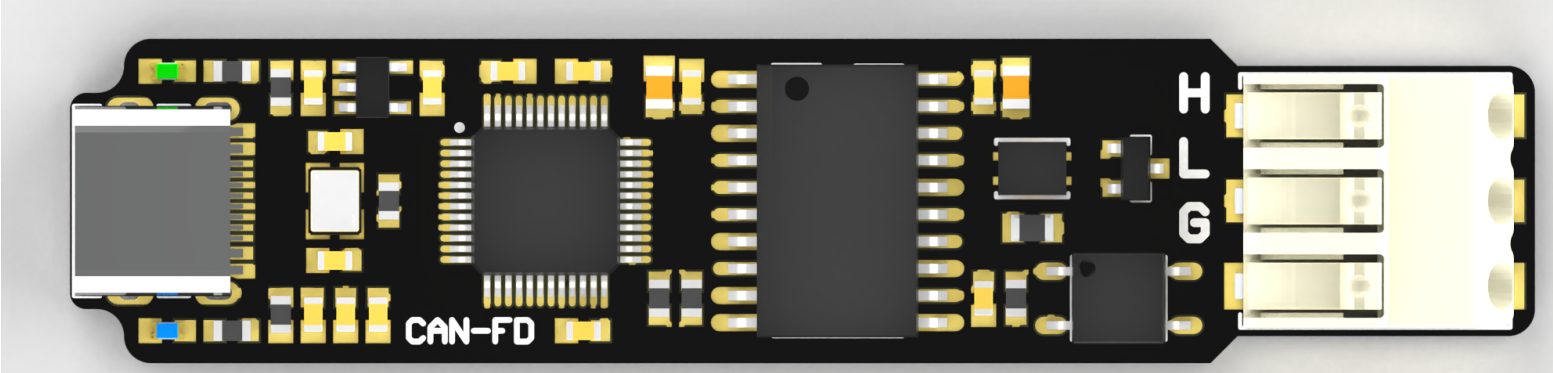
CAN-FD USB-CAN 用户手册

技术支持：baofengdianzi@qq.com

1 CAN-FD模块简介



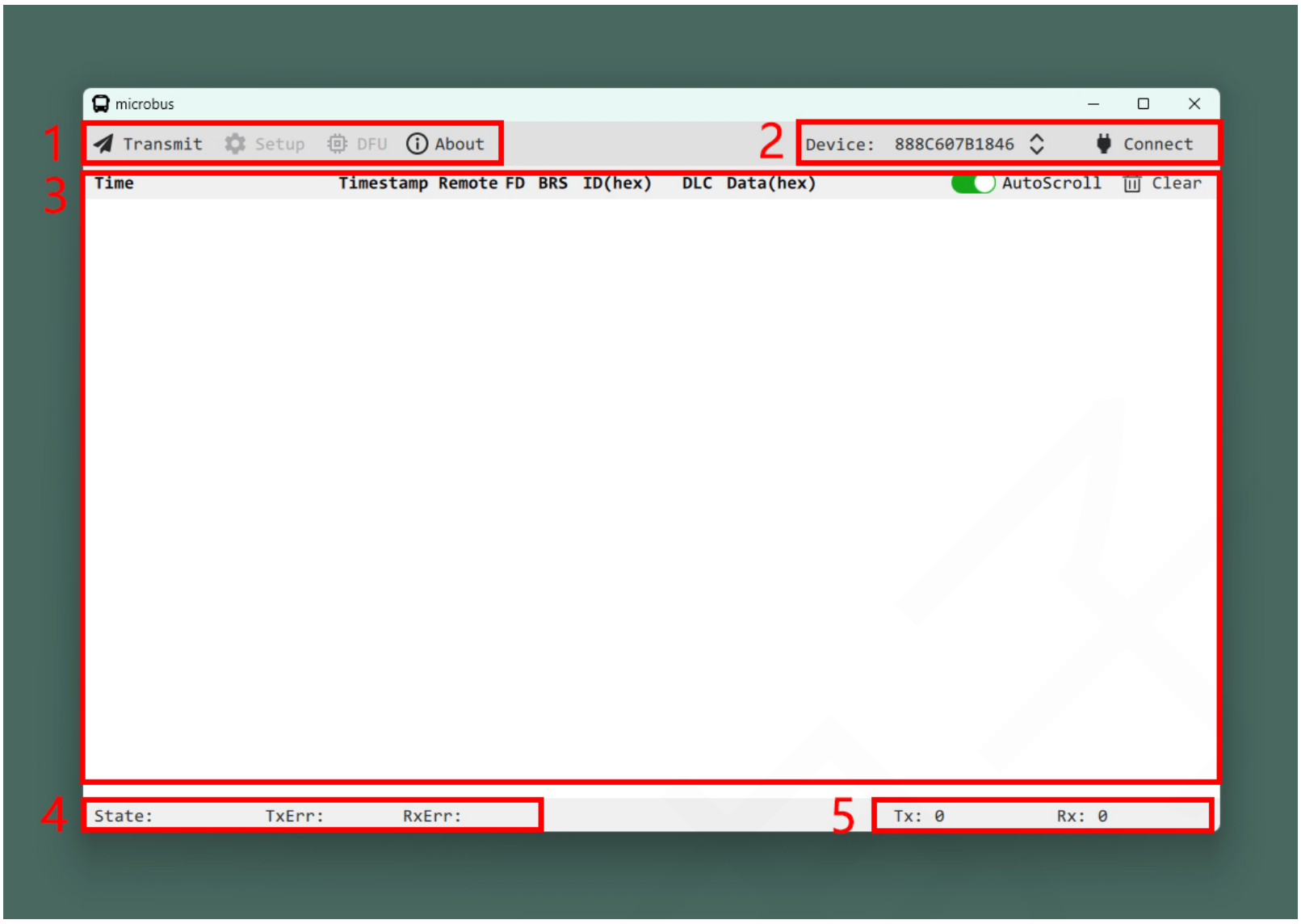
CAN-FD模块 是一款低成本简单好用的 USB-CAN 转换模块。遵循 CAN 2.0A 、CAN 2.0B 、ISO 11891、BOSCH CAN FD 规范。CAN通讯接口高达 5000Vrms 信号和电源隔离。在Windos、Linux、树莓派等系统即插即用免驱动安装。

关键特性：

- 配套软件 microbus
- 支持Python二次开发
- 超小尺寸 6.8cm x 1.5 cm
- CAN-FD 最大波特率 5Mbps
- 采用 Type-C 接口供电和通信
- 支持软件配置120Ω终端电阻使能
- 电源LED指示灯，通信LED指示灯
- 内置CAN总线滤波器，更强抗干扰
- 内部自恢复保险丝，防止损坏主机USB口
- CAN通讯接口高达 5000Vrms 信号和电源隔离
- 采用进口弹簧式接线端子(*支持线规 18~24 AWG*)
- 内置 10 组硬件过滤器，并支持 List、Mask 模式
- CPU 32bit Cortex-M4 高性能内核实现高数据吞吐量
- 支持 CAN 总线协议 2.0A 、2.0B 、ISO 11891、BOSCH CAN FD
- 支持 Normal、Loopback、Silent、SilientLoopBack 等工作模式
- 低温漂高精度晶振，配合USB通讯自动校准，保障高速CAN通讯稳定可靠
- 内置2KB接收缓冲区和2KB发送缓冲区并利用缓冲区平滑技术实现高数据吞吐量

2 mirobus 软件

2.1 主界面

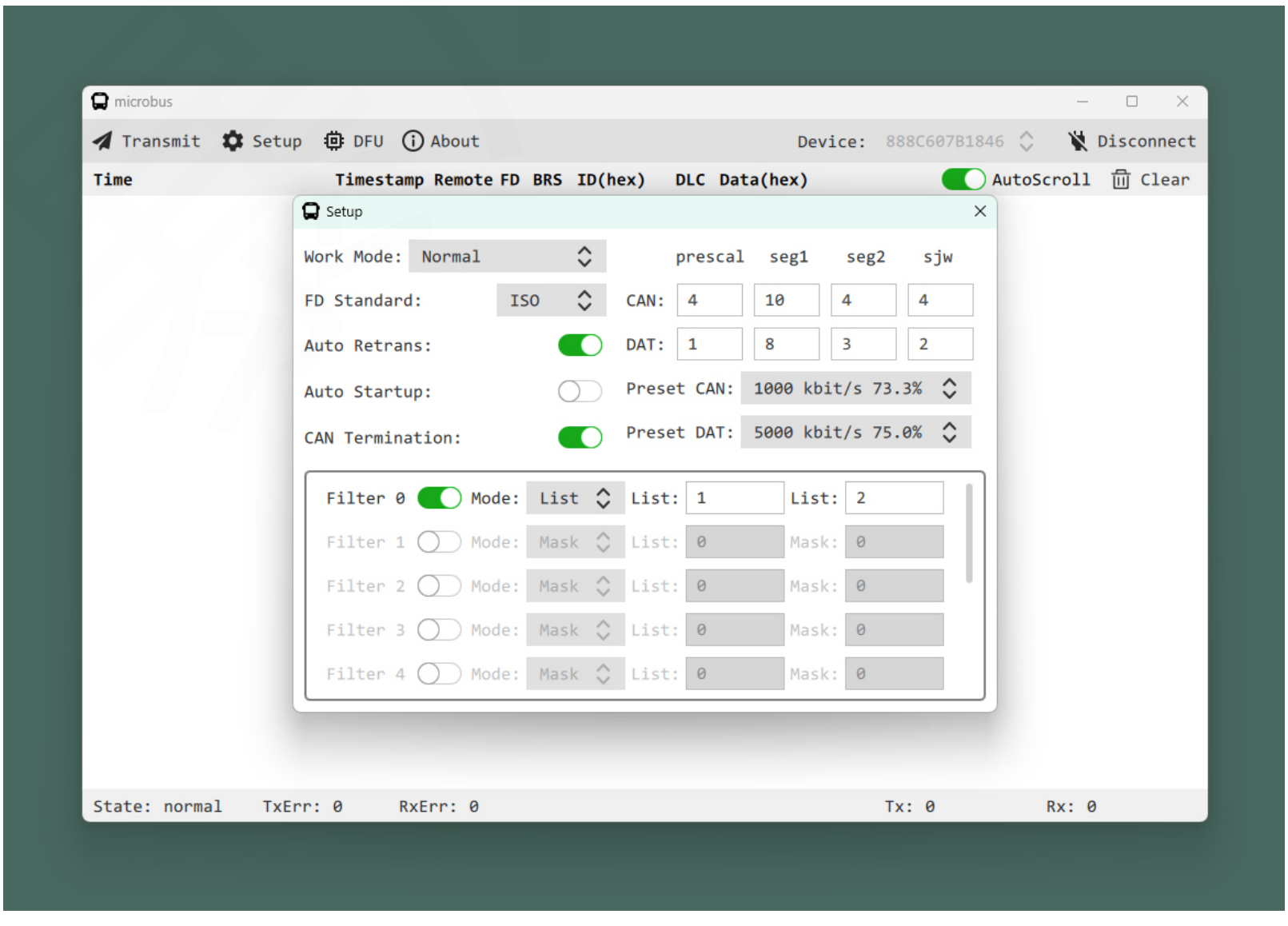


- 文字序号与图中序号对应

1. **工具栏**：用于切换子功能窗口
2. **连接和断开**：连接后模块上的蓝色ACT指示灯亮起表示CAN总线已启动
3. **报文显示窗口**：用于显示发送或者接收到的CAN报文信息
4. **总线状态信息**：CAN总线当前状态，和错误报文计数
5. **发送和接收计数**：发送和接收报文计数

2.2 配置参数

设备连接后，点击工具栏中的 Setup 按钮，打开参数配置面板，如下图：



注意：所有配置项修改后会自动保存到内部ROM中，断电后配置不会丢失！

2.2.1 基本配置

- Work Mode：工作模式配置
  - Normal（正常模式）

    CAN 总线控制器通常工作在正常通信模式下，可以从CAN 总线接收数据，也可以向CAN 总线发送数据
  - Loopback（回环模式）

    在回环通信模式下，由 CAN 总线控制器发送的数据可以被自己接收并存入接收 FIFO ，同时这些发送数据也送至 CAN 网络
  - Silent（静默模式）

    在静默通信模式下，可以从 CAN 总线接收数据，但不向总线发送任何数据
  - SilientLoopBack（回环静默模式）

    在回环静默通信模式下，既不从 CAN 网络接收数据，也不向 CAN 网络发送数据，其发送的数据仅可以被自己接收
- FD Standard：FD 遵循的协议规范配置
  - ISO：遵循 ISO 11891
  - BOSCH：遵循 BOSCH CAN FD
- Auto Retrans：是否启用自动重发机制，禁用后CAN报文数据只会被发送一次，如果因为仲裁失败或者总线错误而导致发送失败，CAN总线控制器不会像通常那样进行数据自动重发
- Auto Startup：是否使能模块上电后自动启动CAN总线
- CAN Termination：是否使能内部 120 欧姆终端电阻

2.2.2 波特率配置

- **选择一**

  可以手动计算CAN仲裁和CAN数据的波特率和采样点，然后将值分别填入CAN和DAT后的 prescal、seg1、seg2、sjw即可**（内部CAN时钟频率为 60 MHz）**
- **选择二**

  使用下拉列表中的预设波特率和采样点，选择好预设项后会自动更新上方的 prescal、seg1、seg2、sjw 值

2.2.3 硬件过滤器配置

如果想要启用某一组过滤器应首先使能，然后选择过滤模式，然后根据选择的过滤模式配置后方的标识符的值

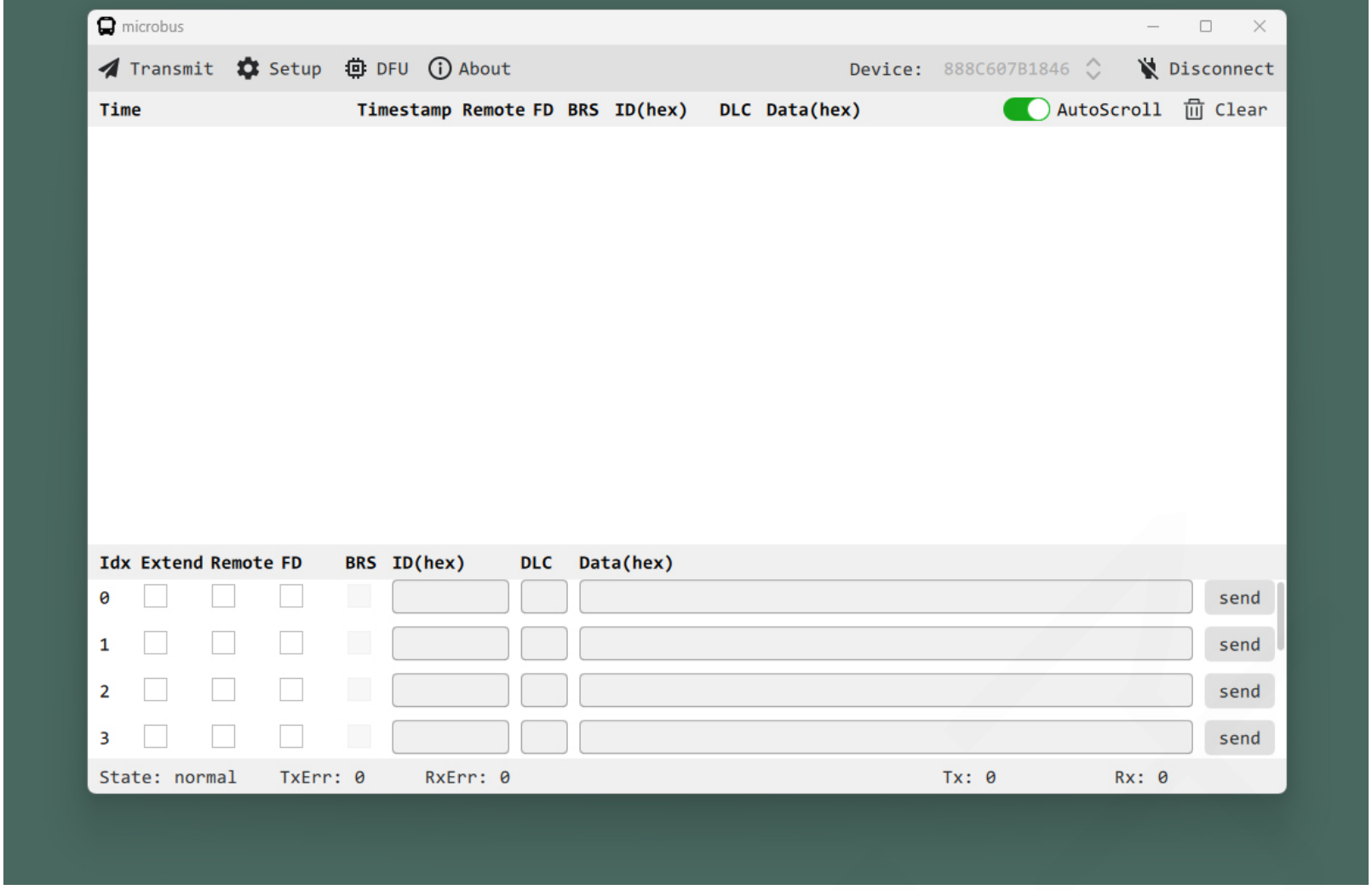
- List 模式：列表模式用来表示与预设的标识符列表中能够匹配则通过，否则丢弃
- Mask 模式：掩码模式用来指定哪些位必须与预设的标识符相同，哪些位无需判断

提示：填入 List 和 Mask 的标识符为 32 位十六进制表示

```
LIST
    bit   31: 帧格式 [0: 标准帧, 1: 扩展帧]
    bit   30: 帧种类 [0: 数据帧, 1: 遥控帧]
    bit   29: Reserved
    bit 28~0: CAN ID
MASK
    bit   31: 帧格式 [0: 标准帧, 1: 扩展帧]
    bit   30: 帧种类 [0: 数据帧, 1: 遥控帧]
    bit   29: Reserved
    bit 28~0: CAN ID
```

2.3 发送CAN报文

设备连接后，点击工具栏中的 Transmit 按钮，打开报文发送窗口，如下图：

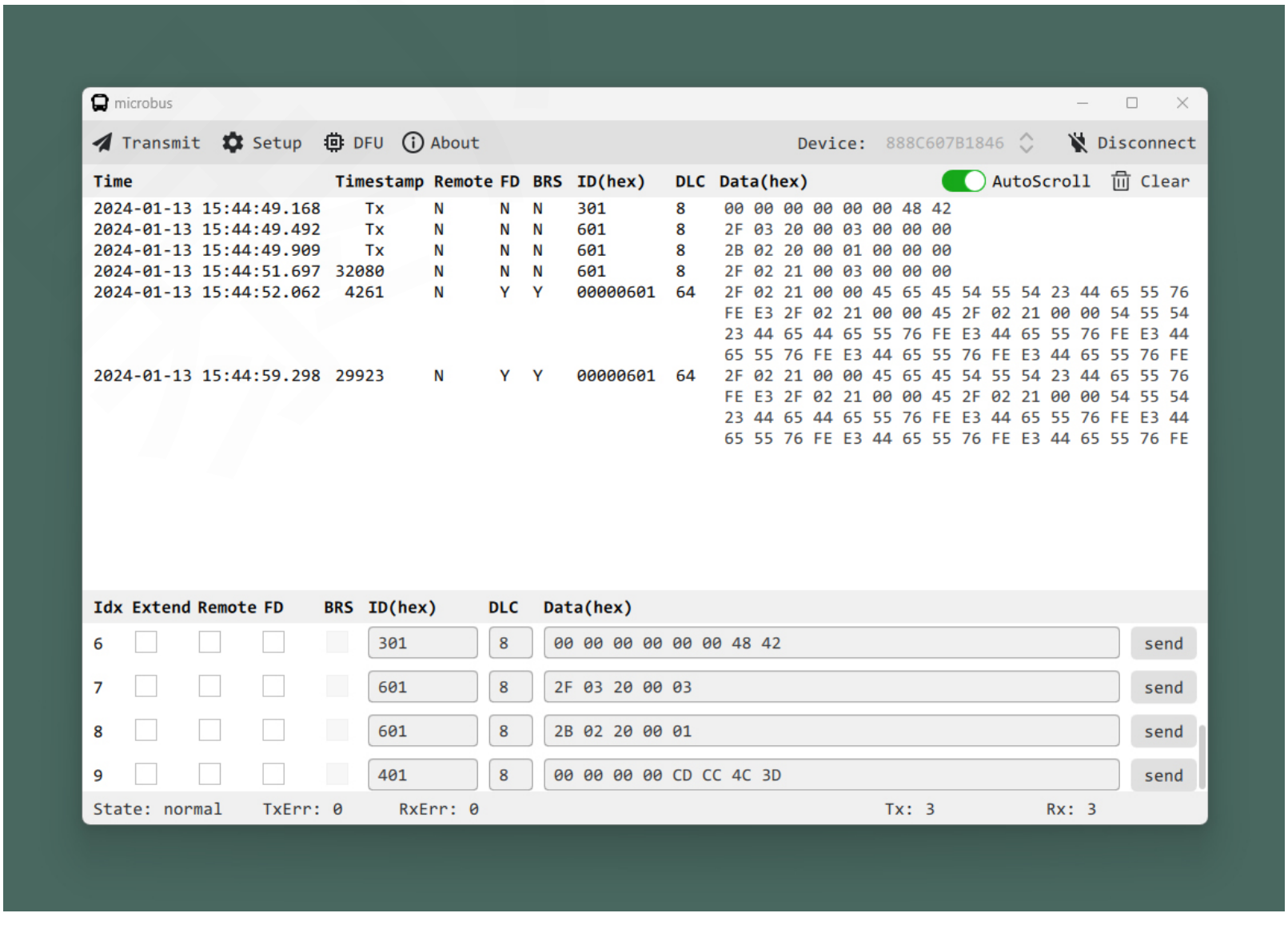


注意：报文发送窗口中可以填入共十条CAN报文，可以根据需要点击其中一条进行发送，软件关闭后会自动保存当前填入报文，以便下次软件启动后加载！

- Extend：是否为扩展帧
- Remote：是否为遥控帧
- FD：是否为 CAN-FD 帧
- BRS：是否支持 CAN-FD 位速率转换
- ID(hex)：CAN ID
- DLC：CAN 数据字节数**（常规帧支持 0~8，CAN-FD帧支持 0~8、12、16、20、24、32、48、64）**
- Data(hex)：CAN 字节数据

2.4 CAN报文显示

设备连接后，发送和接收的报文将实时的显示在报文显示窗口，如下图：



- Time：报文发送或接收时的时间，ms 级精度

- Timestamp：接收报文时间戳（0~65535）单位为 us，如果此报文为发送时显示为 Tx
- Remote：如果为遥控帧显示 Y，否则显示 N
- FD：如果为 CAN-FD 帧显示 Y，否则显示 N
- BRS：如果支持 CAN-FD 位速率转换显示 Y，否则显示 N
- ID(hex)：CAN ID，如果为3位有效数字则为标准帧，如果为8位有效数字则为扩展帧
- DLC：CAN 数据字节数
- Data(hex)：CAN 数据

3 Python库二次开发

canfd 库是基于 Python3 编写的，通过几个简单的函数便可以进行高效的 CAN、CAN-FD 通讯。

3.1 canfd 库安装

推荐通过 Python 包管理工具 **pip** 进行安装，pip的安装请自行Google。

在命令终端中输入如下命令进行 canfd Python 库安装：

```
pip install canfd
```

3.2 canfd API

Class TxMsg

- extend：0：标准帧，1：扩展帧
- remote：0：数据帧，1：遥控帧
- fd：0：常规帧，1：CAN-FD帧
- brs：0：CAN-FD位速率转换失能，1：CAN-FD位速率转换使能
- id：CAN ID
- dlc：CAN数据长度
  - 0: 0 byte
  - 1: 1 byte
  - 2: 2 byte
  - 3: 3 byte
  - 4: 4 byte
  - 5: 5 byte
  - 6: 6 byte
  - 7: 7 byte
  - 8: 8 byte
  - 9: 12 byte
  - 10: 16 byte
  - 11: 20 byte
  - 12: 24 byte
  - 13: 32 byte
  - 14: 48 byte
  - 15: 64 byte
- data：bytearray(64)

Class RxMsg

- extend：0：标准帧，1：扩展帧
- remote：0：数据帧，1：遥控帧
- fd：0：常规帧，1：CAN-FD帧
- brs：0：CAN-FD位速率转换失能，1：CAN-FD位速率转换使能
- id：CAN ID
- dlc：CAN数据长度
  - 0: 0 byte
  - 1: 1 byte
  - 2: 2 byte
  - 3: 3 byte
  - 4: 4 byte
  - 5: 5 byte
  - 6: 6 byte
  - 7: 7 byte
  - 8: 8 byte
  - 9: 12 byte
  - 10: 16 byte
  - 11: 20 byte

    - 12: 24 byte
    - 13: 32 byte
    - 14: 48 byte
    - 15: 64 byte
- data：bytearray(64)
- timestamp_us：微秒时间戳，0~65535

Class CanFD

- scan()

  扫描当前连接到电脑的所有设备，返回设备名称列表。

  :return 设备名称列表
- open(dev, mode, stand, retrans, terminal, can_timing, data_timing)

  启动CAN-FD模块。

  :param dev：设备名称

  :param mode：工作模式，可选 MODE_NORMAL、MODE_LOOPBACK、MODE_SILENT、MODE_SILENT_LOOPBACK。

  :param stand：CAN-FD标准，可选 STAND_ISO、STAND_BOSCH。

  :param retrans：是否自动重发，可选 RETRANS_DISABLE、RETRANS_ENABLE。

  :param terminal：是否使能终端电阻，可选 TERMINAL_DISABLE、TERMINAL_ENABLE。

  :param can_timing：标称波特率时序列表，prescaler(1~1024), seg1(1~128), seg2(1~32), sjw(1~8)

  :param data_timing：数据波特率时序列表，prescaler(1~1024), seg1(1~16)，seg2(1~8)，sjw (1~8)

  :return 0：启动设备成功，-1：设备名称无效 ，-2：启动设备错误
- close()

  停止CAN-FD模块。

  :return 无
- write(tx_msg)

  发送CAN数据帧。

  :param tx_msg：参考 [Class TxMsg]

  :return 0：发送成功，-1：设备未启动
- flush()

  将发送缓冲区的数据强制发送。

  :return 0：刷新成功，-1：设备未启动
- in_waiting()

  获取接收缓冲区当前can帧数量。

  :return 接收缓冲区的can帧数量。
- read()

  从接收缓冲区读取一条can帧。

  :return 参考 [Class RxMsg]
- status()

  获取状态信息。

  :return 错误信息int：
    - bit 6~4: 错误种类
        - [0: 无错误]
        - [[1: 填充错误]
        - [[2: 格式错误]
        - [[3: ACK错误]
        - [[4: 位隐性错]
        - [[5: 位显性错误]
        - [[6: CRC错误]
    - bit 3: Reserved
    - bit 2: 离线错误
    - bit 1: 被动错误
    - bit 0: 警告错误

  :return 接收错误计数

  :return 发送错误计数

3.3 接收数据demo

```
import sys
import time
from canfd import canfd

if __name__ == "__main__":

    # create
    fd = canfd.CanFD()

    # scan device
    dev_list = fd.scan()
    if len(dev_list) == 0:
        print("CAN-FD device not found!")
        sys.exit()

    # print device
    for i in range(len(dev_list)):
        print("dev[%d]: %s" % (i, dev_list[i]))

    # open first device
    # set can baudrate 500K 60%
    can_timing = [8, 8, 6, 4]  # prescaler(1~1024), seg1(1~128), seg2(1~32), sjw(1~8)
    # set fd-data baudrate 500K 60%
    data_timing = [8, 8, 6, 4]  # prescaler(1~1024), seg1(1~16),  seg2(1~8),  sjw (1~8)
    ret = fd.open(
        dev_list[0],
        fd.MODE_NORMAL,
        fd.STAND_ISO,
        fd.RETRANS_DISABLE,
        fd.TERMINAL_ENABLE,
        can_timing,
        data_timing,
    )
    if ret != 0:
        print("device open error!")
        sys.exit()
    print("open")

    # rx msg loop
    print("listening...")
    start_time = time.time()
    while True:
        if time.time() - start_time > 10:
            break

        if fd.in_waiting():
            msg = fd.read()
            print("rx 0x%X : " % msg.id, end="")
            for i in range(fd.dlc_2_len(msg.dlc)):
                print("0x%02X " % msg.data[i], end="")
            print("")

    # close device
    fd.close()
    print("close")
```

3.4 发送数据demo

```
import sys
import time
from canfd import canfd

if __name__ == "__main__":

    # create
    fd = canfd.CanFD()

    # scan device
    dev_list = fd.scan()
    if len(dev_list) == 0:
        print("CAN-FD device not found!")
        sys.exit()

    # print device
    for i in range(len(dev_list)):
        print("dev[%d]: %s" % (i, dev_list[i]))
```

```python
    # open first device
    # set can baudrate 500K 60%
    can_timing = [8, 8, 6, 4]  # prescaler(1~1024), seg1(1~128), seg2(1~32), sjw(1~8)
    # set fd-data baudrate 500K 60%
    data_timing = [8, 8, 6, 4]  # prescaler(1~1024), seg1(1~16),  seg2(1~8),  sjw (1~8)
    ret = fd.open(
        dev_list[0],
        fd.MODE_NORMAL,
        fd.STAND_ISO,
        fd.RETRANS_DISABLE,
        fd.TERMINAL_ENABLE,
        can_timing,
        data_timing,
    )
    if ret != 0:
        print("device open error!")
        sys.exit()
    print("open")

    for i in range(10):
        # tx msg
        msg = canfd.TxMsg()
        msg.extend = 0
        msg.remote = 0
        msg.fd = 0
        msg.brs = 0
        msg.id = 0x123
        msg.dlc = 3
        msg.data[0] = 0xF1
        msg.data[1] = 0xF2
        msg.data[2] = i
        fd.write(msg)

        print("tx 0x%X : " % msg.id, end="")
        for i in range(fd.dlc_2_len(msg.dlc)):
            print("0x%02X " % msg.data[i], end="")
        print("")

        time.sleep(0.01)

    # close device
    fd.close()
    print("close")
```

4 串口二次开发协议

协议采用USB虚拟串口形式以读写寄存器的形式进行二次开发。

4.1 帧格式

读寄存器：

命令字(1 Byte)	帧尾标识(1 Byte)
寄存器地址	固定为 0xAA

读寄存器应答：

命令字(1 Byte)	数据(N Byte)	帧尾标识(1 Byte)
寄存器地址	参考 [4.2 寄存器说明]（注意：组合字节时为小端模式）	固定为 0xAA

写寄存器：

命令字(1 Byte)	数据(N Byte)	帧尾标识(1 Byte)
0x80 + 寄存器地址	参考 [4.2 寄存器说明]（注意：组合字节时为小端模式）	固定为 0xAA

写寄存器应答：

命令字(1 Byte)	写应答标识(1 Byte)	帧尾标识(1 Byte)
0x80 + 寄存器地址	正常应答为 0xFF，异常应答为 0x00	固定为 0xAA

4.2 寄存器说明

0x00 (R) 版本信息

```
    uint8: 固件主版本号
    uint8: 固件次版本号
```

0x01 (RW) 启动停止

```
    uint8:  写0xEE启动CAN总线
            写0xCC关闭CAN总线
```

0x02 (W) CAN帧信息

```
注意：此寄存器较特殊，写此寄存器表示模块向总线上发送CAN报文，同时模块不会返回写寄存器应答，当模块接收到CAN报文时会自动发送读寄存器应答
    uint32: 标识符
            bit   31: 帧格式 [0: 标准帧, 1: 扩展帧]
            bit   30: 帧种类 [0: 数据帧, 1: 遥控帧] !!CAN-FD无遥控帧
            bit   29: Reserved
            bit 28~0: CAN ID
    uint16: 微秒时间戳 0~65535 !!只在接收时有效，发送可忽略
    uint8 : CAN-FD标志
            bit  7~2: Reserved
            bit    1: CAN-FD帧标志位 [0: 常规帧(标准帧或扩展帧), 1: CAN-FD帧]
            bit    0: CAN-FD位速率转换 [0: DISABLE, 1: ENABLE]
    uint8 : CAN数据长度
            bit  7~4: Reserved
            bit  3~0: 数据长度
                    [ 0:  0 byte]
                    [ 1:  1 byte]
                    [ 2:  2 byte]
                    [ 3:  3 byte]
                    [ 4:  4 byte]
                    [ 5:  5 byte]
                    [ 6:  6 byte]
                    [ 7:  7 byte]
                    [ 8:  8 byte]
                    [ 9: 12 byte]
                    [10: 16 byte]
                    [11: 20 byte]
                    [12: 24 byte]
                    [13: 32 byte]
                    [14: 48 byte]
```

```
                [15: 64 byte]
    uint8 : Data 0
    uint8 : Data 1
    uint8 : Data 2
    .
    .
    .
    uint8 : Data 63
```

0x03 (R) 状态信息

```
    uint8: 错误信息
            bit   7: Reserved
            bit 6~4: 错误种类
                    [0: 无错误]
                    [1: 填充错误]
                    [2: 格式错误]
                    [3: ACK错误]
                    [4: 位隐性错]
                    [5: 位显性错误]
                    [6: CRC错误]
            bit   3: Reserved
            bit   2: 离线错误
            bit   1: 被动错误
            bit   0: 警告错误
    uint8: 接收错误计数值
    uint8: 发送错误计数值
```

0x04 (RW) 模式控制

```
    uint8:  bit 7~6: Reserved
            bit 5~4: 工作模式 [0: NORMAL, 1: LOOPBACK, 2: SILENT, 3: SILENT_LOOPBACK]
            bit   3: CAN-FD标准 [0: ISO, 1: BOSCH]
            bit   2: 自动重发 [0: DISABLE, 1: ENABLE]
            bit   1: 上电自动启动CAN总线 [0: DISABLE, 1: ENABLE]
            bit   0: CAN终端电阻 [0: DISABLE, 1: ENABLE]
```

0x05 (RW) 标称波特率时序

```
    uint16: prescaler (1~1024)
    uint8 : seg1 (1~128)
    uint8 : seg2 (1~32)
    uint8 : sjw (1~8)
```

0x06 (RW) 数据波特率时序

```
    uint16: prescaler (1~1024)
    uint8 : seg1 (1~16)
    uint8 : seg2 (1~8)
    uint8 : sjw (1~8)
```

0x10 (RW) 过滤器 0

```
    uint8 : 过滤器控制
                bit 7~2: Reserved
                bit   1: 过滤模式 [0: Mask, 1: List]
                bit   0: 过滤器使能 [0: DISABLE, 1: ENABLE]
    uint32: LIST
                bit   31: 帧格式 [0: 标准帧, 1: 扩展帧]
                bit   30: 帧种类 [0: 数据帧, 1: 遥控帧]
                bit   29: Reserved
                bit 28~0: CAN ID
    uint32: MASK
                bit   31: 帧格式 [0: 标准帧, 1: 扩展帧]
                bit   30: 帧种类 [0: 数据帧, 1: 遥控帧]
                bit   29: Reserved
                bit 28~0: CAN ID
```

0x11 (RW) 过滤器 1

参考过滤器 0

0x12 (RW) 过滤器 2

参考过滤器 0

0x13 (RW) 过滤器 3

参考过滤器 0

0x14 (RW) 过滤器 4

参考过滤器 0

0x15 (RW) 过滤器 5

参考过滤器 0

0x16 (RW) 过滤器 6

参考过滤器 0

0x17 (RW) 过滤器 7

参考过滤器 0

0x18 (RW) 过滤器 8

参考过滤器 0

0x19 (RW) 过滤器 9

参考过滤器 0

0x70 (W) 临时启动CAN通讯

```
注意：临时启动CAN通讯时参数不会保存到内部ROM，适用于需要频繁修改通讯参数的应用
    uint8:  bit 7~6: Reserved
            bit 5~4: 工作模式 [0: NORMAL, 1: LOOPBACK, 2: SILENT, 3: SILENT_LOOPBACK]
            bit   3: CAN-FD标准 [0: ISO, 1: BOSCH]
            bit   2: 自动重发 [0: DISABLE, 1: ENABLE]
            bit   1: 上电自动启动CAN总线 [0: DISABLE, 1: ENABLE]
            bit   0: CAN终端电阻 [0: DISABLE, 1: ENABLE]

    uint16: can_prescaler (1~1024)
    uint8 : can_seg1 (1~128)
    uint8 : can_seg2 (1~32)
    uint8 : can_sjw (1~8)

    uint16: data_prescaler (1~1024)
    uint8 : data_seg1 (1~16)
    uint8 : data_seg2 (1~8)
    uint8 : data_sjw (1~8)
```