 2000, 'enable_120hm': True, 'is_fd': True},
           {'FChannel': 1, 'rate_baudrate': 500, 'data_baudrate': 2000, 'enable_120hm': True, 'is_fd': True},
           {'FChannel': 2, 'rate_baudrate': 500, 'data_baudrate': 2000, 'enable_120hm': True, 'is_fd': True},
           {'FChannel': 3, 'rate_baudrate': 500, 'data_baudrate': 2000, 'enable_120hm': True, 'is_fd': True}]
```

4 发送报文：

4.1 创建对象：

1.CAN 报 文 ： Msg= TLIBCAN(FIdxChn=0, FDLC=8, FIdentifier=0x1, FProperties=1, FData=[0,0,0,0,0,0,0,0])

2. CANFD 报文 Msg= TLIBCANFD(FIdxChn=0, FDLC=8, FIdentifier=0x1, FProperties=1, FFDProperties=1,

FData=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0])

3.python-can_message:msg=can.Message(arbitration_id=0x01, is_extended_id=False, channel=0, dlc=9, data=[0x01, 0x02, 0x01, 0x02, 0x01, 0x02, 0x01, 0x02,], is_rx=False, is_fd=True, bitrate_switch=True)

4.2 发送报文：

```
def send_msg(self, msg, timeout: Optional[float] = 0.1, sync: bool = False, is_cyclic: bool = False):
```

说明：

当 is_cyclic 为 True 时:表示该报文为周期发送,此时 sync 参数无意义, timeout 为周期;

当 is_cyclic 为 False 时：表示该报文为单帧发送，此时 sync 参数为 True，timeout 为延时参数;

当 is_cyclic 为 False 时：表示该报文为单帧发送，此时 sync 参数为 False，timeout 无意义;

4.1 中创建的所有报文类型，直接调用该函数即可发送。

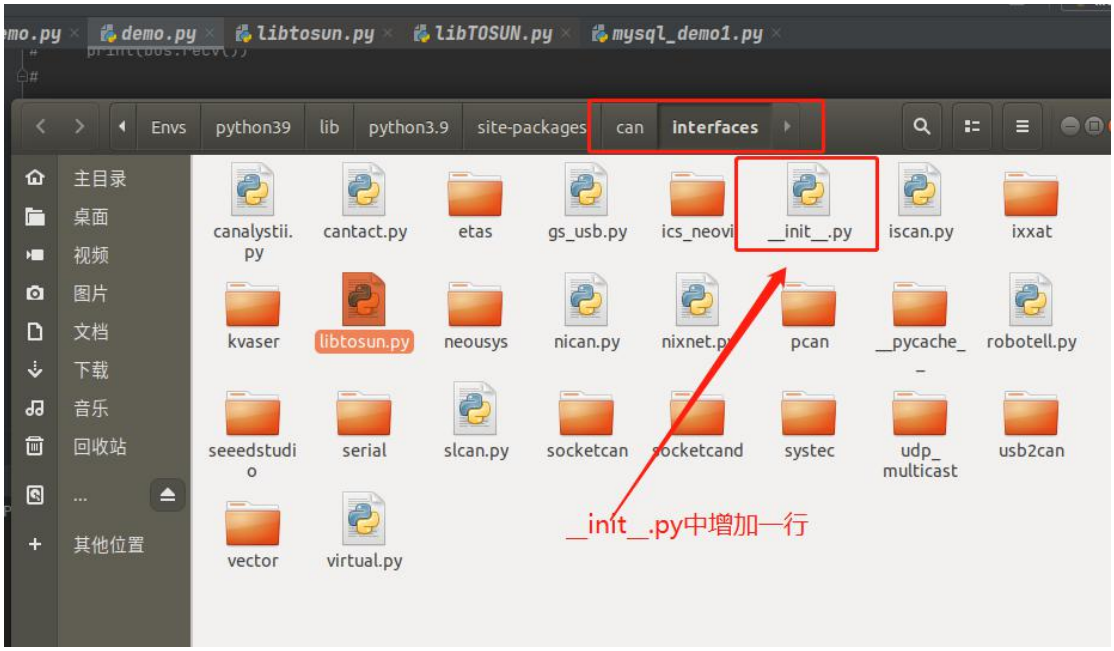
5 接收报文

```
def recv(self, timeout: Optional[float] = 0.1) -> Message:
```

返回的类型为 python-can.Message 类型即 hwHandle.recv()即可查看接收报文，当前如果仅仅查看接收报文，上连接硬件时，将 is_recv_error，is_include_tx 都设置为 False。

二、python-can 使用

1、配置 python-can

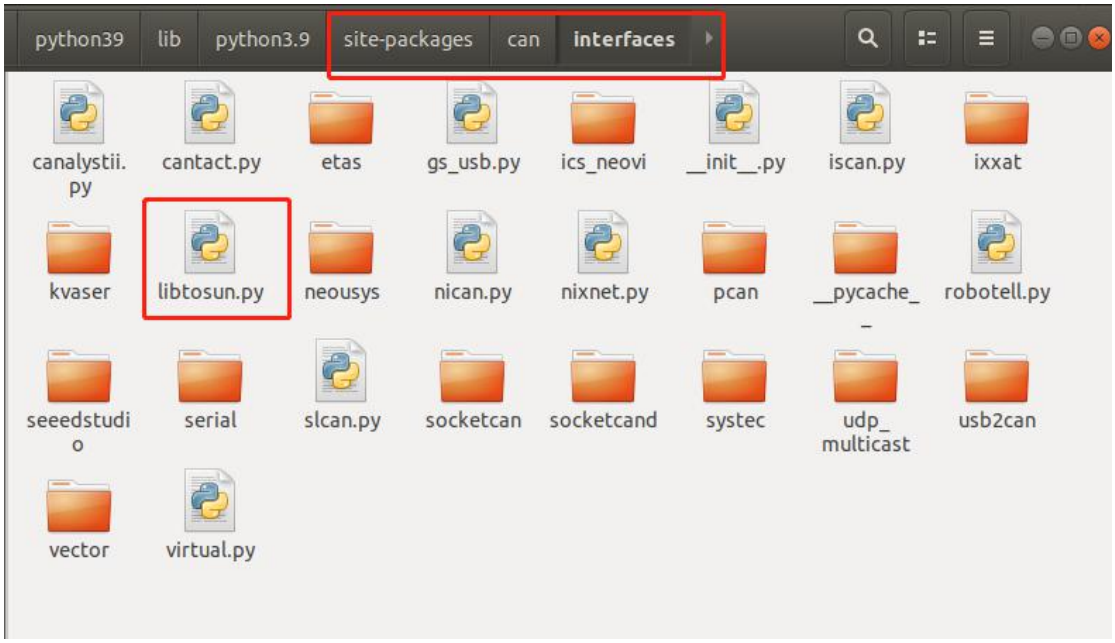


"libtosun":("can.interfaces.libtosun","libtosunBus"),

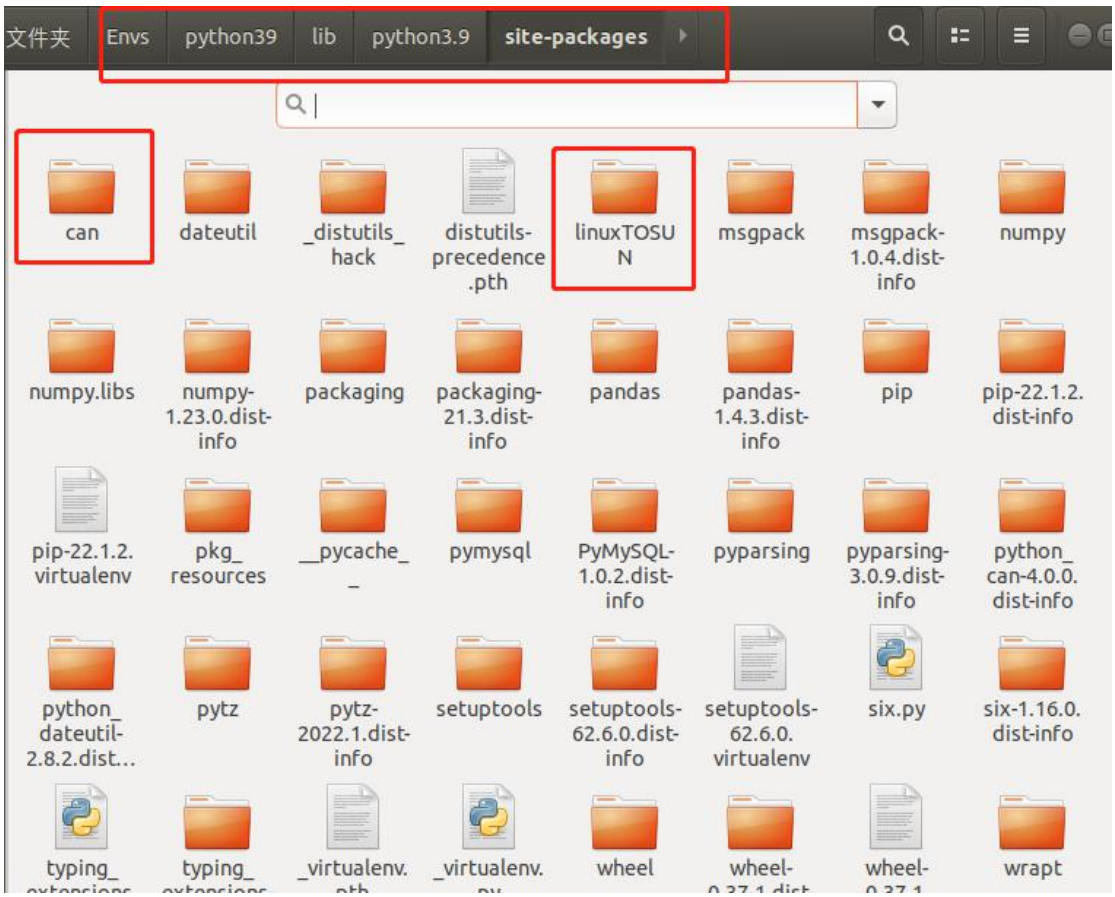
```
Demo.py × demo.py × __init__.py × libtosun.py × libTOSUN.py × mysql_demo1.py ×
"""
Interfaces contain low level implementations that interact with CAN hardware.
"""

# interface_name => (module, classname)
BACKENDS = {
    "kvaser": ("can.interfaces.kvaser", "KvaserBus"),
    "socketcan": ("can.interfaces.socketcan", "SocketcanBus"),
    "serial": ("can.interfaces.serial.serial_can", "SerialBus"),
    "pcan": ("can.interfaces.pcan", "PcanBus"),
    "usb2can": ("can.interfaces.usb2can", "Usb2canBus"),
    "ixxat": ("can.interfaces.ixxat", "IXXATBus"),
    "nican": ("can.interfaces.nican", "NicanBus"),
    "iscan": ("can.interfaces.iscan", "IscanBus"),
    "virtual": ("can.interfaces.virtual", "VirtualBus"),
    "udp_multicast": ("can.interfaces.udp_multicast", "UdpMulticastBus"),
    "neovi": ("can.interfaces.ics_neovi", "NeoViBus"),
    "vector": ("can.interfaces.vector", "VectorBus"),
    "slcan": ("can.interfaces.slcan", "slcanBus"),
    "robotell": ("can.interfaces.robotell", "robotellBus"),
    "canalystii": ("can.interfaces.canalystii", "CANalystIIBus"),
    "systec": ("can.interfaces.systec", "UcanBus"),
    "seeedstudio": ("can.interfaces.seeedstudio", "SeeedBus"),
    "cantact": ("can.interfaces.cantact", "CantactBus"),
    "gs_usb": ("can.interfaces.gs_usb", "GsUsbBus"),
    "nixnet": ("can.interfaces.nixnet", "NiXNETcanBus"),
    "neousys": ("can.interfaces.neousys", "NeousysBus"),
    "etas": ("can.interfaces.etas", "EtasBus"),
    "socketcand": ("can.interfaces.socketcand", "SocketCanDaemonBus"),
    "libtosun":("can.interfaces.libtosun","libtosunBus"),
}
```

将 libtosun.py 放到 can->interfaces 文件夹下



将文件夹 linuxTOSUN 放在 site-packages 文件夹下即 can 同级目录



使用与直接使用 libTOSUN 基本一致，参考代码见附录 2：

附录 1：

```
from libTOSUN.libTOSUN import *
configs = [{'FChannel': 0, 'rate_baudrate': 500,'data_baudrate': 2000, 'enable_120hm': True, 'is_fd':
True},     {'FChannel': 1, 'rate_baudrate': 500,'data_baudrate': 2000, 'enable_120hm': True, 'is_fd':
True},     {'FChannel': 2, 'rate_baudrate': 500,'data_baudrate': 2000, 'enable_120hm': True, 'is_fd':
True},     {'FChannel': 3, 'rate_baudrate': 500,'data_baudrate': 2000, 'enable_120hm': True, 'is_fd': True}]
hwhandle = TSMasterDevice(configs=configs,is_recv_error=True, is_include_tx=True, hwserial=b"")
msg = TLIBCAN(FIdxChn=0, FDLC=8, FIdentifier=0X1, FProperties=1, FData=[1, 2, 3, 4, 5, 6, 7, 8])
hwhandle.send_msg(msg)
pDiagModuleIndex = c_int32(0) # 为 c 类型 传入的为指针，可以随意赋值，运行下方函数后，会对其赋值
hwhandle.tsdiag_can_create(pDiagModuleIndex, 0, 0, 8, 0x123, True, 0X456, True, 0X789, True)
# r 为函数执行返回值，为 0 表示执行成功
# 非 0 可以使用 hwhandle.tscan_get_error_description(r)获取错误信息
r, respond_data = hwhandle.tstp_can_request_and_get_response(pDiagModuleIndex, [0x10, 0x02])
print(list(respond_data))
hwhandle.shut_down()
```

附录 2：

```python
import can
from ctypes import *
configs = [{'FChannel': 0, 'rate_baudrate': 500,
'data_baudrate': 2000, 'enable_120hm': True, 'is_fd': True}, {'FChannel': 1, 'rate_baudrate': 500,
'data_baudrate': 2000, 'enable_120hm': True, 'is_fd': True}, {'FChannel': 2, 'rate_baudrate': 500,
'data_baudrate': 2000, 'enable_120hm': True, 'is_fd': True}, {'FChannel': 3, 'rate_baudrate': 500,
'data_baudrate': 2000, 'enable_120hm': True, 'is_fd': True}]
hwhandle = can.Bus(bustype="libtosun", configs=configs,
is_recv_error=True, is_include_tx=True, hwserial=b"")
msg = can.Message(channel=0,arbitration_id=0x1,
is_extended_id=False, is_remote_frame=False, dlc=8, data=[1, 2, 3, 4, 5, 6, 7, 8])
hwhandle.send(msg)
pDiagModuleIndex = c_int32(0) # 为 c 类型 传入的为指针，可以随意赋值，运行下方函数后，会对其赋值
hwhandle.device.tsdiag_can_create(pDiagModuleIndex, 0, 0, 8, 0x123, True, 0X456, True, 0X789, True)
# r 为函数执行返回值，为 0 表示执行成功
# 非 0 可以使用 hwhandle.tscan_get_error_description(r)获取错误信息
r, respond_data =hwhandle.device.tstp_can_request_and_get_response(pDiagModuleIndex, [0x10, 0x02])
print(list(respond_data))
hwhandle.shutdown()
```