Remote Control of EtherCAT Motor over Ethernet via Python

QEC-M-01 + Moon's Drive — Development Guide

Version 1.2 | 2026.05

ICOP Technology Inc.

Revision History

Version	Date	Description of Changes
1.0	January 2026	Internal engineering version.
1.2	May 2026	First public release. Adds Section 2.1 (86Duino environment setup), Section 2.3 (uploading the code), Chapter 3 (PC network configuration), Chapter 7 (hands-on tutorial), and Chapter 8 download links for the test files.

Table of Contents

1. Overview

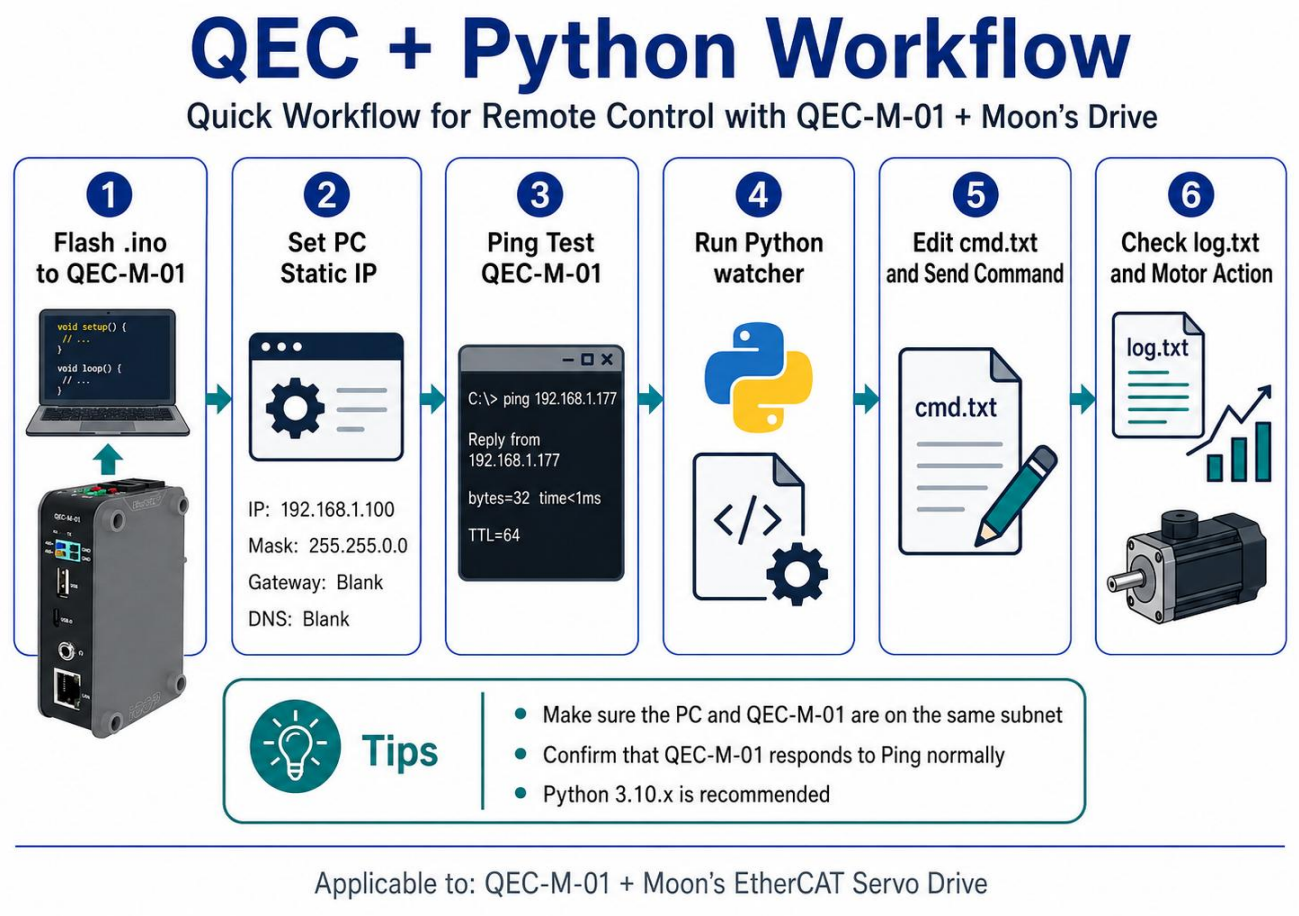
This guide demonstrates a minimal, working remote motor-control architecture. A host Windows PC sends control commands over standard Ethernet (TCP/IP) to a QEC-M-01 running an 86Duino sketch, while continuously logging the position feedback returned by the motor.

The system is composed of two programs:

- `Telnet_Ethercat_MOONS_Log.ino` — flashed onto the QEC-M-01; performs EtherCAT motor control and runs a Telnet server.
- `watch_dir_telnet_log.py` — runs on the Windows PC; watches the command file, sends commands over Telnet, and writes a log.

The architecture requires no special SDK or industrial software — only standard Python 3 and one Ethernet cable.

This guide is suitable for users who want to quickly integrate QEC-M-01 with an upper-level PC application, Python script, test automation system, or custom HMI without directly handling EtherCAT communication on the PC side.



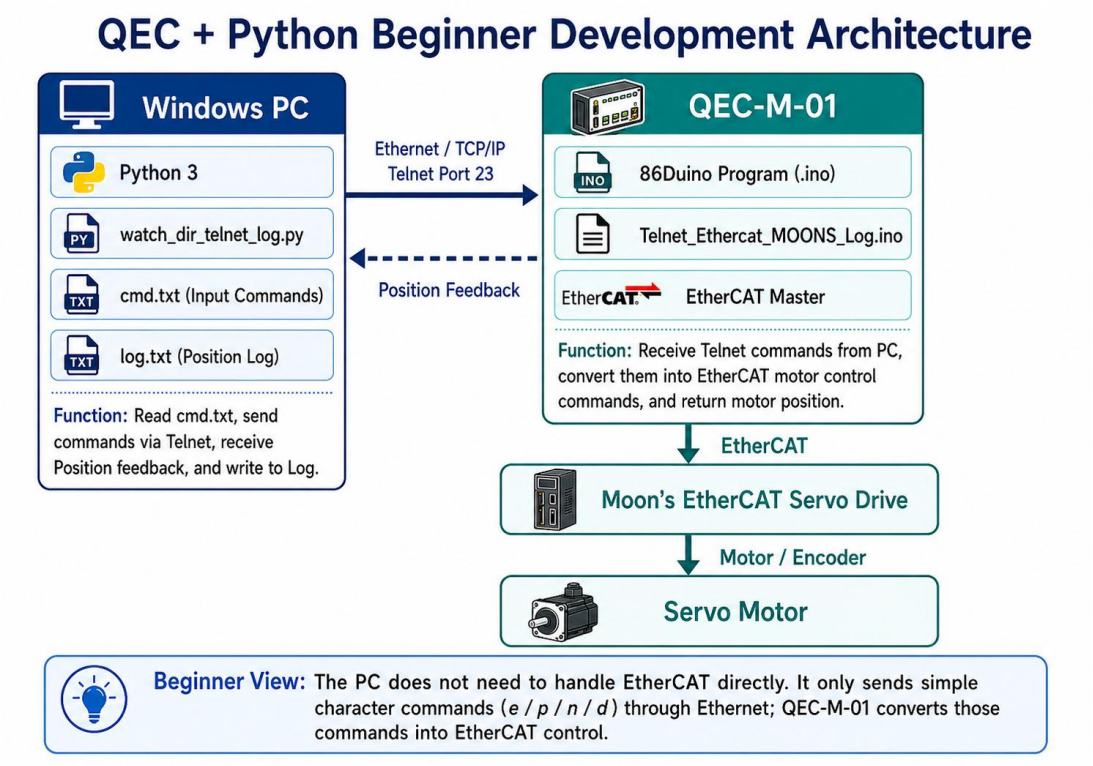
1.1 System Architecture

The hardware and software required for this system are described below.

1.1.1 Hardware Requirements

Component	Model	Description
EtherCAT controller	QEC-M-01	Runs the 86Duino sketch; built-in EtherCAT Master.
EtherCAT servo drive	Moon's (CiA402-compliant)	Connected to the EtherCAT port of the QEC-M-01.
Host PC	Windows (any version)	Runs the Python 3 control program.
Network	Single LAN segment	PC and QEC-M-01 must be in the same subnet.

1.1.2 Communication Architecture



The QEC-M-01 plays a dual role: downward, it controls the servo drive over EtherCAT; upward, it acts as a Telnet server that accepts commands from the PC. The PC side does not need any EtherCAT knowledge — it only needs to be able to open a TCP connection

[Warning]

Telnet is used in this guide for demonstration and testing purposes. It is recommended for closed LAN or lab environments only. For production systems, use a controlled TCP/socket protocol with authentication, command validation, or encryption as required.

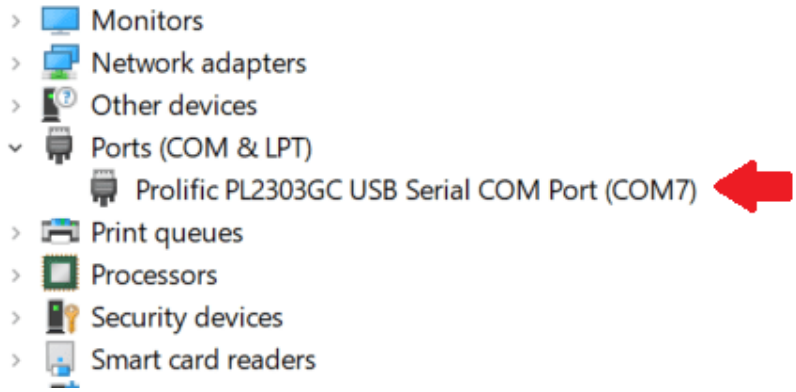
2. Setting Up the QEC-M-01 (Flashing the .ino)

Flash `Telnet_Ethercat_MOONS_Log.ino` onto the QEC-M-01.

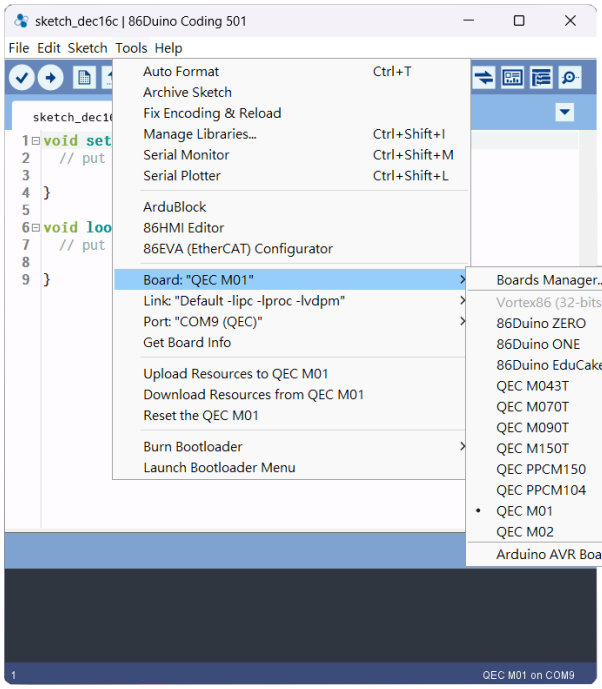
2.1 86Duino Environment Setup

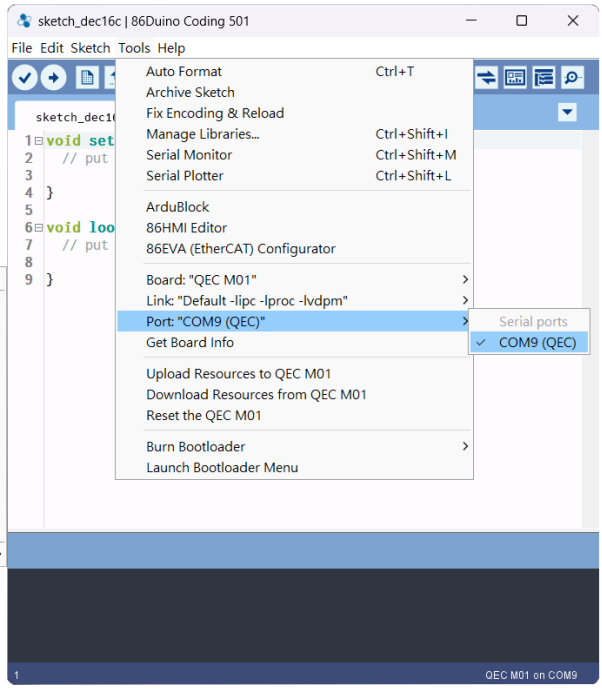
Set up the development environment with the following steps:

1. Connect the QEC-M-01 to your computer with a Micro-USB-to-USB cable (the 86Duino IDE must be installed).
2. Power on the QEC.
3. Open Device Manager (press Win+X and pick it from the menu) → Ports (COM & LPT). Expand the node and you should see "Prolific PL2303GC USB Serial COM Port (COMx)". If it is missing, install the required driver. (The Windows PL2303 driver download here)



4. Launch the 86Duino IDE.
5. Choose the correct board: in the IDE menu select Tools > Board > QEC-M-01 (or whichever QEC MDevice model you are using).
6. Select the port: in the IDE menu select Tools > Port and pick the USB port connected to the QEC MDevice (in this example COM9 (QEC)).





2.2 Walk-through of Telnet_Ethercat_MOONS_Log.ino

The sketch flashed into the QEC-M-01 can be split into three parts: Ethernet network setup, EtherCAT motor initialization, and command parsing with position reporting.

2.2.1 Network Setup

At the top of the .ino, confirm that the following settings match your network environment:

```
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip(192, 168, 1, 177);       // Static IP of QEC-M-01
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 0, 0);
```

> **[Warning]**
> The IP address must be in the same subnet as your PC and must not collide with any other device. Re-flash after changing it.

2.2.2 EtherCAT Motor Initialization

`setup()` performs the EtherCAT initialization in order:

```
master.begin();                        // Start the EtherCAT Master
motor.attach(0, master);               // Attach the first slave (Moon's drive)
master.start(1000000, ECAT_SYNC);      // 1 ms cycle, synchronous mode
motor.setCiA402Mode(CIA402_PP_MODE);   // Profile Position mode
motor.enable();                        // Enable the motor
motor.pp_SetVelocity(100000);          // Velocity: 100,000 pulse/s
motor.pp_SetAcceleration(5000);        // Acceleration
motor.pp_SetDeceleration(5000);        // Deceleration
```

`CIA402_PP_MODE` is the standard Profile Position mode of CiA402, suitable for point-to-point positioning. Moon's drives support this mode out of the box; no extra configuration is needed.

2.2.3 Command Parsing and Position Reporting

`loop()` does two things: receive Telnet commands and broadcast the motor position once per second.

```
char thisChar = client.read();
if (thisChar == 'p') motor.pp_Run(100000);   // Forward to +100000
else if (thisChar == 'n') motor.pp_Run(-100000);  // Reverse to -100000
else if (thisChar == 'd') motor.disable();
else if (thisChar == 'e') motor.enable();

// Report position once per second
if (millis() - t0 >= 1000) {
    server.print("Position: ");
    server.println(motor.getPositionActualValue());
    t0 = millis();
}
```

[Warning]

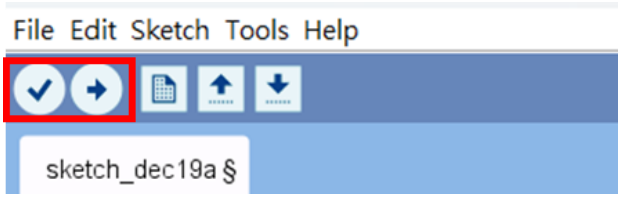
The argument of `pp_Run()` is in pulses (encoder counts). The actual displacement depends on the resolution settings of your drive.

[Tip]

After flashing, use any Telnet client (e.g. PuTTY on Windows) to connect to 192.168.1.177:23. You should see the response "Hello, client!", which confirms the network connection is working.

2.3 Uploading the Code

Once the code is written, click the button on the toolbar to compile. After the compilation completes without errors, click the button to upload. (The sketch name in the screenshot is for illustration only.)

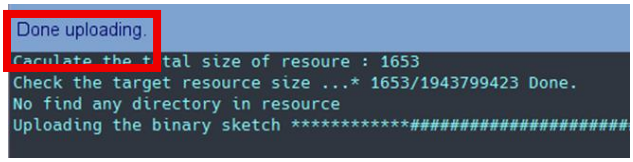


[Figure: 86Duino IDE toolbar — Compile (✓) and Upload (→) buttons highlighted in red.]

[Warning]

During the upload, do NOT close the IDE window and do NOT unplug the USB cable. A typical sketch takes about 10 to 30 seconds to upload.

When the Output window finally shows "Done uploading.", the upload was successful. If red error text appears, fix those errors first. If anything else looks abnormal, please contact us.

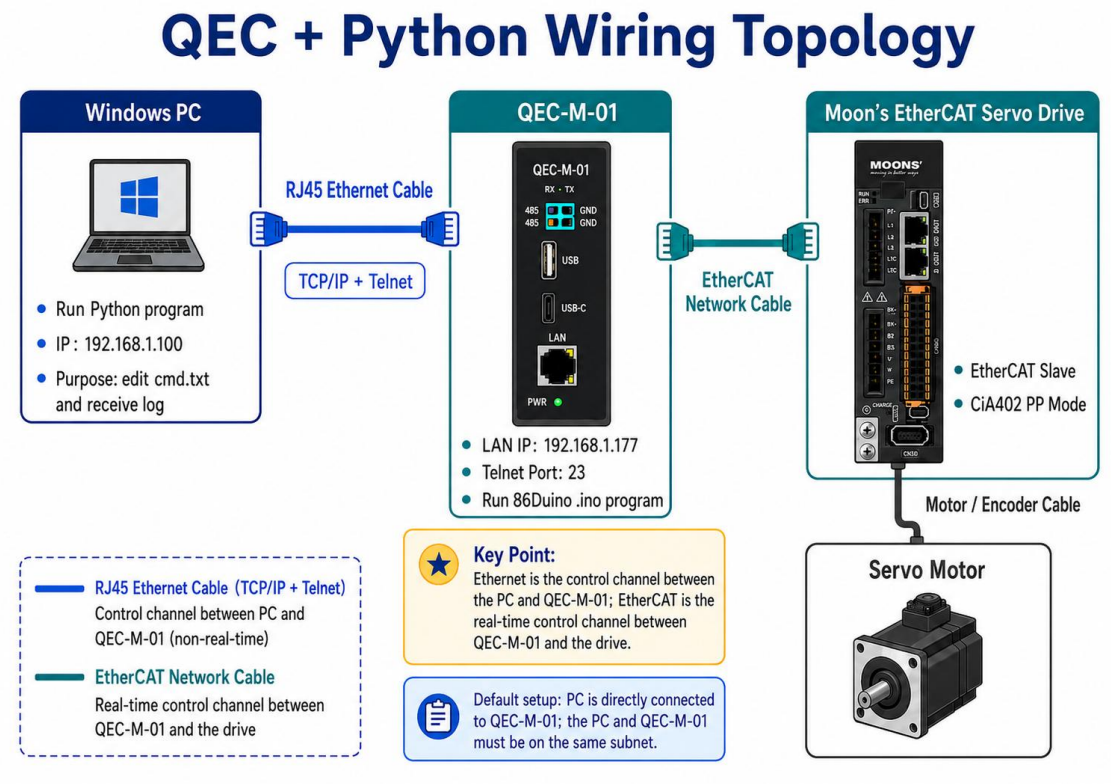


[Figure: Output window showing "Done uploading." — confirmation that the upload succeeded.]

3. PC Network Configuration

By default the PC connects directly to the QEC-M-01 with one RJ45 cable, with no switch or router in between. The PC must therefore be configured with a static IP in the same subnet as the QEC-M-01 (default 192.168.1.177).

3.1 Wiring Topology



3.2 Setting a Static IP on Windows 11 (recommended)

1. Press Win + I to open Settings.
2. In the left pane choose Network & Internet.
3. Click Ethernet. If you have multiple NICs, pick the one connected to the QEC-M-01.
4. Find the IP assignment row and click Edit.
5. Change the dropdown from Automatic (DHCP) to Manual and turn on the IPv4 toggle.
6. Fill in the values below and click Save:
   - IP address: `192.168.1.100`
   - Subnet mask / Subnet prefix length: `255.255.0.0` (or 16)
   - Gateway: leave blank.
   - Preferred / Alternate DNS: leave blank. Keep DNS over HTTPS Off.

[Note]

Some builds of Windows 11 22H2 and later rename the field to "Subnet prefix length" — enter 16 to mean 255.255.0.0.

3.3 Legacy ncpa.cpl Path (Windows 10 / 11, fallback)

1. Press Win + R, type ncpa.cpl and press Enter to open Network Connections.
2. Right-click the NIC connected to the QEC-M-01 and choose Properties.
3. Select Internet Protocol Version 4 (TCP/IPv4) and click Properties.
4. Tick Use the following IP address and enter: IP address 192.168.1.100, Subnet mask 255.255.0.0, leave Default gateway and DNS blank, click OK to save.

3.4 Verification — ipconfig / ping / PuTTY

1. Verify the IP setting took effect: in cmd run `ipconfig`
   - The IPv4 address of the NIC should read 192.168.1.100, with subnet mask 255.255.0.0.
   - If it shows 169.254.x.x, the setting was not applied (still in DHCP mode) — go back and re-check.

4. Setting Up the PC Python Environment

4.1 Environment Requirements

Python 3.7 or later. No additional packages are needed — `watch_dir_telnet_log.py` uses only the standard library (telnetlib, pathlib, argparse, datetime).

[Warning]
telnetlib was deprecated in Python 3.11 and removed in Python 3.13. If your Python is 3.11 or newer, replace telnetlib with the socket module, or pin to Python 3.10 or earlier.

4.2 Preparing the Command File

Create `cmd.txt` in any directory; the initial contents may be blank or any text. For example:

```
C:\my\folder\cmd.txt
```

4.3 Starting the Watcher Program

Open a Windows command prompt, change into the directory that contains `watch_dir_telnet_log.py`, and run:

```
python watch_dir_telnet_log.py --dir "C:\my\folder"
```

4.4 Full Parameter Reference

Parameter	Default	Description	Example
`--dir`	(required)	Directory containing `cmd.txt`	`--dir "C:\folder"`
`--file`	`cmd.txt`	Filename being watched	`--file ctrl.txt`
`--host`	192.168.1.177	IP address of the QEC-M-01	`--host 192.168.1.100`
`--port`	23	Telnet port	`--port 2323`
`--interval`	0.1 (sec)	File polling interval	`--interval 0.05`
`--log`	`log.txt`	Output log filename	`--log output.txt`

[Warning]
By default `log.txt` is written to the working directory at the time the command was issued (i.e. wherever `watch_dir_telnet_log.py` lives), NOT inside the folder given to `--dir`.

5. Remote Motor Control

5.1 Sending Commands

Once the program is running, simply edit the contents of `cmd.txt` to control the motor:

cmd.txt content	Triggered action	Description
e	`motor.enable()`	Enable the motor — must be sent first before any motion.
p	`pp_Run(100000)`	Move to absolute position +100,000 pulses.
n	`pp_Run(-100000)`	Move to absolute position -100,000 pulses.
d	`motor.disable()`	Disable the motor — stops and releases.

The Python program polls `cmd.txt` every 0.1 s and only sends a command when the file content actually changes, avoiding repeated triggers.

> **[Tip]**
> The fastest way to operate is to open `cmd.txt` in Windows Explorer with Notepad, edit, and save with Ctrl+S. Python sees the mtime change and sends the command within 0.1 s.

6. Reading the Log

6.1 Log Format

Each line of `log.txt` has the format:

```
[YYYY-MM-DD HH:MM:SS.mmm] <Telnet response>
```

Example output (from a real test run):

```
[2025-10-22 16:27:27.373] Position: -100000
[2025-10-22 16:27:28.356] Position: -100000
[2025-10-22 16:26:01.873] Position: 99429
[2025-10-22 16:26:02.853] Position: 94534
[2025-10-22 16:26:03.943] Position: 84646
```

During motion the position value changes continuously from the current position toward the target. The full motion trajectory can therefore be reconstructed from the log.

6.2 Known Log Behavior

Because Telnet is a TCP-based stream, the output of `server.print()` and `server.println()` is sometimes split across two lines at the buffer boundary:

```
[2025-10-22 16:26:09.848] -
[2025-10-22 16:26:09.958] 56404
```

Concatenated, these two lines yield the full position value of "-56404".

This is normal TCP fragmentation and does not affect numerical correctness, but any post-processing parser must take it into account.

7. Hands-on Tutorial: First End-to-End Drive Cycle

This chapter ties the previous steps together and walks the reader through a complete e → p → n → d drive cycle, using the Purpose / Prerequisites / Steps / Expected Results / Verification / Common Issues structure, and verifies that the log was written correctly.

7.1 Purpose

Use Python `watch_dir_telnet_log.py` to perform one full e → p → n → d drive cycle, verifying that the `cmd.txt` watcher and the Telnet send mechanism are working, and that motor positions are correctly recorded in `log.txt`.

7.2 Prerequisites

- Step done: the PC NIC is set to 192.168.1.100 / 255.255.0.0, ipconfig confirms it, ping 192.168.1.177 succeeds, and the manual PuTTY test confirms the motor reacts to e/p/n/d.
- Step done: Python is installed; `watch_dir_telnet_log.py` and `cmd.txt` are in place (e.g. `C:\qec\watch_dir_telnet_log.py` and `C:\qec\cmd\cmd.txt`).
- Step done: `Telnet_Ethercat_MOONS_Log.ino` has been flashed onto the QEC-M-01, and the Serial Monitor shows:

```
Begin: 1
Slave: 1
Start: 1
Enable: 1
Chat server address: 192.168.1.177
```

[Warning]
(If the drive's main power is on) the emergency-stop button has been released, the mechanism is clear, and the motor's range of motion will not collide with anything.

7.3 Steps

1. Open cmd and change to the working directory:
   ```
   cd /d C:\qec
   ```
2. Run Python:
   ```
   python watch_dir_telnet_log.py --dir "C:\qec\cmd"
   ```
3. The console should immediately show: [TELNET] Connecting → [TELNET] Connected → [INFO] Watching file...
4. Open `C:\qec\cmd\cmd.txt` in Notepad, change the contents to a single character e, and save it.
5. Within 0.1 s the console should show [SEND] 'e' sent. Any drive alarm should clear; the motor is now energized but not yet moving.
6. Change `cmd.txt` to p and save. The console shows [SEND] 'p' sent and the motor moves forward to about +100,000 pulses.
7. Change `cmd.txt` to n and save. The console shows [SEND] 'n' sent and the motor moves backward to about -100,000 pulses.
8. Change `cmd.txt` to d and save. The console shows [SEND] 'd' sent and the motor is disable.
9. Press Ctrl + C to stop Python. The console shows [INFO] Stopped by user.
10. Inspect `log.txt` (in whichever directory you cd-ed into, i.e. `C:\qec\`). It should contain many lines of the form [YYYY-MM-DD HH:MM:SS.mmm] Position: <value>.

7.4 Expected Results

- Each save of `cmd.txt` produces the corresponding [SEND] message in the console within 0.1 s.
- The motor performs enable / forward / reverse / disable in step with the characters.
- `log.txt` accumulates at least one Position line per second.
- Position values change smoothly from the current position toward ±100,000 during motion.

7.5 Verification

- Console message order: [SEND] 'e' → 'p' → 'n' → 'd'.
- Position values in the Serial Monitor and in `log.txt` change in lock-step (allow ±1 s tolerance).
- After the run, the line count in `log.txt` is roughly equal to the run duration in seconds (with the occasional TCP-fragmented two-liner).

7.6 Common Issues

- [SEND] message appears but the motor does not move: redo the PuTTY manual test in step 2 to localize the problem to the Python side or the drive side.
- `cmd.txt` was edited but Python did not react: some IDEs use a write-and-rename strategy that does not change the file's mtime; use Notepad instead.
- ImportError: No module named 'telnetlib': you are on Python 3.13+; see Chapter 9 (Troubleshooting).
- `log.txt` has no Position lines, only blanks: confirm Python is still running. If the console shows [LOOP] Error, it is usually a socket exception; the program will reconnect automatically.

8. Complete Source Code

This chapter lists the complete source code for reference. If you would rather download the full test files directly (without copy-pasting from this document), please use the links below.

[Download] Complete Test Files

- 86Duino side (.ino plus required library): QEC_Python_Telnet_86DuinoCode.zip
  https://www.qec.tw/wp-content/uploads/2026/05/QEC_Python_Telnet_86DuinoCode.zip
- PC side (watch_dir_telnet_log.py + cmd.txt template): QEC_Python_Telnet_PythonCode.zip
  https://www.qec.tw/wp-content/uploads/2026/05/QEC_Python_Telnet_PythonCode.zip

8.1 Telnet_Ethercat_MOONS_Log.ino

```
/*
  Chat  Server

  A simple server that distributes any incoming messages to all
  connected clients.  To use telnet to  your device's IP address and type.
  You can see the client's input in the serial monitor as well.
  Using an Arduino Wiznet Ethernet shield.

  Circuit:
   Ethernet shield attached to pins 10, 11, 12, 13
   Analog inputs attached to pins A0 through A5 (optional)

  created 18 Dec 2009
  by David A. Mellis
  modified 9 Apr 2012
  by Tom Igoe

*/

#include <SPI.h>
#include <Ethernet.h>
#include "Ethercat.h"
EthercatMaster master;
EthercatDevice_CiA402 motor;

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network.
// gateway and subnet are optional:
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
IPAddress ip(192, 168, 1, 177);
IPAddress dnsserver(192, 168, 1, 1);
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 0, 0);



// telnet defaults to port 23
EthernetServer server(23);
boolean alreadyConnected = false; // whether or not the client was connected previously
int32_t pos = 0;
static uint32_t t0   = 0;

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(115200);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }

  // EtherCAT
  Serial.print("Begin: "); Serial.println(master.begin());
  Serial.print("Slave: "); Serial.println(motor.attach(0, master));
  Serial.print("Start: "); Serial.println(master.start(1000000, ECAT_SYNC));
  motor.setCiA402Mode(CIA402_PP_MODE);
  Serial.print("Enable: "); Serial.println(motor.enable());
  motor.pp_SetMotionProfileType(0); // Linear ramp (trapezoidal profile)
  motor.pp_SetVelocity(100000);
  motor.pp_SetAcceleration(5000);
```

```
  motor.pp_SetDeceleration(5000);

  // initialize the ethernet device
  Ethernet.begin(mac, ip, dnsserver, gateway, subnet);
  // start listening for clients
  server.begin();

  Serial.print("Chat server address:");
  Serial.println(Ethernet.localIP());

  // Set Timing
  t0 = millis();
}

void loop() {
  // Read Position
  pos = motor.getPositionActualValue();

  // wait for a new client:
  EthernetClient client = server.available();

  // when the client sends the first byte, say hello:
  if (client) {
    if (!alreadyConnected) {
      // clead out the input buffer:
      client.flush();
      Serial.println("We have a new client");
      client.println("Hello, client!");
      alreadyConnected = true;
    }

    if (client.available() > 0) {
      // read the bytes incoming from the client:
      char thisChar = client.read();
      if (thisChar == 'p')
        motor.pp_Run(100000);
      else if (thisChar == 'n')
        motor.pp_Run(-100000);
      else if (thisChar == 'd')
        motor.disable();
      else if (thisChar == 'e')
        motor.enable();
    }
  }

  // Error Log / per second
  if (millis() - t0 >= 1000) {
    // echo the bytes back to the client:
    server.print("Position: ");
    server.println(pos);
    // echo the bytes to the server as well:
    Serial.print("Position: ");
    Serial.println(pos);
    t0 = millis();
  }
}
```

8.2 watch_dir_telnet_log.py

```
# -*- coding: utf-8 -*-
"""
Watch the specified file in a folder every 0.1 seconds; only send a command when its content
changes:
  'p' -> motor.pp_Run(100000)
  'n' -> motor.pp_Run(-100000)
  'd' -> motor.disable()
  'e' -> motor.enable()

At the same time, append all data returned from Telnet (numbers/text/strings) to log.txt
(created automatically if it does not exist) on every cycle.
- Non-blocking read of the telnet buffer; each line is prefixed with the local timestamp.
- The connection is automatically re-established if it drops.
"""

import argparse
import time
import telnetlib
from pathlib import Path
from typing import Optional
from datetime import datetime

VALID_CMDS = {'p', 'n', 'd', 'e'}

def connect_telnet(host: str, port: int, retry_seconds: float = 2.0) -> telnetlib.Telnet:
    """Connect to telnet; on failure, retry every retry_seconds until successful."""
    while True:
        try:
            print(f"[TELNET] Connecting to {host}:{port} ...")
            tn = telnetlib.Telnet(host, port, timeout=5)
            print("[TELNET] Connected.")
            return tn
        except Exception as e:
            print(f"[TELNET] Connect failed: {e} ; retry in {retry_seconds}s")
            time.sleep(retry_seconds)

def send_char(tn: telnetlib.Telnet, ch: str) -> bool:
    """Send a single character (no newline). Returns True on success."""
    try:
        tn.write(ch.encode("ascii"))
        print(f"[SEND] '{ch}' sent")
        return True
    except Exception as e:
        print(f"[SEND] Failed: {e}")
        return False

def read_command_char(file_path: Path) -> Optional[str]:
    """
    Read the first non-whitespace character from the file; returns None if it is not
    p/n/d/e or if the file does not exist.
    """
    if not file_path.exists() or not file_path.is_file():
        return None
    try:
        text = file_path.read_text(encoding="utf-8", errors="ignore")
    except Exception:
        return None
    for c in text:
        if not c.isspace():
            c = c.lower()
            return c if c in VALID_CMDS else None
    return None

def append_log_line(log_path: Path, line: str) -> None:
    """Append a single line (including newline) to log.txt (created automatically if
missing)."""
    with open(log_path, "a", encoding="utf-8", errors="ignore") as f:
```

```
        f.write(line)

def drain_telnet_to_log(tn: telnetlib.Telnet, log_path: Path) -> None:
    """
    Non-blocking read of all data currently in the telnet buffer and write it to log.txt
    (each line prefixed with a timestamp).
    """
    try:
        chunk = tn.read_eager()  # Non-blocking; may be empty
    except EOFError:
        # Connection has been closed; let the outer loop handle the reconnect
        raise
    if not chunk:
        return
    text = chunk.decode(errors="ignore").replace("\r\r\n", "\n").replace("\r\n", "\n")
    ts = datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f")[:-3]
    for line in text.splitlines():
        append_log_line(log_path, f"[{ts}] {line}\n")

def main():
    ap = argparse.ArgumentParser(description="Watch file content change and send 'p'/'n'/'d'/'e'
via Telnet; log all Telnet outputs.")
    ap.add_argument("--dir", required=True, help="Folder to watch")
    ap.add_argument("--file", default="cmd.txt", help="File name to watch (under --dir, default
cmd.txt)")
    ap.add_argument("--host", default="192.168.1.177", help="Telnet host (default
192.168.1.177)")
    ap.add_argument("--port", type=int, default=23, help="Telnet port (default 23)")
    ap.add_argument("--interval", type=float, default=0.1, help="Polling interval in seconds
(default 0.1)")
    ap.add_argument("--log", default="log.txt", help="File to write Telnet output (default
log.txt in the working directory)")
    args = ap.parse_args()

    folder = Path(args.dir).resolve()
    if not folder.is_dir():
        raise SystemExit(f"--dir is not a folder: {folder}")
    file_path = (folder / args.file).resolve()
    log_path = Path(args.log).resolve()

    # Establish the telnet connection
    tn = connect_telnet(args.host, args.port)

    last_mtime: Optional[float] = None
    last_cmd: Optional[str] = None

    print(f"[INFO] Watching file: {file_path} (Ctrl+C to stop)")
    try:
        while True:
            try:
                # 1) Send a command only when the file content has changed
                if file_path.exists():
                    mtime = file_path.stat().st_mtime
                    if last_mtime is None or mtime != last_mtime:
                        cmd = read_command_char(file_path)
                        if cmd is not None and cmd != last_cmd:
                            if not send_char(tn, cmd):
                                # On disconnect, reconnect and retry once
                                try:
                                    tn.close()
                                except Exception:
                                    pass
                                tn = connect_telnet(args.host, args.port)
                                send_char(tn, cmd)
                            last_cmd = cmd
                        last_mtime = mtime
                else:
                    last_mtime = None  # Will be detected when the file is recreated
```

```
                # 2) Write all data returned by the telnet side to log.txt
                try:
                    drain_telnet_to_log(tn, log_path)
                except EOFError:
                    # Connection closed; reconnect
                    try:
                        tn.close()
                    except Exception:
                        pass
                    tn = connect_telnet(args.host, args.port)

                time.sleep(args.interval)
            except Exception as e:
                print(f"[LOOP] Error: {e}")
                time.sleep(0.5)
    except KeyboardInterrupt:
        print("\n[INFO] Stopped by user.")
    finally:
        try:
            tn.close()
        except Exception:
            pass

if __name__ == "__main__":
    main()
```

9. Troubleshooting (FAQ)

Symptom	Likely cause and remedy
"Connect failed" keeps retrying	(1) Wrong IP — verify that the QEC-M-01 and the PC are on the same subnet. (2) The QEC-M-01 has not finished booting — wait until the Serial Monitor displays the IP, then start Python.
Commands sent but motor does not respond	(1) Make sure 'e' has been sent first (the motor must be enabled). (2) Check the EtherCAT link — Begin / Slave / Start in the Serial Monitor must all return non-zero.
Position values split across two lines in the log	Normal — caused by TCP fragmentation. Concatenate the digit string of two adjacent lines to recover the full value.
telnetlib ImportError	Python is 3.13 or newer — telnetlib has been removed. Switch to Python 3.10 or earlier, or rewrite using the socket module.
The same command is sent multiple times	`cmd.txt` was repeatedly modified by some external tool (e.g. an automatic backup utility). The Python program only sends a command when the cmd value differs from the previous one, so the same command is never sent twice on its own.

10. Next Steps

This example shows the most basic single-axis remote-control architecture. Possible extensions on the QEC platform include:

- Multi-axis control: add `motor2.attach(1, master)`, and so on, supporting multiple EtherCAT slaves.
- A finer-grained command protocol: pass the target position as a number inside the Telnet string instead of using the fixed ±100,000.
- SubDevice I/O integration: read sensors or drive Digital Output through QEC SubDevice modules.
- HMI integration: wrap the Python control logic in a GUI built with tkinter or PyQt for a visual operator interface.

Visit the QEC product page for QEC-M-01 specifications and the list of supported EtherCAT servo brands: https://www.qec.tw/

11. Contact us

If you run into any technical issues while following this guide, or if you have a purchasing or custom-design enquiry about the QEC product line, please reach out through either of the channels below:

- **Sales / Purchasing:** info@qec.tw (QEC product line — EtherCAT MDevice, etc.)
- **Technical Support / R&D:** info@icop.com.tw (ICOP Technology Inc.)

We typically respond within 1 to 2 business days. For urgent on-site issues, please write "URGENT" in the email subject and attach a full screenshot of the Serial Monitor output to speed up triage.