```
class Command(Enum):
   MOTOR_AZ_STOP = 1
   MOTOR_SWEEP = 2
   MOTOR_AZ_RELEASE = 3
   MOTOR_AZ_GOTO_ENCODER_POS = 4
   MOTOR_AZ_GOTO_ABSOLUTE_POS = 5
   MOTOR_AZ_GOTO_ANGLE_POS = 6
   MOTOR_AZ_MOVE_STEP = 7
   MOTOR_AZ_MOVE_VELOCITY_MODE = 8
   MOTOR_EL_STOP = 9
   MOTOR_EL_GOTO_ABSOLUTE_POS = 11
   MOTOR_EL_GOTO_ANGLE_POS = 12
   MOTOR_EL_MOVE_STEP = 13
   MOTOR_EL_MOVE_VELOCITY_MODE = 14
class Udat():
def __init__(self, log, com_port):
      self.mbus_dev = Modbus(log, com_port)
def get_firmware(self):
      return self.mbus_dev.read_holding_register_u64(40001)
def get_nanotec_firmware(self):
      nanotec_firmware = ""
      for i in range(17):
         fir_digit = self.mbus_dev.read_holding_register_u16(40019 + i)
         nanotec_firmware += chr(fir_digit)
      return nanotec_firmware
def get_alive_counter(self):
     return self.mbus_dev.read_holding_register_s32(40005)
def get_encoder_value(self):
      return self.mbus_dev.read_holding_register_s32(40017)
def get_actual_az_angle(self):
     return self.mbus_dev.read_holding_register_s32(40065) / 1000
def get_actual_el_angle(self):
      return self.mbus_dev.read_holding_register_s32(40067) / 1000
def get_gps_data(self):
      year, mon, day, hour, min, sec, dummy1, dummy2 =
self.mbus_dev.read_holding_registers_u128(40009)
      nsec = self.mbus_dev.read_holding_register_u64(40015)
      # year = self.mbus_dev.read_holding_register_u16(40009)
      # mon = self.mbus_dev.read_holding_register_u16(40010)
      # day = self.mbus_dev.read_holding_register_u16(40011)
      # hour = self.mbus_dev.read_holding_register_u16(40012)
      # min = self.mbus_dev.read_holding_register_u16(40013)
      # sec = self.mbus_dev.read_holding_register_u16(40014)
      return year, mon, day, hour, min, sec, nsec
def get_system_status(self):
      return self.mbus_dev.read_holding_register_u64(40036)
def status_details(self):
      status = self.get_system_status()
      Utility.show_val("azcw_limit", Utility.get_bit(status, 1))
      Utility.show_val("azccw_limit", Utility.get_bit(status, 2))
      Utility.show_val("el_up_limit", Utility.get_bit(status, 3))
      Utility.show_val("el_down_limit", Utility.get_bit(status, 4))
      # Utility.show_val("az_motor_ok", Utility.get_bit(status, 5))
      # Utility.show_val("el_motor_ok", Utility.get_bit(status, 6))
      # Utility.show_val("pol_motor_ok", Utility.get_bit(status, 7))
def goto_az_encoder(self, target_encoder):
      self.mbus_dev.write_holding_registers(40040, target_encoder, DataType.INT32T)
```

```
        return self.apply_command(Command.MOTOR_AZ_GOTO_ENCODER_POS)
    def move_step(self, step_deg):
        self.mbus_dev.write_holding_registers(40044, Utility.x1000_ang_to_count(step_deg),
DataType.INT32T)
        return self.apply_command(Command.MOTOR_AZ_MOVE_STEP)
    def apply_command(self, command):
        return self.mbus_dev.write_holding_registers(40064, command.value, DataType.UINT16T)
    def right_angle(self, angle):
        return self.mbus_dev.write_holding_registers(40056, Utility.x1000_ang_to_count(angle),
DataType.INT32T)
    def left_angle(self, angle):
        return self.mbus_dev.write_holding_registers(40058, Utility.x1000_ang_to_count(angle),
DataType.INT32T)
    def angle_calibration(self, az_count, el_count, az_angle, el_angle):
        self.mbus_dev.write_holding_registers(40069, az_count, DataType.INT32T)
        self.mbus_dev.write_holding_registers(40071, el_count, DataType.INT32T)
        self.mbus_dev.write_holding_registers(40073, Utility.x1000_ang_to_count(az_angle), DataType.INT32T)
        self.mbus_dev.write_holding_registers(40075, Utility.x1000_ang_to_count(el_angle), DataType.INT32T)
    def software_limits(self, az_cw, az_ccw, el_up, el_down):
        self.mbus_dev.write_holding_registers(40087, Utility.x1000_ang_to_count(az_cw), DataType.INT32T)
        self.mbus_dev.write_holding_registers(40089, Utility.x1000_ang_to_count(az_ccw), DataType.INT32T)
        self.mbus_dev.write_holding_registers(40091, Utility.x1000_ang_to_count(el_up), DataType.INT32T)
        self.mbus_dev.write_holding_registers(40093, Utility.x1000_ang_to_count(el_down), DataType.INT32T)
    def velocity_mode_speed(self, speed_as_degree_second):
        return self.mbus_dev.write_holding_registers(40083,
Utility.x1000_ang_to_count(speed_as_degree_second), DataType.INT32T)
      def goto_el_absolute(self, target_pos):
        self.mbus_dev.write_holding_registers(40054, target_pos, DataType.INT32T)
        return self.apply_command(Command.MOTOR_EL_GOTO_ABSOLUTE_POS)
    def set_sweep_speed(self, sweep_speed):
        self.mbus_dev.write_holding_registers(40085, Utility.x1000_ang_to_count(sweep_speed),
DataType.UINT32T)
    def get_resolvers(self):
        az = self.mbus_dev.read_holding_register_u32(40095)
        el = self.mbus_dev.read_holding_register_u32(40097)
        return az, el
    def goto_az_count(self, angle):
        self.mbus_dev.write_holding_registers(40052, Utility.x1000_ang_to_count(angle), DataType.INT32T)
        self.apply_command(Command.MOTOR_AZ_GOTO_ABSOLUTE_POS)
    def goto_az_angle(self, angle):
        self.mbus_dev.write_holding_registers(40048, Utility.x1000_ang_to_count(angle), DataType.INT32T)
        self.apply_command(Command.MOTOR_AZ_GOTO_ANGLE_POS)
    def goto_el_count(self, count):
        self.mbus_dev.write_holding_registers(40054, count, DataType.INT32T)
        self.apply_command(Command.MOTOR_EL_GOTO_ABSOLUTE_POS)
    def goto_el_angle(self, angle):
        self.mbus_dev.write_holding_registers(40050, Utility.x1000_ang_to_count(angle), DataType.INT32T)
        self.apply_command(Command.MOTOR_EL_GOTO_ANGLE_POS)
```