```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

'''
Serial packet handler module
'''

__author__ = "Heethesh Vhavle"
__version__ = "1.0.0"
__email__ = "heethesh@cmu.edu"

import time
import serial
import struct
import traceback


class Packet:
    def __init__(self):
        self.is_open = False

        # TX data
        self.tx_slot_encoder = False
        self.tx_encoder = {'encoder_count': 0, 'encoder_velocity': 0}
        self.tx_temperature = 0
        self.tx_ultrasonic_distance = 0
        self.tx_flex_sensor = 0
        self.tx_servo_angle = 0

        # RX data
        self.rx_global_switch = False
        self.rx_state = 0
        self.rx_servo_angle = 0
        self.rx_motor_angle = 0
        self.rx_motor_velocity = 0
        self.rx_stepper_value = 0
        self.rx_stepper_dir = 0
        self.rx_stepper_flag = False

    def start(self, com_port, baud=115200, timeout=0):
        # Configure serial port
        self.ser = serial.Serial()
        self.ser.port = com_port
        self.ser.baudrate = baud
        self.ser.timeout = timeout

        # Time to wait until the board becomes operational
        wakeup = 2
        try:
            self.ser.open()
            print("\n>>> Opening COM Port: " + self.ser.port)
            for i in range(1, wakeup):
                time.sleep(1)
        except Exception as error:
            traceback.print_tb(error.__traceback__)
            self.ser.close()
            self.is_open = False

        # Clear buffer
        self.ser.flushInput()
        self.ser.flushOutput()
        self.is_open = True

    def close(self):
        try:
            self.ser.close()
        except AttributeError as e:
            print(e)
        self.is_open = False

    def generate_frame(self, data, data_format, mode):
        checksum = 0
        frame = ''
        direc = {'tx': '<', 'rx': '>'}

        # Pack data into bytes
        header = struct.pack('cc', '$'.encode('utf-8'), direc[mode].encode('utf-8'))
        payload = struct.pack(data_format, *data)
        data_length = struct.pack('B', len(payload))

        # Calculate checksum
        for byte in payload:
            checksum ^= byte
        checksum = struct.pack('B', checksum)

        # Complete frame
        frame = header + data_length + payload + checksum
        return frame

    def send_packet(self, data, data_format, mode='tx'):
        # Make frame
        tx_frame = self.generate_frame(data, data_format, mode)

        # Send data
        try:
            self.ser.write(tx_frame)

        except Exception as error:
            print(error)
            traceback.print_tb(error.__traceback__)

        # Clear buffer
        self.ser.flushInput()
        self.ser.flushOutput()

    def recieve_packet(self, data_format, data_length):
        checksum = 0
        calcsum = 0
        payload = ''
        rx_data = []

        # Recieve data
        try:
            if self.ser.inWaiting() >= (data_length + 4):
                # Verify header
                if self.ser.read(1).decode('utf-8') != '$':
                    return None
                if self.ser.read(1).decode('utf-8') != '>':
                    return None

                # Verify data length
                data = int(ord(self.ser.read(1).decode('utf-8')))
                if data != data_length:
                    return None

                payload = self.ser.read(data_length)
                checksum = self.ser.read(1)

                # Clear buffer
                self.ser.flushInput()
                self.ser.flushOutput()

                # Verify checksum
                for byte in payload:
                    calcsum ^= byte
                if calcsum != ord(checksum):
                    return None

                # Unpack data
                rx_data = list(struct.unpack(data_format, payload))
                return rx_data

        except Exception as error:
            traceback.print_tb(error.__traceback__)

        # Clear buffer
        self.ser.flushInput()
        self.ser.flushOutput()

        return None

    def send(self):
        data = [
            self.rx_global_switch,
            self.rx_state,
            self.rx_servo_angle,
            self.rx_motor_angle,
            self.rx_motor_velocity,
            self.rx_stepper_value,
            self.rx_stepper_dir,
            self.rx_stepper_flag,
        ]

        self.send_packet(data, '<BBBhhHBB')

    def recieve(self, delay=0.2, max_retries=5):
        data = None
        retries = 0
        while not data:
            if retries > max_retries:
                return False
            time.sleep(delay)
            retries += 1
            data = self.recieve_packet('<BifBHHB', 15)

        self.parse_data(data)
        # self.display()
        return True

    def parse_data(self, data):
        # Boolean data
        self.tx_slot_encoder = data[0]

        # Encoder data
        self.tx_encoder['encoder_count'] = data[1]
        self.tx_encoder['encoder_velocity'] = data[2]

        # Sensors data
        self.tx_temperature = data[3]
        self.tx_ultrasonic_distance = data[4]
        self.tx_flex_sensor = data[5]

        # Servo angle
        self.tx_servo_angle = data[6]

    def display(self):
        print('tx_slot_encoder:', self.tx_slot_encoder)
        print('tx_encoder:', self.tx_encoder)
        print('tx_temperature:', self.tx_temperature)
        print('tx_ultrasonic_distance:', self.tx_ultrasonic_distance)
        print('tx_flex_sensor', self.tx_flex_sensor)
        print('tx_servo_angle:', self.tx_servo_angle)
        print()


if __name__ == '__main__':
    packet = Packet()
    packet.start('/dev/ttyACM0')

    # Recieve test
    while True:
        packet.recieve()

    # Send test
    # packet.rx_global_switch = False
    # packet.rx_state = 10
    # packet.rx_servo_angle = 90
    # while True:
    #     packet.send()
    #     time.sleep(1)

    packet.close()
```