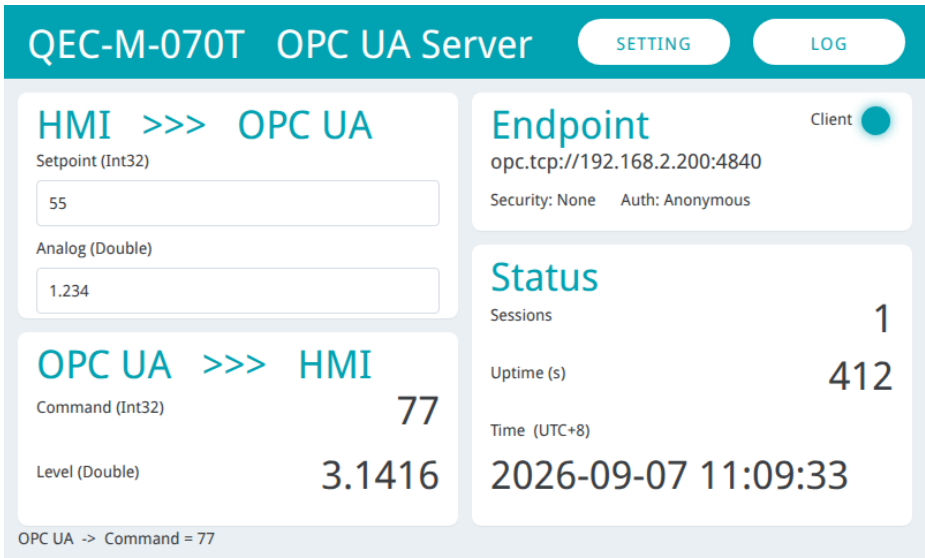


QEC OPC UA Server Test Tool - User Manual v 1.0

Using 86Duino to expose HMI data over OPC UA and to validate the link from a SCADA client

Development Guide | Version 1.0 | September 2026
ICOP Technology Inc.



Cover — The Main page with one client connected.

Revision History

Version	Date	Description
1.0	September 2026	First release. Documents the QEC_OPCTUA_Server_HMI release sketch: on-panel REBOOT, the Active now column, and node direction protection.

Table of Contents

Revision History	1
Table of Contents	2
1. Overview	3
1.1 System Architecture	3
1.1.1 Hardware Requirements	3
1.1.2 Communication Architecture	4
2. Setting Up the QEC MDevice	5
2.1 PC-Side Installation	5
2.1.1 86Duino Coding IDE 502	5
2.1.2 open62541 library	5
2.1.3 UaExpert (for validating the server)	5
2.2 Code Description	8
2.2.1 Network Settings	9
2.2.2 HMI	9
2.2.3 Application Logic	10
2.2.4 Task / Runtime	10
2.2.5 OPC UA Server	10
2.3 Uploading the Code	11
3. OPC UA Address Space and Node Contract	12
3.1 Node Table	12
3.2 Access Direction	13
3.3 What you see once a client connects	13
4. HMI Page Function Description	14
4.1 Main — bidirectional data flow	14
4.2 Log — a live console on the panel	15
4.3 Settings — change IP or port without reflashing	16
CONFIRM validation rules	17
Double-press confirmation	18
5. Walkthrough: First Data Exchange	19
5.1 What this chapter does	19
5.2 Before you start	19
5.3 Steps	19
5.4 Once connected	21
5.5 One write in each direction	21
Things you will notice along the way	22
5.6 These messages are normal	22
6. Network Configuration Persistence	23
6.1 Why a restart is required	23
6.2 Why the Port Can Be Changed on the Panel	23
7. Troubleshooting	24
8. Complete Code and Downloads	26
9. Next Steps	27
9.1 Connect your own upstream system	27
9.2 Replace the nodes with your own data	27
9.3 Turn the QEC into a client instead	27
9.4 Coexisting with EtherCAT	27
9.5 Security, before this goes live	27
10. Contact Us	29

1. Overview

`QEC_OPCUA_Server_HMI` is an OPC UA server test tool that runs on any **QEC-M-series MDevice with a built-in panel**. Flash it and the server is online at `opc.tcp://<IP>:4840` on the next boot, ready for UaExpert on a PC, or the plant SCADA or MES, to connect and read and write data.

The tool has three parts, and **all of them run on the same Vortex86EX2**:

Part	Implementation	Notes
OPC UA server	open62541 v1.1.2 (86Duino / DMP port)	10 nodes under <code>Objects/QEC</code> , Security None / Anonymous
HMI panel	Touch panel, LVGL86 + the 86HMI designer	Three pages: Main / Log / Settings
Configuration persistence	EEPROM bank 0	IP / mask / gateway / port are editable on the panel and applied on restart

No external OPC UA gateway, no server process on a PC, no separate HMI panel PC.

The tool exists for three jobs:

1. Prove that the OPC UA link between a QEC unit and the plant's upstream system works
2. Confirm that data flows **both ways** — panel input readable upstream, upstream writes visible on the panel
3. Validate the network settings and the data types before you start writing your own application

1.1 System Architecture

1.1.1 Hardware Requirements

Component	Recommended Specification	Description
QEC MDevice	Any QEC-M-series MDevice with a built-in panel	Screenshots in this manual use the QEC-M-070T (Vortex86EX2, 7" 800×480 resistive touch panel); other panel models in the series work the same way
Network	1× Gigabit RJ45	OPC UA uses this port. The other 2× 10/100 RJ45 ports are EtherCAT only — a cable in the wrong port means no connection
Host PC	Windows 10/11 or Linux x86-64, same subnet as the QEC	Runs UaExpert, or your existing SCADA / MES
Power	Per the model's specification	Where a power cycle is impractical on site, the panel's REBOOT button performs a soft reset

Software requirements

Component	Version	Description
86Duino Coding IDE	502	The version announced on the QEC site
open62541 library	v1.1.2	Installed from the IDE's Library Manager (search <code>OPC</code>)
86HMI designer	Bundled with the IDE	Generates <code>myhmi.h</code> / <code>myhmi.cpp</code>
UaExpert	2.x	Unified Automation's free OPC UA client, used for validation

Moving to another QEC-M model: the sketch needs no changes. The HMI layout is laid out for **800×480**; if your model's resolution differs, re-arrange it once in the 86HMI designer (see §8). The `QEC.ModelName` node and the `"QEC-M-070T"` in

the boot banner are hard-coded strings that should be changed to match a different model.

1.1.2 Communication Architecture

In OPC UA the connection **always** goes client-initiated, server-passive; two clients cannot connect to each other. So the question of *which side is the server* determines the shape of the whole system.

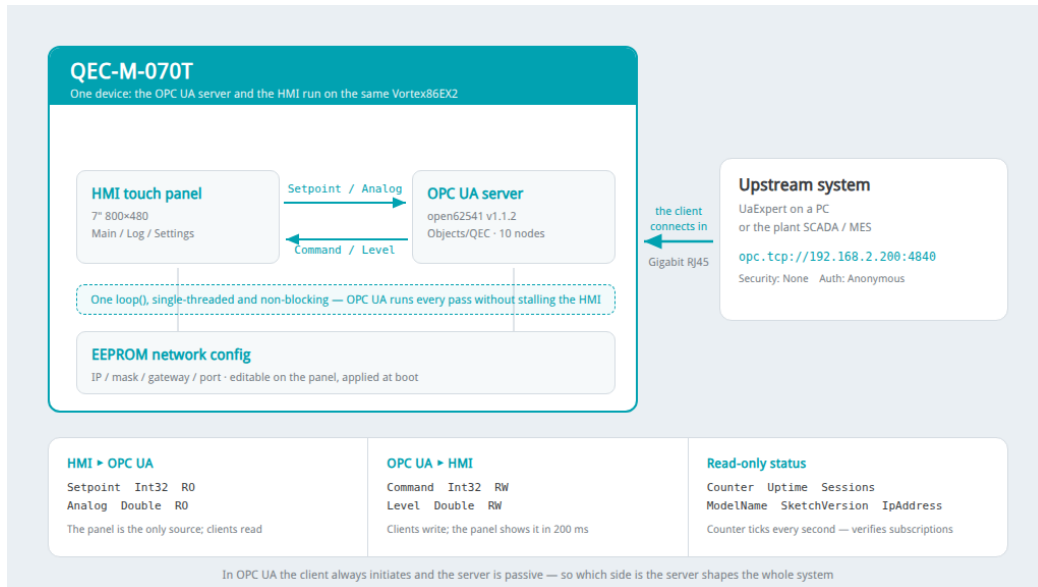


Figure 1-1 — System architecture and data flow: the upstream system connects in as a client, while the HMI panel, the OPC UA server and the configuration store all run on the same Vortex86EX2.

In this tool the QEC is the server. Telling the two roles apart is easy — look at which API set the sketch uses:

```
UA_Server_new() / UA_Server_run_startup() / UA_Server_run_iterate() -> QEC is the server (this manual)
UA_Client_new() / UA_Client_connect() / UA_Client_readValue...() -> QEC is the client (separate manual)
```

The `QEC.Sessions` node reports how many clients are currently connected — a count only a server can know.

Both roles can coexist in one sketch (server towards the upstream system, client towards downstream devices); the library supports both.

Non-blocking by design: every pass of `loop()` calls `UA_Server_run_iterate(server, false)`, which never blocks. That is the prerequisite for running EtherCAT alongside it later (see §9).

2. Setting Up the QEC MDevice

2.1 PC-Side Installation

2.1.1 86Duino Coding IDE 502

1. Download and install **86Duino Coding IDE 502** from the QEC site
2. Connect the QEC MDevice to the PC with a USB cable
3. Open Windows **Device Manager** and confirm the QEC's COM port appears under "Ports (COM port)" (**COM12** in the examples)
4. In the IDE, set **Tools** → **Board** to your QEC-M model (**QEC-M-070T** in this manual) and **Tools** → **Port** to the COM port from the previous step

2.1.2 open62541 library

1. **Sketch** → **Include Library** → **Manage Libraries...**
2. Type **OPC** in the search box → find **open62541** → **Install**
3. After installation, **open62541** appears in the **Sketch** → **Include Library** list at version **1.1.2**

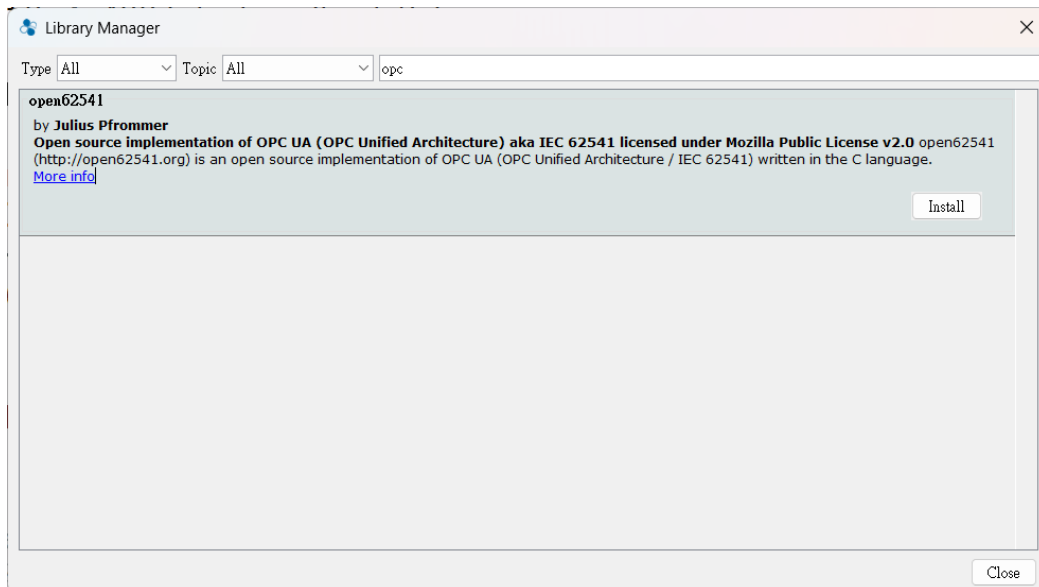


Figure 2-1 — Library Manager after searching opc: the open62541 entry is the library to install. Click Install; Sketch → Include Library then reports version 1.1.2.

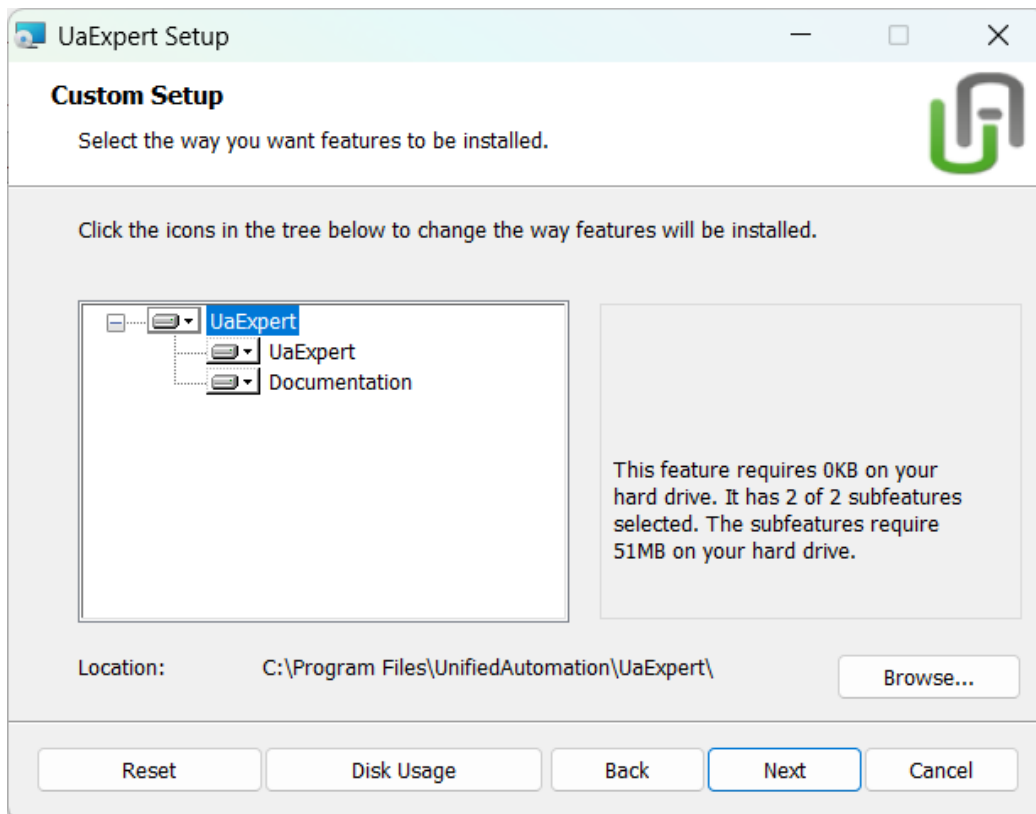
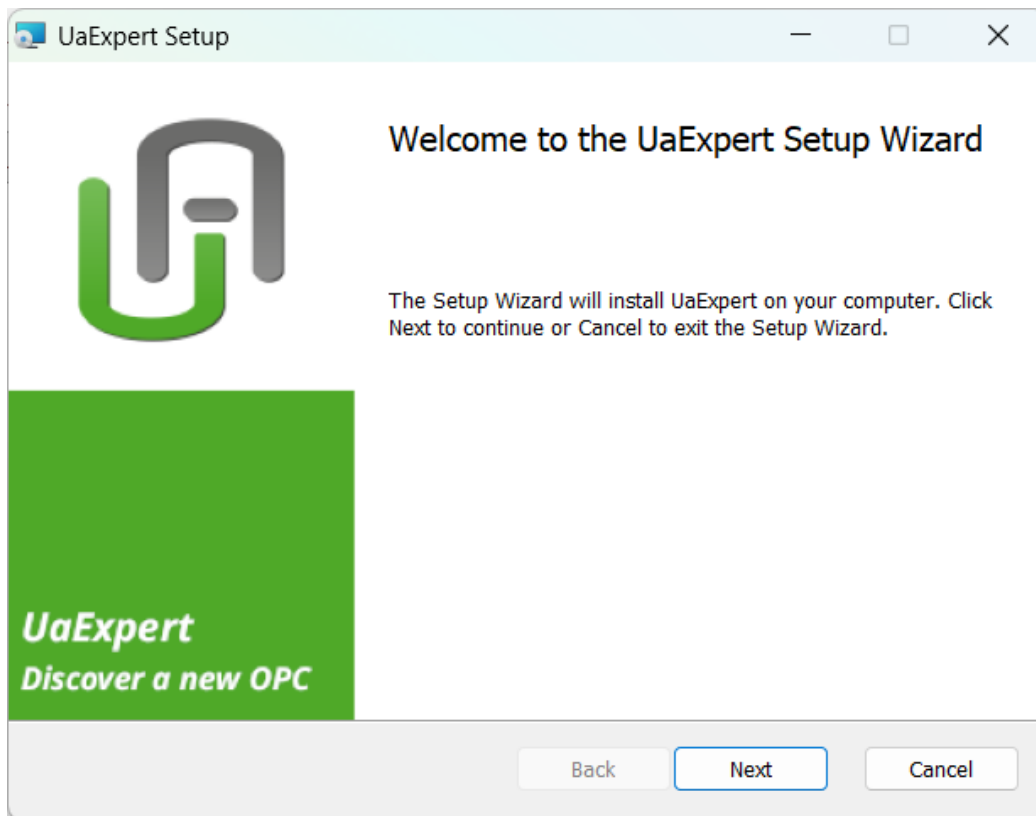
Offline install: copy the **open62541** folder straight into the IDE's **libraries** directory and restart the IDE.

Upgrade note: if an older version is still present (its folder is capitalised **Open62541**), **delete it first**. The new **library.properties** uses the lower-case name **open62541**, and with both present the IDE may pick the old one.

2.1.3 UaExpert (for validating the server)

UaExpert is Unified Automation's free OPC UA client, and §5 uses it to connect into the QEC's server.

1. Register an account on the Unified Automation site (the free edition is enough)
2. **Downloads** → **OPC UA Clients** → UaExpert → download and install
3. Click **Next** through the wizard: leave **Custom Setup** at its defaults (UaExpert + Documentation, about 51 MB) → **Install** → **Finish**



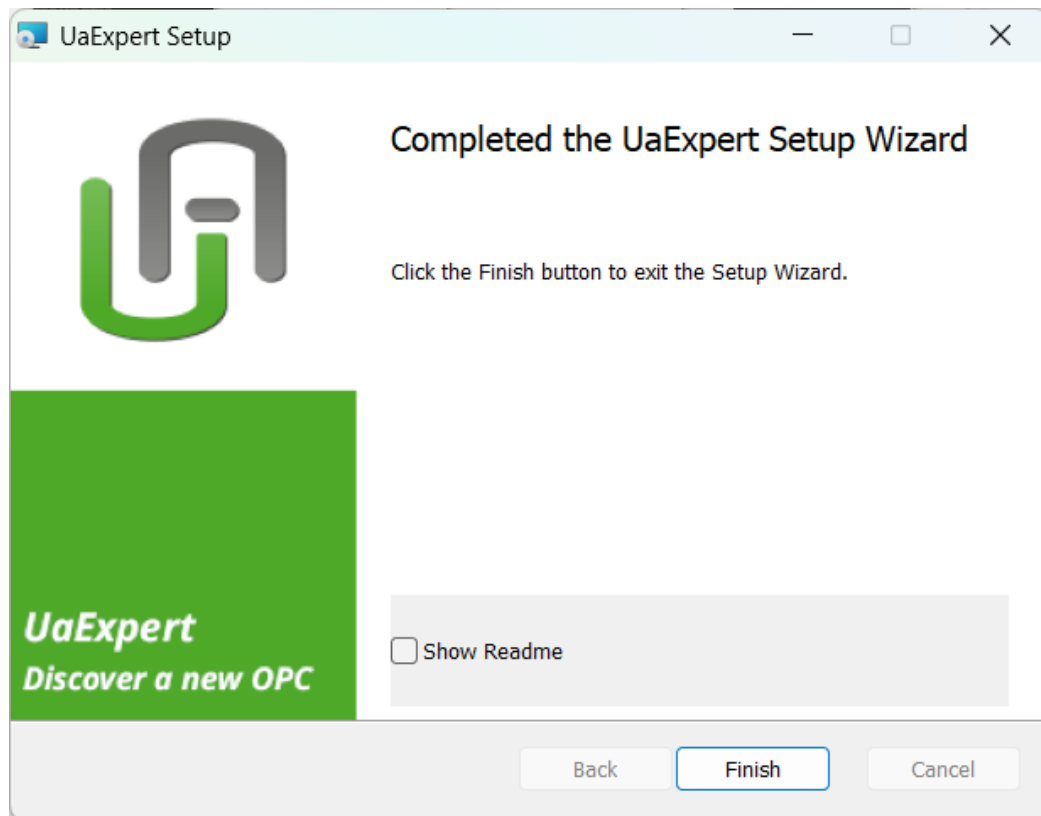


Figure 2-2 / 2-3 / 2-4 — The three installer pages that are relevant here. Keep the *Custom Setup* defaults; the defaults install UaExpert and the documentation into *C:\Program Files\UnifiedAutomation\UaExpert*.

1. On first launch the **New Application Instance Certificate** dialog appears; fill in **Common Name** and press **OK**

Figure 2-5 — This dialog only appears on the first launch. *Common Name* defaults to *UaExpert@<computer name>*, and the default is fine — the certificate is only how *UaExpert* identifies itself, and it has nothing to do with the QEC side* (the server runs Security None and validates no certificate). The red cross on *Organization* does not matter; leave it empty. The computer name is masked in this figure.*

This step happens once. The certificate lives in `%APPDATA%\unifiedautomation\uaexpert\PKI\`; only a new machine or a clean reinstall makes you repeat it.

2.2 Code Description

`QEC OPCUA_Server_HMI.ino` is about 720 lines and falls into five layers:

Layer	Representative Files / Functions	Function
Network Settings	<code>cfgLoad()</code> / <code>cfgSave()</code> / <code>cfgDefaults()</code> / <code>parseIp()</code> / <code>maskValid()</code> / <code>Ethernet.begin()</code>	EEPROM read/write and validation, static IP setup
HMI	<code>myhmi.h</code> / <code>myhmi.cpp</code> , <code>Hmi.*</code> , <code>qlog()</code> / <code>setStatus()</code> / <code>setCfgStatus()</code>	Three page layouts, touch keypad, on-panel log, status bar
Application Logic	<code>handleHmiEvents()</code> / <code>syncOpcToHmi()</code> / <code>tick1s()</code> / <code>doConfirm()</code> / <code>doDefault()</code> / <code>doReboot()</code>	Button handling, bidirectional sync, settings validation, restart
Task / Runtime	<code>setup()</code> / <code>loop()</code> / <code>helperYield()</code> / <code>BEGIN_HMI_EVENT_PROC</code>	Single-threaded non-blocking schedule: OPC UA every pass, 200 ms sync, 1 s tick
OPC UA Server	<code>UA_Server_new()</code> / <code>UA_ServerConfig_setMinimal()</code> / <code>buildAddressSpace()</code> / <code>addVar()</code> / <code>writeInt32()</code> / <code>readInt32()</code>	Server creation, node tree, node read/write

2.2.1 Network Settings

The network configuration lives in the first 18 bytes of **EEPROM bank 0 (EEPROM_16K)**:

addr	Content
0	magic 'Q'
1	magic 'C'
2	version (currently 1)
3–6	IP
7–10	subnet mask
11–14	gateway
15–16	port (high, low)
17	checksum = XOR(0..16)

At boot `cfgLoad()` validates magic, version and checksum. **If any check fails it falls back to the sketch's `DEF_* defaults`** and says so in the log (`no valid EEPROM config - using defaults`). A first flash, or a wiped EEPROM, therefore never leaves the unit unable to boot.

```
#define DEF_IP      "192.168.2.200"
#define DEF_MASK    "255.255.255.0"
#define DEF_GW      "192.168.2.1"
#define DEF_PORT     4840
```

The factory defaults are used only when EEPROM holds no valid data. Set them to the subnet you use most often on site.

2.2.2 HMI

Every log line goes to both the **Serial Monitor (115200)** and the **on-panel Log page**, stamped with local time:

```
static void qlog(const char *s) {
    Serial.println(s);
    if (!hmiReady) return;
    /* ...timestamp, write to the TextList... */
    if (logNext > LOG_LINES) { /* full: clear but leave a marker */
        /* ... */
    }
}
```

When the Log page fills up at 99 lines it is cleared and line 1 reads ----- log full - cleared and restarted -----, so you are never left with a handful of lines that could be mistaken for a log with few entries.

2.2.3 Application Logic

Interval	What happens
Every pass	<code>UA_Server_run_iterate(server, false)</code>
200 ms	<code>syncOpcToHmi()</code> : read <code>Command / Level</code> onto the panel, update <code>Sessions / Uptime / LED</code>
1 s	<code>tick1s()</code> : <code>Counter +1</code> , write <code>Uptime / Sessions</code> , update the panel clock
On event	<code>handleHmiEvents()</code> : page changes, input applied, <code>CONFIRM / DEFAULT / REBOOT</code>

`Setpoint` and `Analog` are written **only when the user confirms an entry** (`Hmi.numberInputApplied()`), not polled every pass, so typing does not stream partial values into the nodes.

2.2.4 Task / Runtime

```
void loop() {
    if (server && serverOk) UA_Server_run_iterate(server, false);

    unsigned long now = millis();
    if (now - lastFast >= 200) { lastFast = now; syncOpcToHmi(); }
    if (now - lastSlow >= 1000) { lastSlow = now; tick1s(); }

    BEGIN_HMI_EVENT_PROC
    {
        handleHmiEvents();
    }
    END_HMI_EVENT_PROC
}
```

`BEGIN_HMI_EVENT_PROC / END_HMI_EVENT_PROC` expand to an `Hmi.eventPop()` loop capped at 128 iterations, followed by a call to `helperYield()` — and **`helperYield()` is what refreshes the display**. Keep that in mind when writing any kind of wait: the panel does not update during `delay()`; to keep the screen alive you must call `helperYield()`.

2.2.5 OPC UA Server

The order of these calls in `setup()` matters; reordering any two of them causes a failure later in the sequence:

```
putenv("TZ=CST-8"); tzset(); /* [0] time zone, before any UA_DateTime_now() */
Hmi.begin(); /* [1] HMI */
cfgLoad(); /* [2] settings */
Ethernet.begin(mac, ip, dns, gw, mask); /* [3] static IP; shares the SWS stack with open62541 */
server = UA_Server_new();
UA_ServerConfig_setMinimal(cfg, cfgPort, NULL); /* [4] setMinimal() rather than setDefault() so the
configured port takes effect */
cfg->customHostname = UA_STRING_ALLOC(ipstr); /* publish the discovery URL as an IP, not the library
default host name */
cfg->accessControl.activateSession = ...; /* ... session-counting hooks ... */
buildAddressSpace(); /* ... [5] address space ... */
UA_Server_run_startup(server);
```

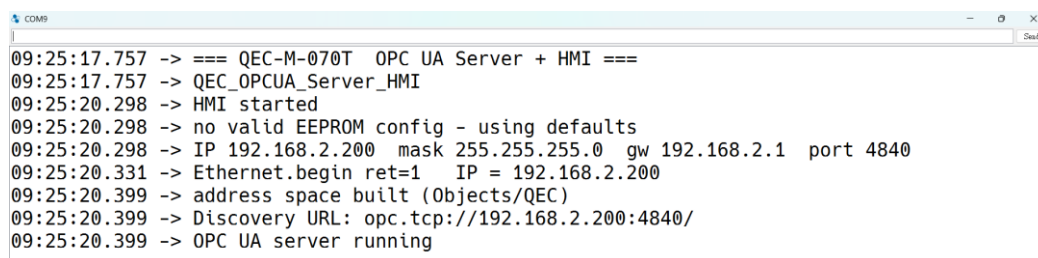
`addVar()` sets `attr.dataType` on every node:

```
attr.dataType = type->typeId; /* without this UaExpert shows BaseDataType */
```

2.3 Uploading the Code

1. Confirm that **DEF_IP** and the other **DEF_*** defaults match the subnet used on site
2. Press **Upload** (the → arrow) in the IDE
3. Compiling and uploading takes about **30–60 seconds**
4. Open the Serial Monitor (**115200**); within a few seconds you should see:

```
=== QEC-M-070T  OPC UA Server + HMI ===
QEC_OPCTUA_Server_HMI
HMI started
config loaded from EEPROM
IP 192.168.2.200 mask 255.255.255.0 gw 192.168.2.1 port 4840
Ethernet.begin ret=1 IP = 192.168.2.200
address space built (Objects/QEC)
Discovery URL: opc.tcp://192.168.2.200:4840/
OPC UA server running
```



```
09:25:17.757 -> === QEC-M-070T  OPC UA Server + HMI ===
09:25:17.757 -> QEC_OPCTUA_Server_HMI
09:25:20.298 -> HMI started
09:25:20.298 -> no valid EEPROM config - using defaults
09:25:20.298 -> IP 192.168.2.200 mask 255.255.255.0 gw 192.168.2.1 port 4840
09:25:20.331 -> Ethernet.begin ret=1 IP = 192.168.2.200
09:25:20.399 -> address space built (Objects/QEC)
09:25:20.399 -> Discovery URL: opc.tcp://192.168.2.200:4840/
09:25:20.399 -> OPC UA server running
```

Figure 2-6 - The boot messages in the Serial Monitor. *Ethernet.begin ret=1 (not 0) and Discovery URL: opc.tcp://<IP>:4840/ (an IP, not 86Duino) must both be visible, and the last line must read OPC UA server running. This particular unit had never had its settings written, so line 4 reads no valid EEPROM config - using defaults — see the note below. The first two lines appear about 2 s after reset (delay(2000)); the gap between them and the rest is Hmi.begin().*

Check	Must be
Ethernet.begin ret=	1 (0 means networking did not start — see §7)
IP =	the IP you configured, not 0.0.0.0
Discovery URL:	opc.tcp://<IP>:<port>/, not opc.tcp://86Duino:4840/
Last line	OPC UA server running

On a first flash the fourth line reads **no valid EEPROM config - using defaults instead**, which is what Figure 2-6 shows. Nothing is wrong: the EEPROM has never been written, so `cfgLoad()` falls back to the sketch's **DEF_*** values (§2.2.1) and the unit comes up on those. Press **CONFIRM** once on the Settings page and every boot after that reads **config loaded from EEPROM**. **If it keeps saying no valid EEPROM config after a CONFIRM, that is a real fault** — see §7.

Press Disconnect in UaExpert before uploading. Uploading resets the board; UaExpert's watchdog gives up after a few timeouts and will not reconnect on its own when the board comes back (see §7).

3. OPC UA Address Space and Node Contract

3.1 Node Table

Root			
└ Objects			
└ Server		(standard OPC UA nodes, NS0)	
└ QEC		(Object, ns=1;s=QEC)	
└ Setpoint	Int32	RO	<- panel out, read-only to clients
└ Analog	Double	RO	<- panel out, read-only to clients
└ Command	Int32	RW	<- written by the client
└ Level	Double	RW	<- written by the client
└ Counter	Int32	RO	
└ Uptime	Int32	RO	
└ Sessions	Int32	RO	
└ ModelName	String	RO	
└ SketchVersion	String	RO	
└ IPAddress	String	RO	

NodeId	Type	Access	Description
ns=1;s=QEC.Setpoint	Int32	RO	Entered in the panel's nin1 , read by the client
ns=1;s=QEC.Analog	Double	RO	Entered in the panel's nin2 (floating point), read by the client
ns=1;s=QEC.Command	Int32	RW	Written by the client, shown in the panel's num1
ns=1;s=QEC.Level	Double	RW	Written by the client, shown in the panel's num2 to 4 decimal places
ns=1;s=QEC.Counter	Int32	RO	Increments every second. Subscribe to it and you can watch DataChange notifications arrive
ns=1;s=QEC.Uptime	Int32	RO	Seconds since boot (millis() wraps at ~49.7 days, at which point this restarts from 0)
ns=1;s=QEC.Sessions	Int32	RO	Currently connected client sessions
ns=1;s=QEC.ModelName	String	RO	"QEC-M-070T"
ns=1;s=QEC.SketchVersion	String	RO	"QEC_OPCTUA_Server_HMI" — the value of QEC_SKETCH_VERSION , which carries no version number
ns=1;s=QEC.IPAddress	String	RO	The IP actually bound. If it differs from the Settings page, the change has not been applied yet

Server specification

Item	Value
Endpoint	opc.tcp://<QEC_IP>:<port> (port is configurable on the panel; default 4840)
SecurityPolicy	None (UA_ENABLE_ENCRYPTION not compiled in)
MessageSecurityMode	None
UserTokenType	Anonymous
ApplicationUri	urn:open62541.server.application
Subscription / DataChange	Supported
Method call	Supported
Discovery (ServersOnNetwork)	Not compiled in — the endpoint URL must be entered manually
Events / ModelChangeEvents	Not compiled in — re-browse manually after changing nodes

3.2 Access Direction

Direction is part of the contract. **Setpoint** and **Analog** are read-only to clients; **Command** and **Level** are the ones clients write:

Direction	Nodes	Source of truth
HMI → OPC UA	Setpoint (Int32), Analog (Double)	The panel. Clients may only read
OPC UA → HMI	Command (Int32), Level (Double)	The client. The panel only displays
System status	Counter / Uptime / Sessions / ModelName / SketchVersion / IpAddress	The firmware. Clients may only read

Why are **Setpoint and **Analog** read-only?** The panel never reads these two nodes back (`syncOpcToHmi()` reads only **Command** and **Level**). If clients could write them, a SCADA writing 999 to **Setpoint** would leave the node at 999 while the panel still shows the old value — inconsistent, and with no indication. Read-only turns that into an immediate **BadNotWritable** instead of a write that appears to succeed while the panel ignores it.

Both data types still have a writable node (**Command** and **Level**), so write testing is unaffected.

Note: `UA_Server_writeValue()` is a *local* server-side write and is not subject to `accessLevel`, so the firmware still writes RO nodes freely — **QEC.Counter** has always been RO and still increments every second.

3.3 What you see once a client connects

Any standard OPC UA client that connects and expands **Objects/QEC** gets the ten nodes above. Three of them are worth a second look, because they are the direct evidence that the path really is open:

- **The DataType is a concrete type.** Select **Setpoint** and its Attributes read **Int32**, not **BaseDataType** — your upstream system never has to guess how to interpret a value.
- **Counter updates without any client action.** Drag it into the Data Access View: the value increments every second and the timestamp follows, which means subscription DataChange notifications are being delivered.
- **The panel reacts at the same time.** The **Client** LED lights fully and **Sessions** becomes 1 — the connection state is visible on the device itself, with no need to go back and check the client.

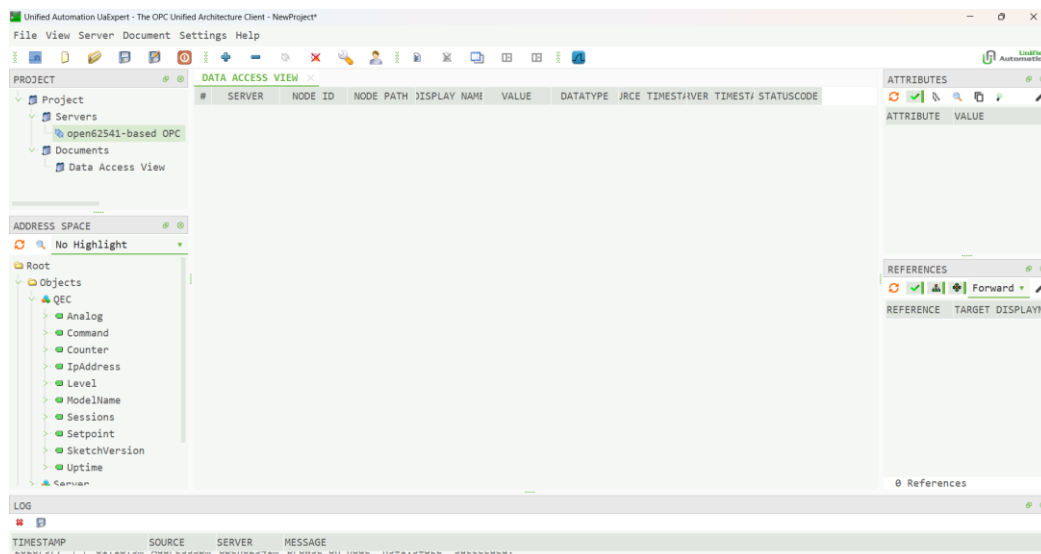


Figure 3-1 — UaExpert's address space with Root / Objects / QEC expanded, showing all 10 nodes.

4. HMI Page Function Description

Three pages, Material / Light theme, designed for 800×480.

4.1 Main — bidirectional data flow

The left half is the two directions of data flow, the right half is connection information: the endpoint, the Client LED, Sessions and Uptime.

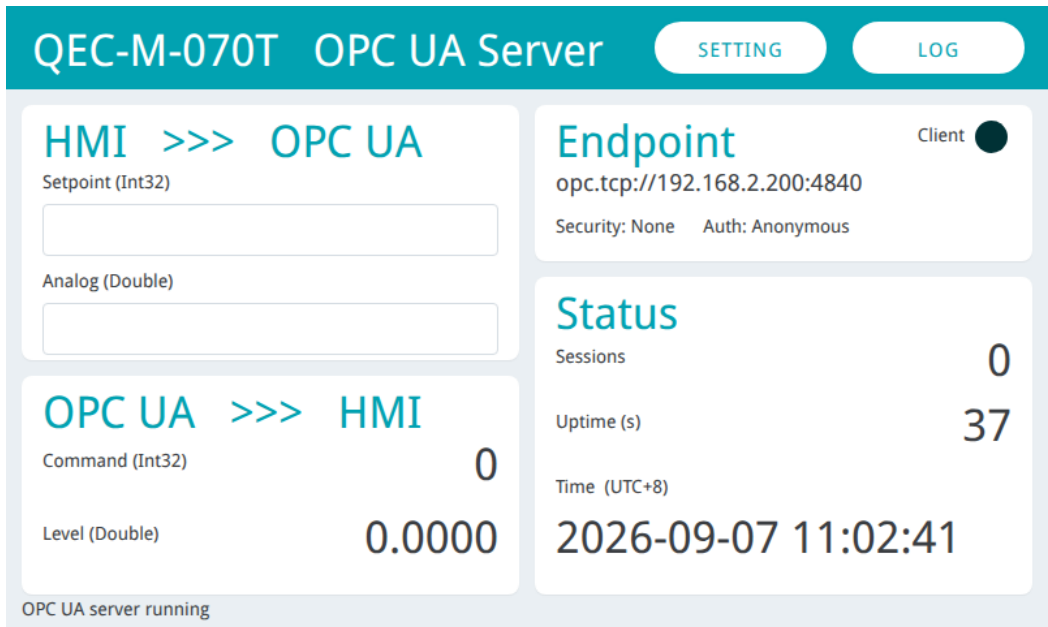


Figure 4-1 — Main page, not connected: the Client LED dimmed, Sessions = 0.

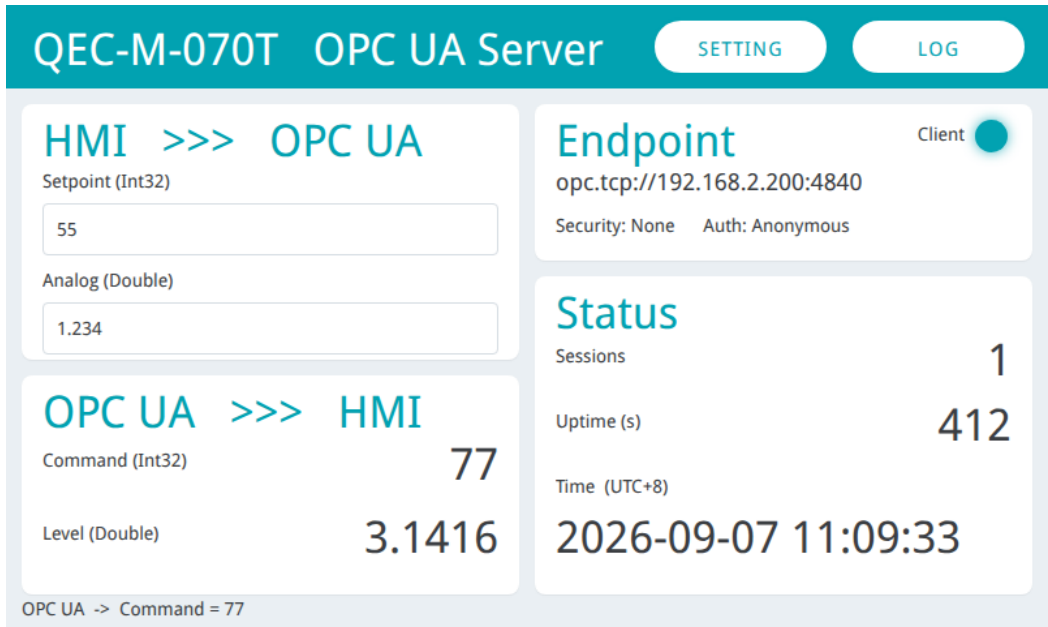


Figure 4-2 — Main page, connected: LED fully lit, Sessions = 1, Setpoint = 55, Analog = 1.234, Command = 77, Level = 3.1416, status bar reading OPC UA -> Command = 77.

Field / Button	Function
Setpoint (Int32) input	Tap to open the touch keypad, type, confirm — written to QEC.Setpoint immediately
Analog (Double) input	Same, floating point, written to QEC.Analog
Command (Int32) display	Current value of QEC.Command, refreshed every 200 ms

Field / Button	Function
Level (Double) display	Current value of <code>QEC.Level</code> , to 4 decimal places
Endpoint	Filled in at boot with the IP and port actually in effect — not a hard-coded string
Client + LED	Fully lit while a client is connected, dimmed when none is
Sessions	Number of connected clients
Uptime (s)	Seconds since boot
Time (UTC+8)	Local time; the zone label is computed from <code>TZ</code>
Status bar (bottom)	The most recent event, matching the last line of the Log page. 769 px wide, so long messages are not truncated
SETTING	Go to the Settings page (and refill the fields from the stored settings , discarding edits that were never CONFIRMed)
LOG	Go to the Log page

Example event messages

```
HMI -> Setpoint = 55
HMI -> Analog = 1.2340
OPC UA -> Command = 77
OPC UA -> Level = 3.1416
client connected (sessions = 1)
client disconnected (sessions = 0)
```

A failed write appends **FAILED** to the message and logs `write QEC.Setpoint failed: <StatusCode>`.

4.2 Log — a live console on the panel

Everything the device has said, in sync with the Serial Monitor.

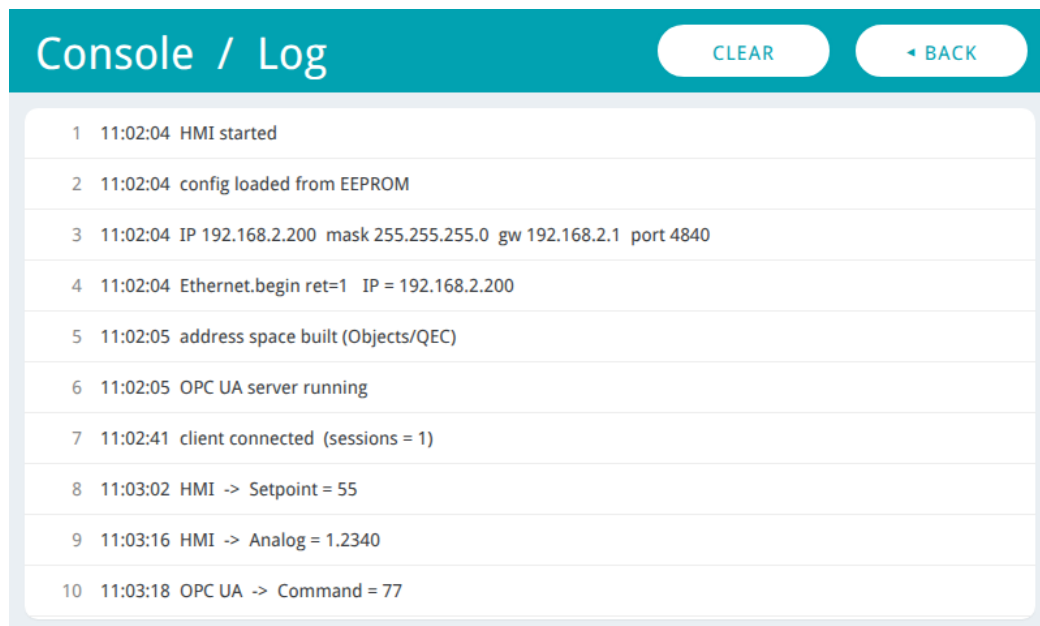


Figure 4-3 — The Log page showing the boot sequence plus client connected (sessions = 1) and one HMI -> Setpoint entry.

Field / Button	Function
TextList (<code>p2t11</code>)	99 numbered lines, auto-scrolled to the newest entry
When full	Cleared, with ----- log full - cleared and restarted ----- written to line 1

Field / Button	Function
CLEAR	Clear and restart from line 1
BACK	Return to the Main page

The content matches the Serial Monitor — with the exception of the two boot lines (=== ... === and the sketch name), which are printed before `Hmi.begin()` and so appear on serial only. When the panel is all you have on site, this page is your only debug window.

4.3 Settings — change IP or port without reflashing

The left column holds the values stored in EEPROM, the right column **Active now** is what is really running — side by side they show whether a change has taken effect.

Settings [← BACK](#)

Network Configuration

		Active now
IP Address	192.168.2.200	IP 192.168.2.200
Subnet Mask	255.255.255.0	Mask 255.255.255.0
Gateway	192.168.2.1	GW 192.168.2.1
OPC UA Port	4840	Port 4840

Stored in EEPROM - press REBOOT (or power-cycle) to apply

[DEFAULT](#) [REBOOT](#) [✓ CONFIRM](#)

Figure 4-4 — The Settings page as first entered: the left column and the Active now column match.

Settings [← BACK](#)

Network Configuration

		Active now
IP Address	192.168.2.210	IP 192.168.2.200
Subnet Mask	255.255.255.0	Mask 255.255.255.0
Gateway	192.168.2.1	GW 192.168.2.1
OPC UA Port	4840	Port 4840

Stored in EEPROM - press REBOOT (or power-cycle) to apply

[DEFAULT](#) [REBOOT](#) [✓ CONFIRM](#)

Saved: 192.168.2.210:4840 - press REBOOT to apply

Figure 4-5 — The left column changed to a new IP while Active now still shows the old values, status bar reading Saved: ... - press REBOOT to apply. The clearest illustration of "saved but not yet applied".

Field / Button	Function
IP Address / Subnet Mask / Gateway	TextInput; tap to open the touch keypad
OPC UA Port	NumberInput, 1–65535
Active now (right column)	The values actually in effect since boot. A mismatch with the left column means the settings were changed but not yet applied
CONFIRM	Validate, then write to EEPROM
DEFAULT	Restore the sketch's DEF_* factory values and write them. Press twice within 3 s
REBOOT	Soft-restart so the settings take effect. Press twice within 3 s
Status message (bottom)	Validation failure reasons, save results, double-press prompts
BACK	Return to the Main page

CONFIRM validation rules

If any check fails, the reason is shown and **nothing is written to EEPROM**:

Check	Failure message
IP is four octets 0–255	IP address format error
Subnet is four octets 0–255	Subnet mask format error
Gateway is four octets 0–255	Gateway format error
Port is 1–65535	Port must be 1 - 65535
First IP octet is not 0	IP address must not start with 0
Mask must be contiguous	Subnet mask must be contiguous, e.g. 255.255.255.0
IP is not the network address	IP is the network address of this subnet
IP is not the broadcast address	IP is the broadcast address of this subnet

Figure 4-6 — After entering 255.0.255.0 as the subnet and pressing CONFIRM, the status bar reads Subnet mask must be contiguous, e.g. 255.255.255.0.

Mask contiguity is checked bitwise (a mask such as 255.0.255.0 is rejected; stored and applied it would take the

whole device off the network). The network and broadcast addresses are derived **from the mask**, so **192.168.5.0** remains valid under a /16.

A gateway outside the subnet is flagged but not blocked (bench setups often need no gateway); the confirmation becomes **Saved: ... - WARNING gateway not in this subnet.**

Double-press confirmation

DEFAULT and **REBOOT** both overwrite site settings or drop live connections, so both require **two presses within 3 seconds**:

First press -> status bar: Press REBOOT again within 3 s to restart the device
Second press -> executed

Settings

◀ BACK

Network Configuration

Active now

IP Address	192.168.2.200	IP	192.168.2.200
Subnet Mask	255.255.255.0	Mask	255.255.255.0
Gateway	192.168.2.1	GW	192.168.2.1
OPC UA Port	4840	Port	4840

Stored in EEPROM - press REBOOT (or power-cycle) to apply

DEFAULT

REBOOT

✓ CONFIRM

Press REBOOT again within 3 s to restart the device

Figure 4-7 — After one press of REBOOT, the status bar reads Press REBOOT again within 3 s to restart the device.

The two buttons are mutually exclusive: pressing **REBOOT** clears any armed **DEFAULT** and vice versa, and **CONFIRM** clears both. Entering the Settings page also resets both.

5. Walkthrough: First Data Exchange

5.1 What this chapter does

Using **UaExpert** on a PC as the upstream system, follow the shortest route: **connect** → **see the nodes** → **write one value in each direction**. It takes about ten minutes and does not involve writing any code.

Doing it once establishes three things: the QEC is reachable on your network as a standard OPC UA server, the data types it publishes are correct, and data moves in both directions. **Whether you go on to connect a SCADA system, an MES, or your own program makes no difference to the QEC** — they all speak the same standard interface.

UaExpert is simply the most convenient thing to demonstrate with. If you already have a SCADA or MES, or a client of your own written with Python's `opcua` or the C# OPC Foundation SDK, point it at the same endpoint — the node names are identical.

5.2 Before you start

1. The QEC MDevice is flashed with `QEC_OPCTUA_Server_HMI` and serial shows `OPC UA server running`
2. The cable is in the **Gigabit RJ45** port (not one of the two 10/100 EtherCAT ports)
3. The PC and the QEC are on the **same subnet** (check with `ipconfig` on the PC)
4. UaExpert is installed on the PC (installation steps in §2.1.3)

5.3 Steps

#	Step	Action
1	Install UaExpert	Register on the Unified Automation site → Downloads → OPC UA Clients → UaExpert → download and install
2	Create a certificate	On first launch it asks for an Application Instance Certificate ; fill in the Common Name and generate it
3	Open Add Server	Server → Add...
4	Enter the endpoint	Expand Custom Discovery → double-click < Double click to Add Server... > → type <code>opc.tcp://192.168.2.200:4840</code> → Enter
5	Expand and pick the security setting	Click the > at the left of that row and select None - None (uatcp-uasc-uabinary) beneath it
6	Pick the authentication	Set Authentication Settings to Anonymous → OK
7	Connect	Right-click the server in the project tree → Connect
8	Confirm the connection	UaExpert logs changed to 'Connected' ; on the QEC panel the LED is fully lit and Sessions = 1
9	Browse the nodes	Expand Root / Objects / QEC in the address space and confirm 10 nodes
10	Drag into the DA View	Drag Setpoint , Analog , Command , Level and Counter into the Data Access View

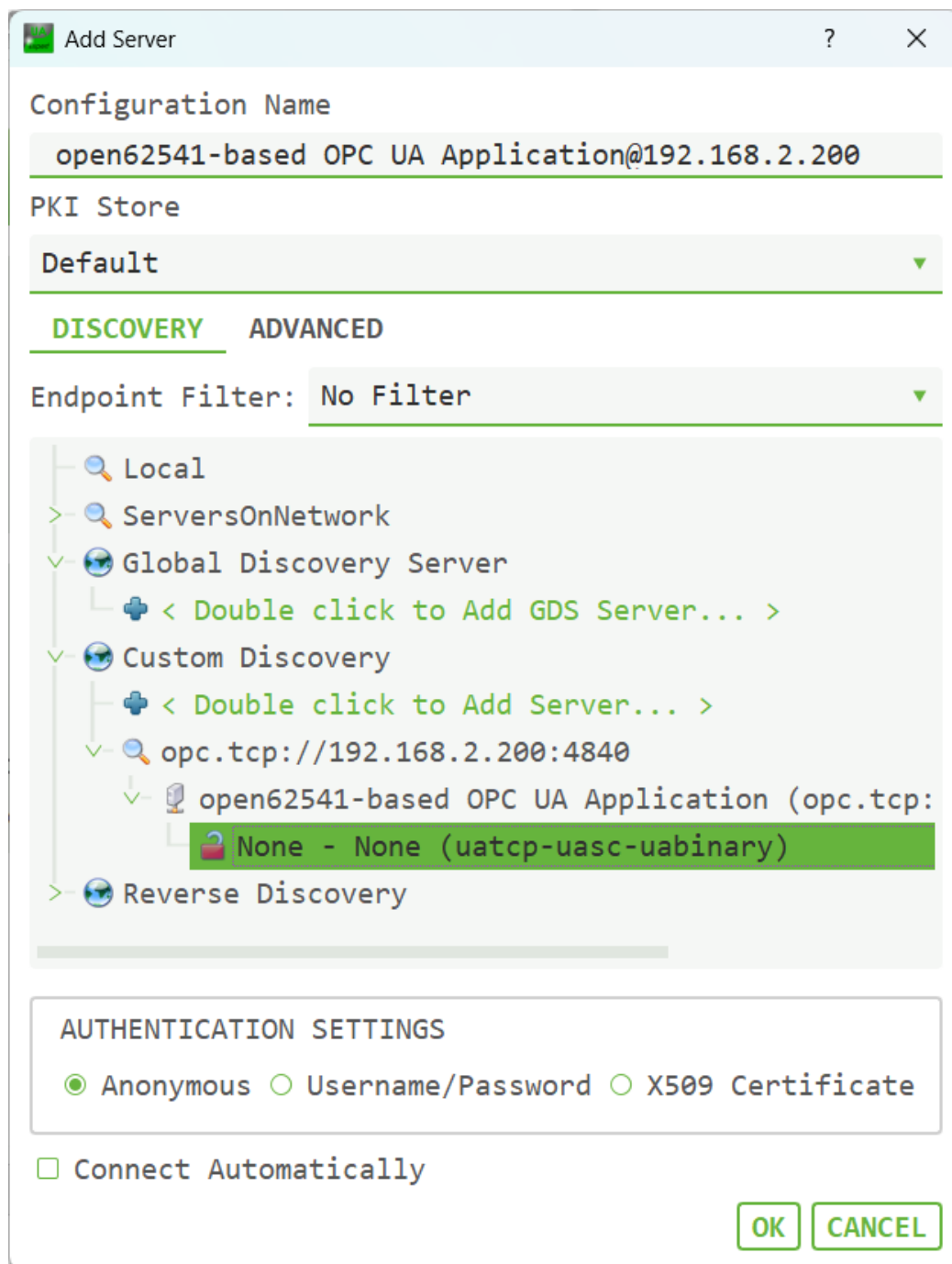


Figure 5-1 — The Add Server dialog with the endpoint expanded, None - None (uatcp-uasc-uabinary) selected and Authentication set to Anonymous. Note the endpoint is entered as an IP.

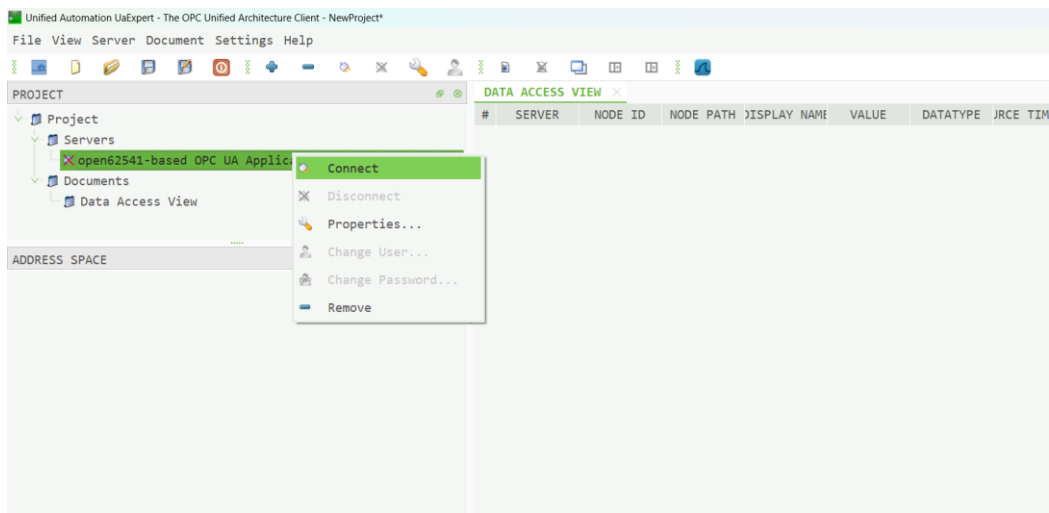


Figure 5-2 — The right-click → Connect context menu on the server.

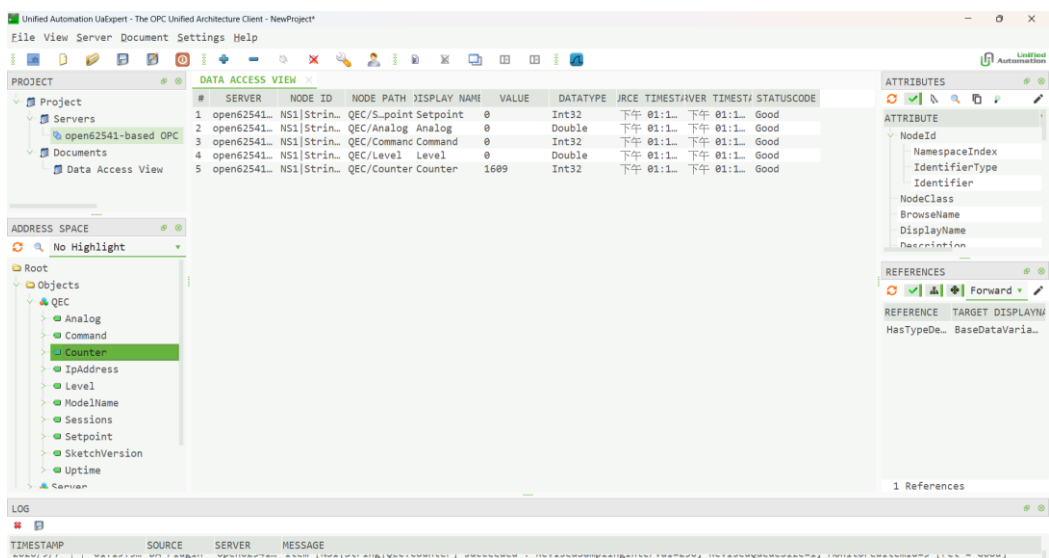


Figure 5-3 — Connected, with five rows in the DA View (including Counter ticking), DataType showing Int32 / Double and StatusCode all Good. Counter is at 1609 here and advances once a second.

Step 5 is the one that most often goes wrong. With only the URL row selected the **OK** button stays greyed out; you must expand the row and select the **None - None (uatcp-uasc-uabinary)** entry beneath it. Also, a **Local Network** scan will **never** find this device (discovery is not compiled in), so always enter the URL by hand.

5.4 Once connected

UaExpert logs **changed** to 'Connected' and the address space grows an **Objects/QEC** branch. At the same moment the QEC panel lights its **Client** LED, **Sessions** becomes 1, and the Log page adds **client connected (sessions = 1)**. Both ends reacting together is what a working connection looks like.

5.5 One write in each direction

This section covers two actions, one in each direction. Together they verify the complete path.

Panel → upstream system

Enter **55** in the **Setpoint** field on the Main page and confirm. **QEC/Setpoint** in UaExpert's DA View immediately becomes 55, and the panel status bar reads **HMI -> Setpoint = 55**. For a float, use **Analog** and enter **1.234**.

Upstream system → panel

Write **77** to **QEC/Command** from UaExpert. The panel shows **Command = 77** within 200 ms and the status bar reads **OPC UA -> Command = 77**. For a float, write **3.1416** to **QEC/Level**; the panel displays it to four decimal places.

Things you will notice along the way

What you see	What it means
Counter increments every second, timestamp follows	Subscription DataChange delivery is working
The source timestamp matches the panel's Time , with no 8-hour offset	The time zone is set correctly (TZ=CST-8)
Writing QEC/Setpoint is refused with BadNotWritable	Direction protection is in force (§3.2). Expected, not a fault
Sessions returns to 0 after Disconnect in UaExpert	Connection state is visible on the device itself
After a power cycle the log reads config loaded from EEPROM	Network settings live in EEPROM and survive (§6)

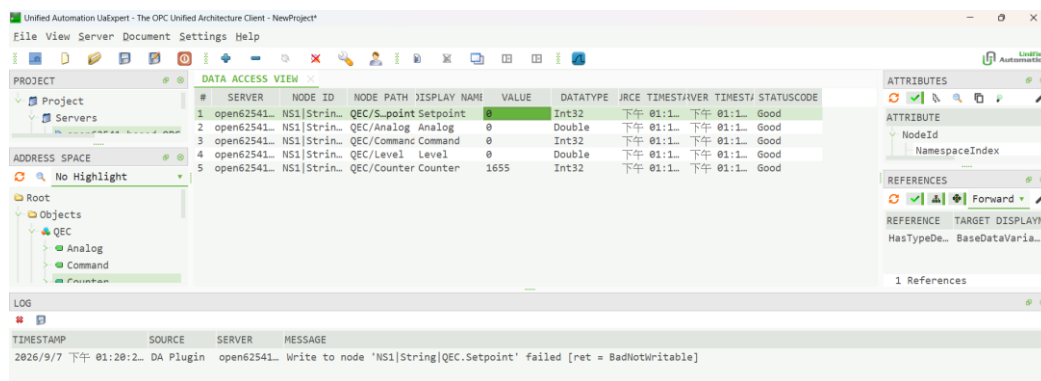


Figure 5-4 — Writing to the read-only **QEC/Setpoint** returns **BadNotWritable**. A write in the wrong direction is refused immediately rather than appearing to succeed while the panel ignores it.

At this point the QEC's OPC UA interface is confirmed working. The node names, types and directions are in §3, and adding a node of your own is a single line in **buildAddressSpace()** (§9).

5.6 These messages are normal

First-time users usually read the following as faults. None of them are:

Message	What it means
The OK button in Add Server is greyed out	Only the discovery URL row is selected. Click > and select None - None (uatcp-uasc-uabinary) beneath it
A Local Network scan does not find the QEC	Expected — discovery is not compiled in. Enter the endpoint URL manually
The server returned no certificate	Expected — SecurityPolicy is None. Ignore it
Register for ModelChangeEvents failed: BadNotSupported	Expected — event subscriptions are not compiled in. One consequence only: re-browse manually after changing nodes

6. Network Configuration Persistence

Changing the network settings requires no recompile, no reflash, and no power cycle at the breaker. When a site moves to a different subnet, this is where the QEC differs most from a device with its IP fixed in firmware.

Item	Purpose	Operation
EEPROM configuration block	Keep IP / mask / gateway / port across power cycles	First 18 bytes of bank 0, with magic + version + XOR checksum; any mismatch falls back to <code>DEF_*</code>
Active now column	Separate "saved but not yet applied" from "actually running"	The right column of the Settings page; shows the values in effect since boot, refilled by <code>activeToUi()</code> each time the page is opened
REBOOT button	Complete the save-then-apply loop on the panel itself	Two presses within 3 s

6.1 Why a restart is required

The OPC UA server binds to the IP during `UA_Server_run_startup()`, and both `customHostname` and the discovery URL are fixed at that moment. Changing the IP would mean reinitialising the SWS stack (`SwsSock::init()` internally calls `SWS_SockCleanup()` first) and rebuilding the whole server — too risky inside a single-threaded `loop()`.

The design is therefore **save** → **restart** → **apply**, which is the approach least likely to tear down a live connection unexpectedly.

6.2 Why the Port Can Be Changed on the Panel

The standard `UA_ServerConfig_setDefault()` hard-codes port 4840. This project calls the underlying `setMinimal()` directly and passes the configured value:

```
UA_ServerConfig_setMinimal(cfg, cfgPort, NULL);
```

7. Troubleshooting

Issue	Possible Cause	Solution
Compile error 'p3b4' was not declared in this scope	You edited the <code>data_project.hpj</code> layout without saving in the 86HMI designer — <code>myhmi.h/cpp</code> are still the old ones and lack the new objects	Open <code>data_project.hpj</code> in the designer, save once to regenerate <code>myhmi.h/cpp</code> , then compile. You will not hit this unless you edit the layout (both files ship with the project)
Serial shows <code>IP = 0.0.0.0, Ethernet.begin ret=0</code>	Cable in the wrong port / IP conflict / bad EEPROM contents	Confirm the Gigabit port; pick an unused IP; on the panel SETTING → DEFAULT (×2) → REBOOT (×2)
Discovery URL: shows 86Duino instead of an IP	<code>customHostname</code> was not set	Confirm the <code>.ino</code> sets <code>cfg->customHostname = UA_STRING_ALLOC(ipstr)</code> before <code>run_startup()</code>
Boot stops on the Log page showing <code>run_startup FAILED</code>	IP conflict / port in use / cable not in the Gigabit port	The panel buttons still work: BACK → SETTING to fix the settings or press DEFAULT → REBOOT . No reflash needed
The OK button in UaExpert's Add Server is greyed out	Only the discovery URL row is selected	Click > and select None - None (uatcp-uasc-uabinary) beneath it
UaExpert connects but browsing is empty / it reconnects repeatedly	The discovery URL is not an IP	See the third row above
<code>BadNodeIdUnknown</code>	UaExpert is still using the node set it cached in an earlier session	Disconnect → Remove the server → add it again; also delete and re-drag stale DA View rows
A newly added node does not appear in UaExpert	Events / ModelChangeEvents are not compiled in	Re-browse manually
The source timestamp never updates	Expected — subscriptions are report-by-exception	Subscribe to <code>QEC.Counter</code> or <code>Server/ServerStatus/CurrentTime</code>
UaExpert timestamps are 8 hours off	TZ unset or wrong	<code>#define QEC_TZ "TZ=CST-8"</code> at the top of the <code>.ino</code> (Taipei is UTC+8, POSIX writes <code>-8</code>); use <code>"TZ=UTC0"</code> if the RTC keeps UTC
After re-uploading, UaExpert will not reconnect and logs <code>BadTimeout</code> repeatedly	Expected — uploading resets the board and UaExpert's watchdog gives up	Right-click Connect; or Disconnect → wait 3 s → Connect; or Remove and add again. Always disconnect before uploading
Settings changed but nothing happened	No restart	Press REBOOT (twice within 3 s), or power-cycle. Compare against the Active now column
Cannot connect after changing the port	UaExpert is still using the old port	Change the endpoint to <code>opc.tcp://<IP>:<new port></code>
REBOOT does nothing	This IDE version's port-IO behaviour differs	Power-cycle the device manually; nothing else is affected
Every boot reports <code>no valid EEPROM config</code>	The EEPROM write did not complete, or the checksum does not match	Press CONFIRM once on the Settings page; if it still fails the EEPROM may be faulty
Status bar messages are truncated	The designer shrank that label to fit its text	See the warning in §8; confirm <code>p1116</code> is still at least 700 px wide

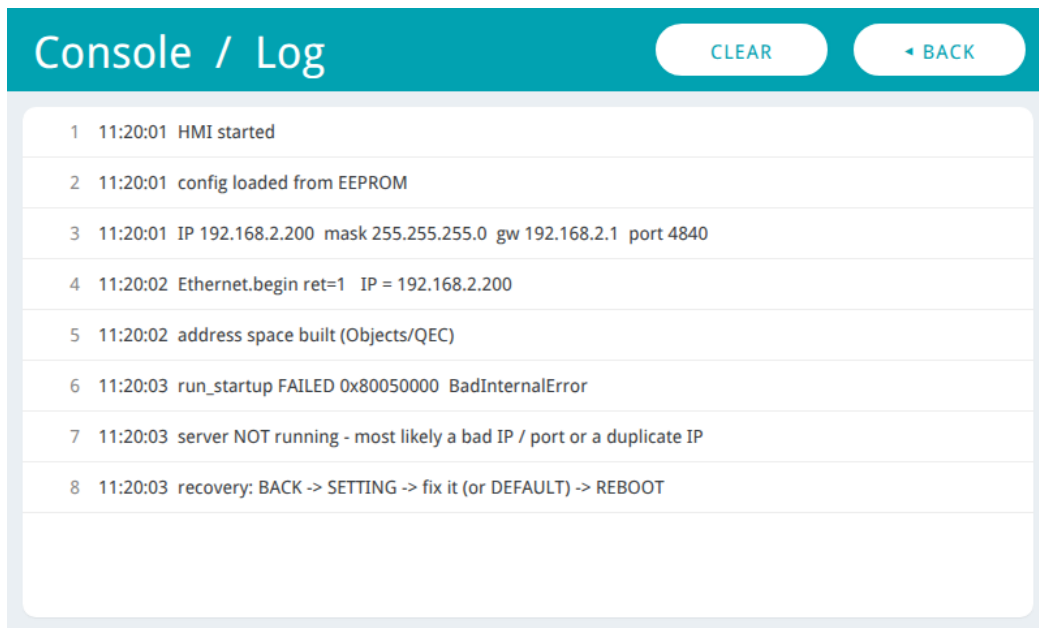


Figure 7-1 — What a failed `run_startup()` looks like: the panel jumps to the Log page, names the status code, and prints the recovery path. The buttons still respond, so an incorrect IP never leaves the panel unusable with reflashing as the only recovery.

8. Complete Code and Downloads

QEC OPCUA_Server_HMI\ ├─ QEC OPCUA_Server_HMI.ino ├─ README.md ├─ CHANGELOG.md ├─ myhmi.h ├─ myhmi.cpp ├─ data\ └─ _project.hpj	Main program (~720 lines; this is the file you edit) Developer notes (node table, the five library limitations and workarounds) Every design decision and measurement behind the code Generated by the 86HMI designer (do not edit) Generated by the 86HMI designer (do not edit) HMI layout project (3 pages; edit in the designer)
--	---

File	Description
QEC OPCUA_Server_HMI.ino	All application logic. The five-layer structure is described in §2.2
data_project.hpj	The HMI layout. Edit layouts in the designer, never <code>myhmi.*</code> directly
README.md	Includes the five limitations of the 86Duino port of open62541 and this project's workarounds (static IP, discovery URL, time zone, <code>attr.dataType</code> , session counting). Not a single line of library source was modified
CHANGELOG.md	Includes the field notes, such as the designer overwriting label widths

⚠ **Read this before editing the HMI layout: the designer overwrites label widths.** Setting a label's `Width` in the `.hpj` JSON has no effect — on save, the 86HMI designer re-auto-fits every label's width to its *design-time text* (measured at roughly 8.15 px per character). The placeholder strings in this project are therefore deliberately set to the *longest realistic* value: the status bar `p1116` carries a 94-character placeholder, which auto-fits to **769 px** on save. **After editing the text of these labels, check that the width has not shrunk**, or long runtime messages will be truncated.

Remember to save after editing the layout, so `myhmi.h` / `myhmi.cpp` are regenerated.

Example-code ZIP download: https://www.qec.tw/wp-content/uploads/2026/09/QEC OPCUA_Server.zip

9. Next Steps

With the QEC's OPC UA interface confirmed working, the next step is connecting it to your own system. The usual next steps are listed below, with the part of the project each one affects.

9.1 Connect your own upstream system

This step usually requires no change on the QEC side at all. Any standard OPC UA client — your SCADA, MES or historian — connects by entering the same endpoint; if you are writing the client yourself, Python's `opcua`, the C# OPC Foundation SDK and Node-RED's OPC UA nodes all work.

Only three items have to be passed to the client side:

Item	Value
Endpoint	<code>opc.tcp://<the QEC's IP>:<port></code> — copy the line from the panel's Main page
Security	SecurityPolicy None , MessageSecurityMode None , Anonymous
Nodes	<code>ns=1;s=QEC.<name></code> , listed in §3.1

9.2 Replace the nodes with your own data

The ten nodes are demonstration data. To expose what the machine actually produces, one line in `buildAddressSpace()` gives you a new node:

```
addVar("QEC.MyValue", "MyValue", &UA_TYPES[UA_TYPES_FLOAT], &f0, RW);
```

Three habits from the existing nodes are worth keeping: **set `attr.dataType` explicitly** (otherwise the upstream system sees `BaseDataType`), **decide the direction before choosing RO or RW** (the reasoning in §3.2 applies to your nodes too), and **keep the `QEC.` prefix** so the tree stays easy to group later.

Once there are more than a handful, group them — for example `DeviceInfo` / `Motor1` / `IO` objects under `Objects/QEC`.

9.3 Turn the QEC into a client instead

If the server you need to read belongs to **another device** (a third-party PLC, a PC, an instrument), the roles reverse. That side emphasises completely different things — the connection is asynchronous and the nodes are defined by the other end, so the tool needs to **browse the other side's node tree on the panel itself**. See [QEC OPC UA Client Test Tool - User Manual](#).

Two QEC units paired is the least troublesome setup: one running the server, one running the client, with no PC in between.

9.4 Coexisting with EtherCAT

This is exactly why `loop()` was written non-blocking from the start, and the room is already there: update EtherCAT after each `UA_Server_run_iterate()`, publish PDO values into nodes and turn node values into target positions.

Keep the motion-related nodes read-only in the first iteration; open them for writing once the data flow and cycle time are proven stable.

9.5 Security, before this goes live

This release does not compile `UA_ENABLE_ENCRYPTION`, so SecurityPolicy is None and authentication is Anonymous — **anyone who can reach this IP can read and write the nodes**. That is fine on a test bench or a closed production

segment, but isolate the device with a firewall or a dedicated subnet before it touches an office network or the outside world.

10. Contact Us

Sales Inquiry	info@qec.tw
Technical Support	info@icop.com.tw
Website	https://www.qec.tw

QEC OPC UA Server Test Tool - User Manual v 1.0 ICOP Technology Inc.