```python
import serial
import time
import logging
from typing import Optional

# ------------------------------------------------
# Global file names (as in extern QString)
# ------------------------------------------------
myFileName = "data.txt"
logFileName = "serial.log"

# ------------------------------------------------
# Logger setup (replacement for Logger.h)
# ------------------------------------------------
logging.basicConfig(
    filename=logFileName,
    level=logging.INFO,
    format="%(asctime)s [%(levelname)s] %(message)s"
)
logger = logging.getLogger("SerialManager")


class SerialManager:
    def __init__(self):
        self.serial_port: Optional[serial.Serial] = None
        self.isopen = False

        self.ser_data = bytearray(2048)
        self.user_data = bytearray(2048)

        self.tcp1Data = -100.0
        self.tcp2Data = -100.0
        self.tcp3Data = -100.0
        self.tcp4Data = -100.0
        self.wing1Data = -100.0
        self.wing2Data = -100.0

        self.actuatorDataLink = 0
        self.cguDataLink = 0

    # ------------------------------------------------
    # Utility
    # ------------------------------------------------
    @staticmethod
    def hex_fn(a: int, b: int) -> int:
        return ((b << 8) | a)

    @staticmethod
    def delay(ms: int):
        time.sleep(ms / 1000.0)

    # ------------------------------------------------
    # Serial init / close
    # ------------------------------------------------
    def initSerialPort(self, portName: str) -> int:
        try:
            self.serial_port = serial.Serial(
                port=portName,
                baudrate=9600,
                bytesize=serial.EIGHTBITS,
                parity=serial.PARITY_NONE,
                stopbits=serial.STOPBITS_ONE,
                timeout=2
            )
            self.isopen = True
            print(f"Success {portName}")
            logger.info(f"Opened Successfully {portName}")
            return 0
        except Exception as e:
            self.isopen = False
            logger.error(str(e))
            return 1
```

```
73
74      def serclose(self):
75          if self.serial_port and self.serial_port.is_open:
76              self.serial_port.close()
77              self.isopen = False
78              print("closed")
79          else:
80              print("Not Closed")
81
82      def serFlush(self):
83          if self.serial_port:
84              self.serial_port.reset_input_buffer()
85              self.serial_port.reset_output_buffer()
86
87      # ------------------------------------------------
88      # TX / RX
89      # ------------------------------------------------
90      def serTx(self, data: bytes) -> int:
91          if not self.isopen:
92              return -1
93          self.serFlush()
94          written = self.serial_port.write(data)
95          print("Bytes Written :", written)
96          logger.info("Tx : " + data.hex().upper())
97          return written
98
99      def serTxStr(self, s: str):
100         self.serTx(s.encode("ascii"))
101
102     def serRx(self) -> int:
103         if not self.isopen:
104             return 0
105
106         rx = bytearray()
107         start = time.time()
108         while time.time() - start < 2:
109             if self.serial_port.in_waiting:
110                 rx.extend(self.serial_port.read(self.serial_port.in_waiting))
111             time.sleep(0.01)
112
113         if rx:
114             self.user_data[:len(rx)] = rx
115             self.ser_data[:len(rx)] = rx
116             print("rxByte.length", len(rx), "Data:", rx.hex().upper())
117             logger.info("Rx : " + rx.hex().upper())
118             return len(rx)
119
120         print("read timeout")
121         return 0
122
123     def serRxNumber(self, number: int) -> int:
124         length = self.serRx()
125         return number if length == number else length
126
127     # ------------------------------------------------
128     # Commands
129     # ------------------------------------------------
130     def CMode(self) -> str:
131         serialData = bytearray()
132         self.serFlush()
133
134         self.serTxStr("L")
135
136         time.sleep(1)
137         if self.serial_port.in_waiting:
138             serialData.extend(self.serial_port.read(self.serial_port.in_waiting))
139
140         return serialData.decode(errors="replace")
141
142     def relayOn(self, chan: int) -> int:
143         cmd = bytes([ord('u'), chan, ord('1'), ord('*')])
144         self.serTx(cmd)
```

```
145             status = self.serRx()
146             return 0 if status in (0, 1) else status
147 
148         def relayOn_alt(self, chan: int) -> int:
149             cmd = bytes([ord('y'), chan, ord('1'), ord('*')])
150             self.serTx(cmd)
151             return self.serRxNumber(1)
152 
153         def relayOnCmd(self, chan: int) -> int:
154             cmd = bytes([ord('r'), chan, ord('1'), ord('*')])
155             self.serTx(cmd)
156             return 1 if self.serRx() >= 1 else 0
157 
158         def wingUncage(self) -> int:
159             return 0 if self.relayOnCmd(0) == 1 else 6
160 
161         def wingCage(self) -> int:
162             return 0 if self.relayOnCmd(1) == 1 else 7
163 
164         def setIpcode(self, ipfreq: int, ipcode: int) -> int:
165             cmd = bytes([ord('p'), 2, ipfreq, ipcode, ord('*')])
166             self.serTx(cmd)
167             status = self.serRxNumber(2)
168             return 1 if status == 2 else 0
169 
170         def ipVal(self) -> str:
171             self.serFlush()
172             self.serTxStr("v*")
173             if self.serRxNumber(7) == 7:
174                 return "".join(f"{b:02x}" for b in self.ser_data[:5])
175             return ""
176 
177         def cguData(self):
178             self.serFlush()
179             self.serTxStr("c*")
180             if self.serRxNumber(35) == 35:
181                 self.cguDataLink = 1
182             else:
183                 self.cguDataLink = 0
184 
185         def writeTelData(self, inc: int) -> int:
186             cmd = b"t" + (b"I" if inc == 0 else b"D") + b"*"
187             self.serTx(cmd)
188             return 0 if self.serRxNumber(1) == 1 else 1
189 
190         def actuatorFeedback(self):
191             self.serFlush()
192             self.serTxStr("f*")
193             if self.serRxNumber(12) == 12:
194                 act = []
195                 for i in range(0, 12, 2):
196                     v = self.hex_fn(self.ser_data[i], self.ser_data[i+1])
197                     act.append((v - 2048) * 20.0 / 4096.0)
198 
199                 (
200                     self.tcp1Data,
201                     self.tcp2Data,
202                     self.tcp3Data,
203                     self.tcp4Data,
204                     self.wing1Data,
205                     self.wing2Data,
206                 ) = act[:6]
207 
208                 self.actuatorDataLink = 1
209             else:
210                 self.actuatorDataLink = 0
211 
212         def actuatorNull(self) -> int:
213             self.serFlush()
214             self.serTxStr("n*")
215             return 0 if self.serRxNumber(1) == 1 else 9
216 
```

```
217         def actuatorCycling(self) -> int:
218             self.serFlush()
219             self.serTxStr("5*")
220             rx = bytearray()
221             end_time = time.time() + 1.3
222 
223             while time.time() < end_time:
224                 if self.serial_port.in_waiting:
225                     rx.extend(self.serial_port.read(self.serial_port.in_waiting))
226                 time.sleep(0.01)
227 
228             if rx.endswith(b'@'):
229                 print("actuatorCycling Completed")
230                 return 0
231             return -1
232 
233         def skrSetSTABCmd(self) -> int:
234             cmd = bytes([ord('S'), 7, ord('G'), 1, 14, 0x21, 0x00, 0xAA, ord('*')])
235             self.serTx(cmd)
236             self.delay(100)
237             return 0
238 
239         def skrSetTrackCmd(self) -> int:
240             cmd = bytes([ord('S'), 7, ord('G'), 1, 14, 0x31, 0x00, 0xAA, ord('*')])
241             self.serTx(cmd)
242             self.delay(100)
243             return 0
244 
```