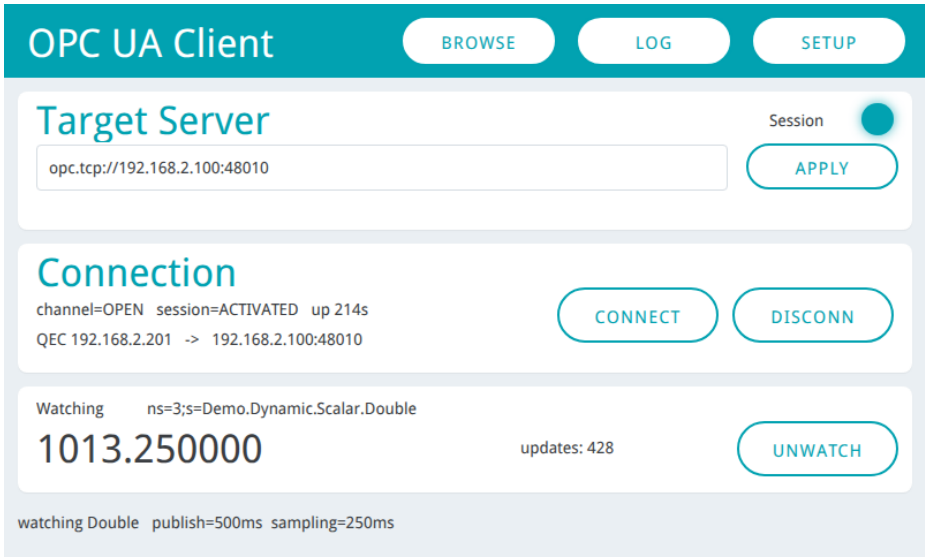


QEC OPC UA Client Test Tool — User Manual v 1.0

Use an 86Duino to connect to any OPC UA server and browse, read, write and subscribe to its nodes right on the panel

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



Cover The Home page: a session is running and one node is subscribed.

Revision History

Version	Date	Notes
1.0	September 2026	First release. Covers the QEC_OPCTUA_CLIENT_HMI release sketch: non-blocking connect, an on-panel node browser with live values, type-aware writes, favourites.

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	6
2.1 PC-Side Installation	6
2.1.1 86Duino Coding IDE 502	6
2.1.2 open62541 library	6
2.1.3 UaCPPServer (optional test server)	6
2.2 Code Description	8
2.2.1 Network Settings	9
2.2.2 HMI	9
2.2.3 Application Logic	9
2.2.4 Task and Runtime	10
2.2.5 OPC UA Client	10
2.3 Uploading the Code	11
3. Connecting to a Server and Working with Nodes	13
3.1 The Endpoint	13
3.2 Browsing the Address Space	13
3.3 Node Attributes	14
3.4 Supported Write Types	14
3.5 Subscription	14
4. HMI Page Reference	15
4.1 Home — the Connection	15
4.2 Browse — Finding Nodes	16
4.3 Node — Using a Node	18
4.4 Log — a Live Console on the Panel	20
4.5 Setup — the QEC's Own Network and Subscription Intervals	21
CONFIRM Validation Rules	21
Double Confirmation	22
5. Running It Once: Your First Server Connection	23
5.1 What This Chapter Does	23
5.2 Before You Start	23
Choosing a Server to Test Against	23
5.3 Steps	24
5.4 Once Connected	24
5.5 Four Things to Try	24
Things You Will Notice Along the Way	25
5.6 These Messages Are Normal	25
6. Configuration Persistence	26
6.1 What Needs a Restart and What Does Not	26
6.2 Why the Target URL Lives on the Home Page	26
6.3 EEPROM Layout Version Changes	26
7. Troubleshooting	27
8. Complete Code and Downloads	28
9. Next Steps	29
9.1 Against an Existing Server on Site	29
9.2 Two QECs Paired	29
9.3 The QEC as a Server Instead	29
9.4 Possible Extensions	29
9.5 Security Before You Go Live	29
10. Contact Us	30

1. Overview

QEC_OPCUA_CLIENT_HMI is an OPC UA client test tool that runs on any **QEC-M-series MDevice with a built-in panel**. Flash it, type the target endpoint on the panel and press CONNECT, and you can browse the server's address space, read and write values and subscribe to nodes — **with no PC involved at any point**.

It is the mirror image of the Server test tool: there, the QEC waits for SCADA to connect in; here, the QEC connects out.

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

Part	Implementation	Notes
OPC UA client	open62541 v1.1.2 (86Duino / DMP port)	Async connect, Browse, Read, Write, Subscription
HMI panel	Touch panel, LVGL86 + the 86HMI designer	Five pages: Home / Browse / Node / Log / Setup
Configuration persistence	EEPROM bank 0	The QEC's own network, target URL, 6 favourites, subscription intervals

No PC-side OPC UA client, no laptop at the machine, no separate HMI.

There are three reasons this tool exists:

1. To prove the QEC can reach a third-party OPC UA server on the plant network
2. **To find out what is actually inside that server** — with no documentation and no laptop available, browse the address space directly on the panel
3. To verify node data types, access rights and subscription behaviour before writing any application code of your own

1.1 System Architecture

1.1.1 Hardware Requirements

Component	Recommended	Notes
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); other panel models in the series work the same way
Network	1× Gigabit RJ45	OPC UA goes through this port. The other 2× 10/100 RJ45 are EtherCAT only , and the wrong port means no connection
OPC UA server	Any server offering an opc.tcp endpoint	A demo server on a PC, a third-party PLC, or another QEC running the Server test tool
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	Notes
86Duino Coding IDE	502	The version announced on the QEC site
open62541 library	v1.1.2	Installed from the IDE's Library Manager (search OPC)
86HMI designer	Bundled with the IDE	Generates myhmi.h / myhmi.cpp

Component	Version	Notes
An OPC UA server to test against	—	Usually Unified Automation's free demo server; the example in this manual is <code>opc.tcp://192.168.2.100:48010</code>

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-M-070T"** in the boot banner is also a hard-coded string you may want to change.

1.1.2 Communication Architecture

An OPC UA connection is **always** client-initiated and server-passive; two clients cannot talk to each other. So "who is the server" decides the shape of the whole system.

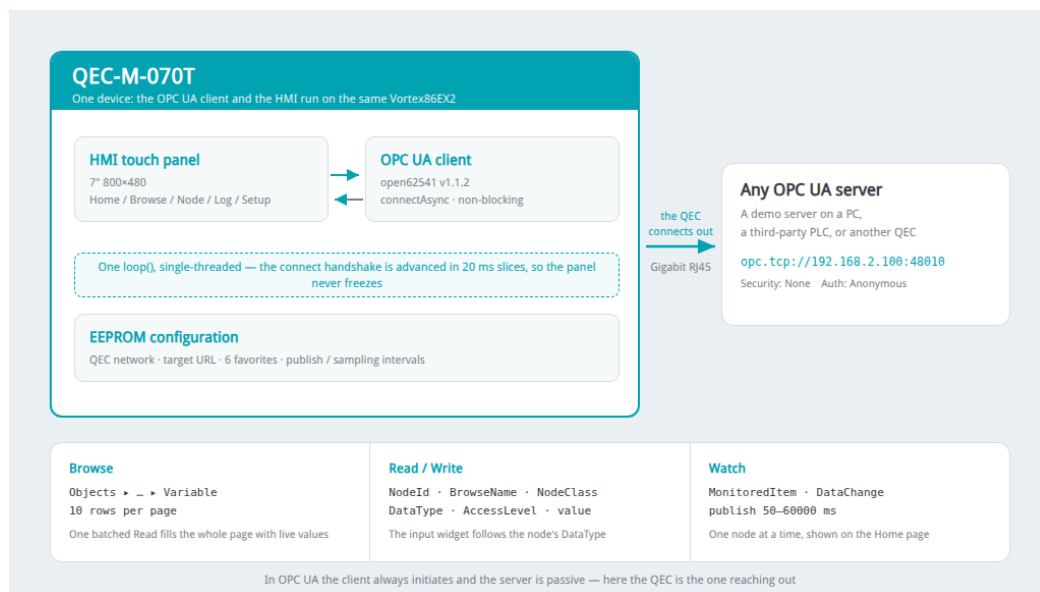


Figure 1-1 System architecture / data-flow diagram (1024×576).

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

```
UA_Client_new() / UA_Client_connectAsync() / UA_Client_Service_read() -> QEC is the client (this manual)
UA_Server_new() / UA_Server_run_startup() / UA_Server_writeValue() -> QEC is the server
(separate manual)
```

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

Non-blocking is a deliberate design decision, and it matters more on the client side than on the server side.

`UA_Client_connect()` is a **blocking** call: when the other end is unreachable it blocks for the full `cc->timeout` period, and during that time neither `handleHmiEvents()` nor `helperYield()` runs — the panel is frozen. This tool takes the async path instead:

```
/* ... doConnect(), §8 ... */
UA_StatusCode rc = UA_Client_connectAsync(client, cfgUrl);
...
/* ... loop(), §16 ... */
/* advances the handshake, then carries Publish and DataChange traffic */
UA_StatusCode rc = UA_Client_run_iterate(client, ITER_MS);
```

`UA_Client_connectAsync()` only initialises; the actual TCP connect happens inside `UA_ClientConnectionTCP_poll()`, whose `select()` timeout is exactly the value `loop()` passes to `UA_Client_run_iterate()`. **So whether or not the other end is there, one pass blocks for at most 20 ms.**

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**

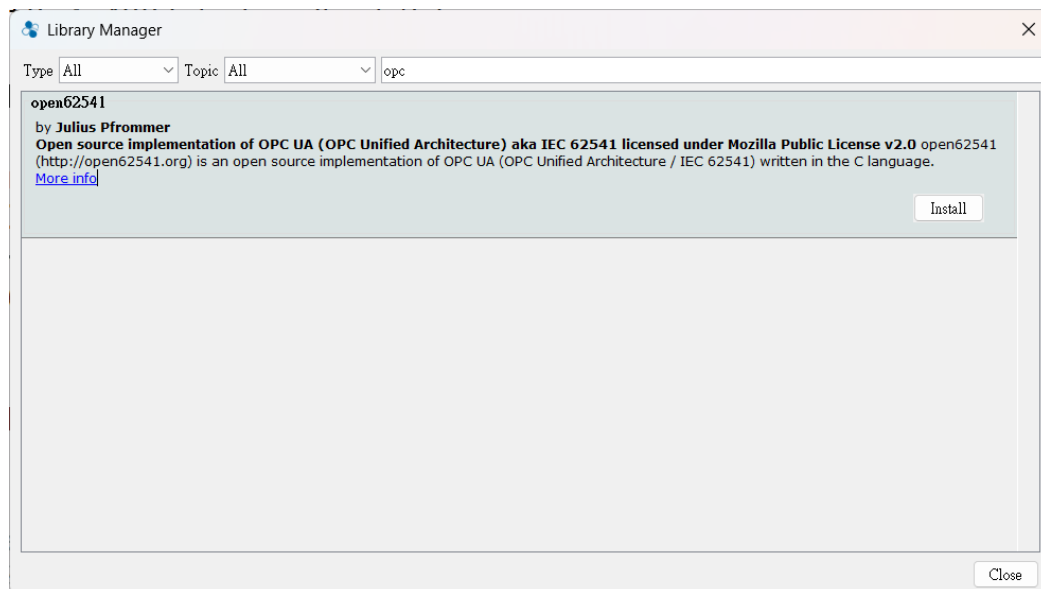


Figure 2-1 After searching for **opc** in the Library Manager, **open62541** is the library to install; click **Install**, and afterwards you will see version **1.1.2** under **Sketch** → **Include Library**. The **Server** and **Client** tools use the same library.

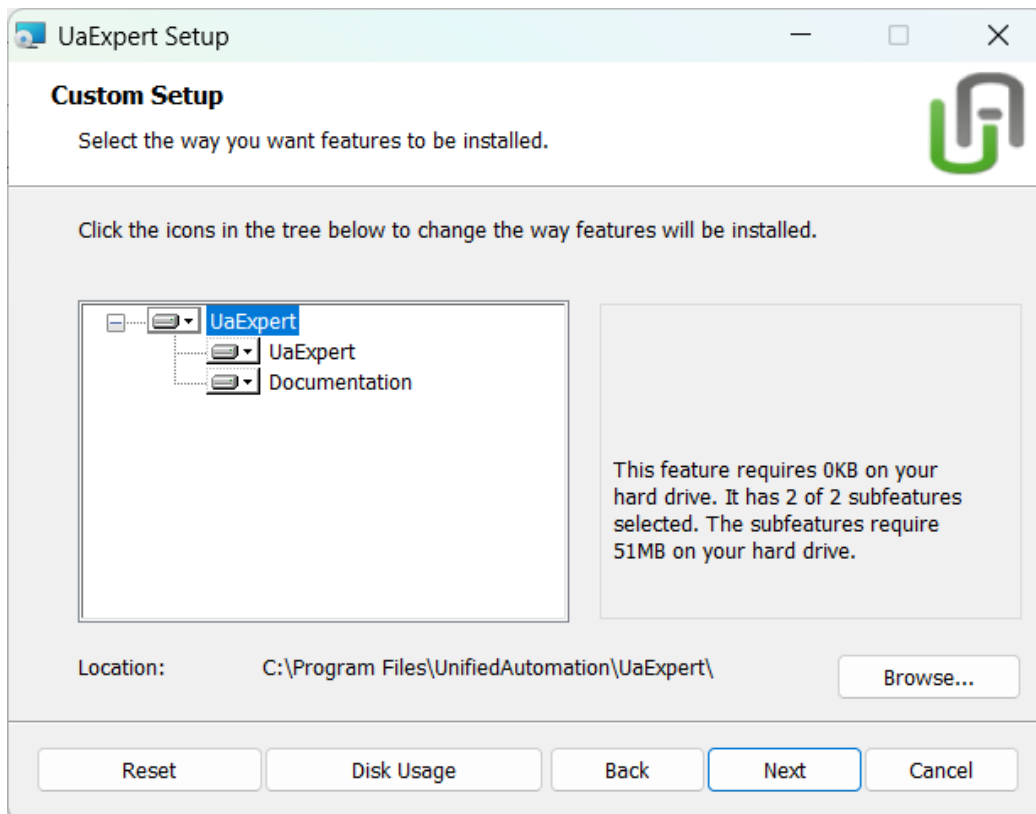
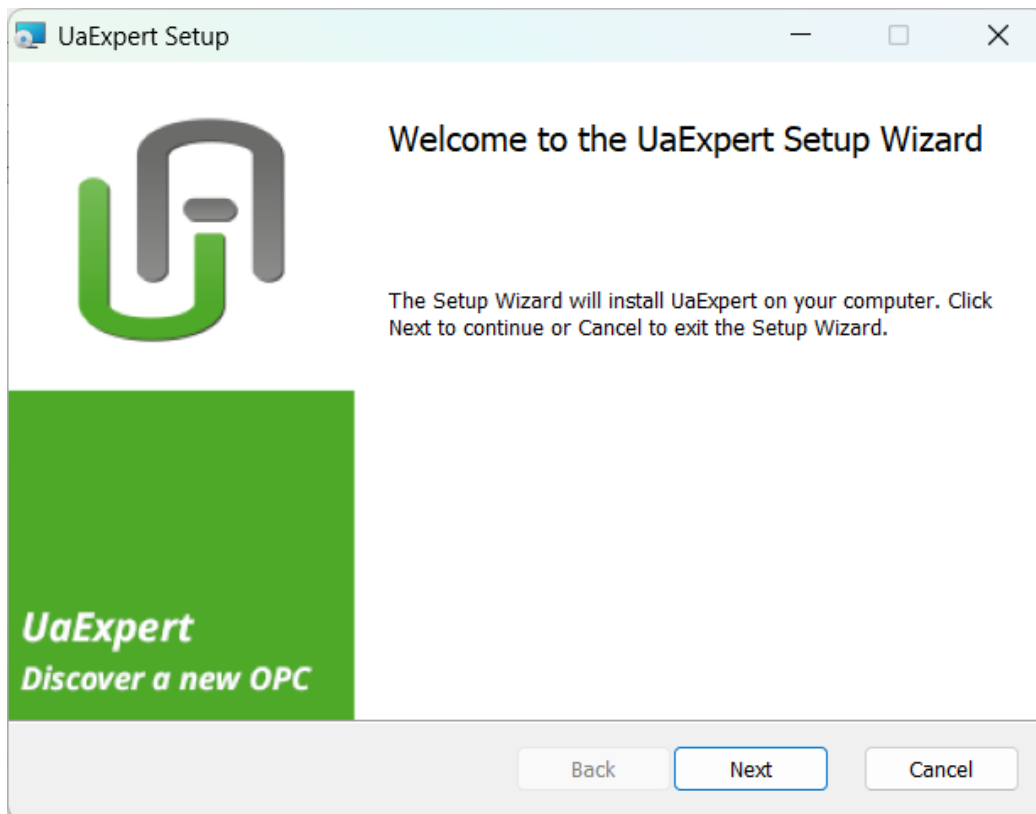
Offline install: copy the **open62541** folder straight into the IDE's **libraries** folder and restart the IDE.

Upgrading: if an earlier version is still on the machine (folder named with a capital **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 UaCPPServer (optional test server)

This client needs an OPC UA server to talk to. If you do not have one yet, the quickest route is Unified Automation's demo server **UaCPPServer** — it is installed as part of the **UaExpert** package.

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 → **Install** → **Finish**



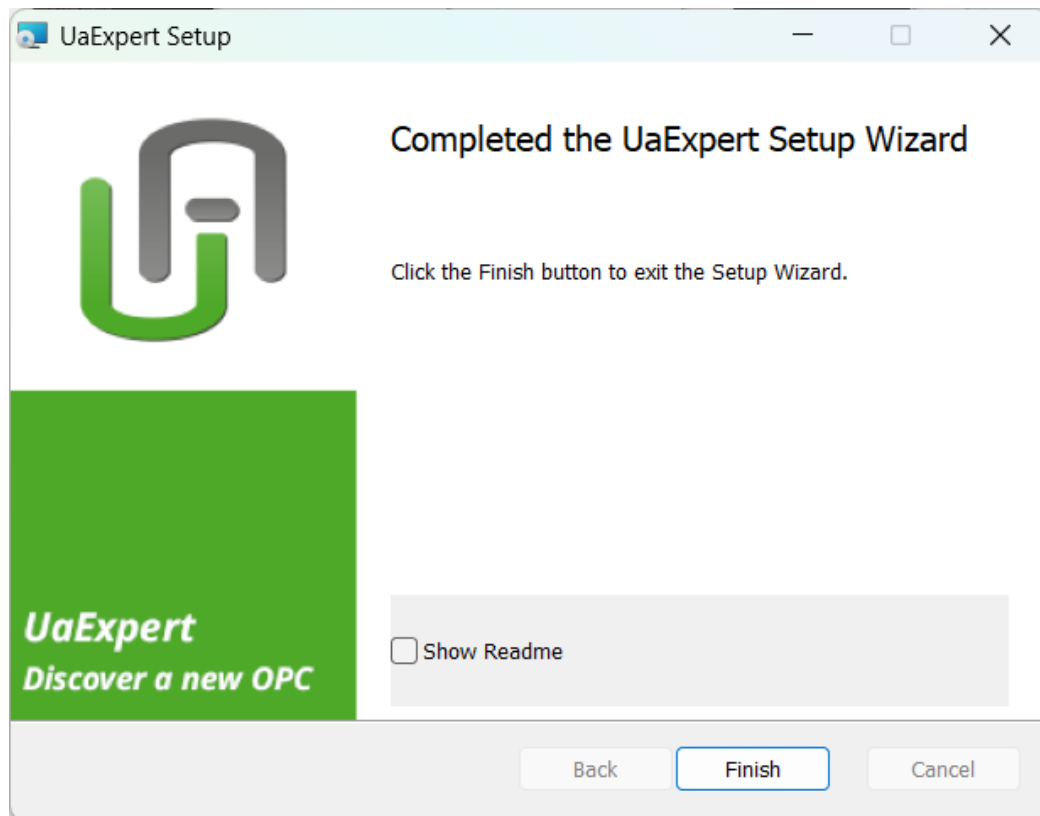


Figure 2-2 / 2-3 / 2-4 — The three installer pages that are relevant here. Keep the *Custom Setup* alone; the defaults put the whole package into *C:\Program Files\UnifiedAutomation*, UaCPPServer included.

Start UaCPPServer and its console prints:

```
Server opened endpoints for following URLs:
opc.tcp://<computer name>:48010
```

Note the port 48010, but do not copy that host name. The host part of this client's URL must be an **IP address** (§2.2.1); a computer name will never connect. Run **ipconfig** on the PC to get the IP.

If an OPC UA server is already available to you — a PLC or gateway on site, or a second QEC running the Server test tool — use that instead and skip this step entirely; see §5.2.

2.2 Code Description

QEC_OPCTUA_CLIENT_HMI.ino is roughly 1,380 lines in 16 numbered sections, and falls into five layers:

Layer	Key files / functions	Role
Network Settings	<code>cfgLoad()</code> / <code>cfgSave()</code> / <code>cfgDefaults()</code> / <code>parseIp()</code> / <code>maskValid()</code> / <code>urlProblem()</code> / <code>Ethernet.begin()</code>	EEPROM read/write and validation, static IP setup, endpoint URL validation
HMI	<code>myhmi.h</code> / <code>myhmi.cpp</code> , <code>Hmi.*</code> , <code>qlog()</code> / <code>setStatus()</code> / <code>setCfgStatus()</code>	Five-page layout, touch numeric and full keyboards, on-panel log, status bars
Application Logic	<code>handleHmiEvents()</code> / <code>brBrowse()</code> / <code>nodeLoad()</code> / <code>doWrite()</code> / <code>doWatchStart()</code> / <code>favToggle()</code>	Button handling, address-space browsing, attribute reads, type-aware writes, subscription, favourites
Task / Runtime	<code>setup()</code> / <code>loop()</code> / <code>helperYield()</code> / <code>BEGIN_HMI_EVENT_PROC</code>	Single-threaded non-blocking scheduling: 20 ms client iterate, 200 ms UI sync, 1 s LIVE refresh

Layer	Key files / functions	Role
OPC UA Client	<code>UA_Client_new()</code> / <code>UA_Client_connectAsync()</code> / <code>UA_Client_Service_browse()</code> / <code>UA_Client_Service_read()</code> / <code>UA_Client_MonitoredItems_createDataChange()</code>	Connection state machine, browse, batched read, write, subscription

2.2.1 Network Settings

Configuration lives in the first 676 bytes of **EEPROM bank 0 (EEPROM_16K)**:

addr	Contents
0	magic 'Q'
1	magic 'C'
2	version (currently 5)
3–6	QEC IP
7–10	Subnet Mask
11–14	Gateway
15–78	Target server URL (64 bytes)
79–142	Last selected node (64 bytes)
143–526	Favourite NodeIds, 6 × 64
527–670	Favourite BrowseNames, 6 × 24
671–672	publish interval ms (high, low)
673–674	sampling interval ms (high, low)
675	checksum = XOR(0..674)

At boot `cfgLoad()` validates the magic, the version and the checksum, and **if any one of them fails it falls back automatically to the sketch's DEF_* defaults**, noting **no valid EEPROM config - using defaults** in the log. So a first flash or a wiped EEPROM never leaves the device unable to start.

```
#define DEF_IP      "192.168.1.200"
#define DEF_MASK    "255.255.255.0"
#define DEF_GW      "192.168.1.1"
#define DEF_URL      "opc.tcp://192.168.1.100:48010"
#define DEF_PUBMS    500
#define DEF_SAMPMS   250
```

The factory defaults are used only when the EEPROM holds no valid data. Set them to the subnet and server you use most often on site. **The host part of the target URL must be an IP address, not a computer name.**

2.2.2 HMI

Every log line goes to both the **Serial Monitor (115200)** and the **panel's Log page**, timestamped in local time. The Home, Browse and Node pages each carry a status bar, and `setStatus()` **writes all three at once**, so whichever page you are on you see the result of the last action.

When the Log page fills up at 99 lines it is cleared, leaving **----- log full - cleared and restarted -----** on line 1, so you never come back to a handful of lines without knowing they were cleared.

2.2.3 Application Logic

Cadence	What happens
Every pass	<code>UA_Client_run_iterate(client, 20)</code> — advances the handshake and carries Publish and DataChange traffic
200 ms	<code>syncUi()</code> : connection state, LED, uptime, subscribed value
1 s	LIVE refresh, only while the Browse page is showing and LIVE is on
On retry	Increasing backoff 1 / 2 / 5 / 10 s, and only after CONNECT has been pressed
Event-driven	<code>handleHmiEvents()</code> : page changes, browse-row taps, READ / WRITE / WATCH / FAV, CONFIRM / DEFAULT / REBOOT

2.2.4 Task and Runtime

```
void loop() {
  if (client) {
    /* advances the handshake, then carries Publish and DataChange traffic */
    UA_StatusCode rc = UA_Client_run_iterate(client, ITER_MS);
    /* ...LINK_CONNECTING -> LINK_UP, or fail and schedule a retry... */
  }

  /* retry only after CONNECT was pressed, with increasing backoff */
  if (linkState == LINK_IDLE && autoRetry && (long)(millis() - nextTry) >= 0)
    doConnect();

  /* LIVE refresh, only while the Browse page is showing */
  if (liveMode && linkState == LINK_UP && !favMode &&
      Hmi.getCurrentPage() == p2 && millis() - lastLive >= LIVE_MS) {
    lastLive = millis();
    brRefreshPage();
  }
  if (millis() - lastSync >= 200) { lastSync = millis(); syncUi(); }

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

`BEGIN_HMI_EVENT_PROC` / `END_HMI_EVENT_PROC` expand to an `Hmi.eventPop()` loop that runs at most 128 times and ends by calling `helperYield()` — **and `helperYield()` is what redraws the screen**. Keep that in mind whenever you write any kind of wait: the panel does not update during `delay()`, and keeping the screen alive means calling `helperYield()`.

Known limitation. `UA_Client_Service_browse()`, `UA_Client_Service_read()` and `UA_Client_writeValueAttribute()` are still **synchronous** in open62541 1.1 — there is no non-blocking version to call. The connection itself never stalls the panel, but a Read or Browse issued at the exact moment the link drops can block for up to `SVC_TIMEOUT`. That value is deliberately set to 2000 ms:

```
/* ---- Timeouts -----
 * SVC_TIMEOUT    Read / Write / Browse are synchronous in open62541 1.1;
 *                this is how long they can block. Also the async connect timeout.
 * ITER_MS        Maximum time run_iterate() blocks per loop pass.
 * CONNECT_GUARD  Fuse against the state machine sticking in LINK_CONNECTING. */
#define SVC_TIMEOUT    2000
#define ITER_MS        20
#define CONNECT_GUARD  8000UL
```

On a local network OPC UA responses are milliseconds, so the 2 s limit is generous and you will practically never hit.

2.2.5 OPC UA Client

The connection is a three-state machine driven from `loop()`: pressing CONNECT moves it from `LINK_IDLE` to `LINK_CONNECTING`, and the session being ACTIVATED moves it on to `LINK_UP`. A failed attempt returns it to `LINK_IDLE` with a backoff, and so does a link that drops.

`doConnect()` only initiates the connection; it does not block:

```
UA_ClientConfig_setDefault(cc);
cc->timeout = SVC_TIMEOUT;
UA_StatusCode rc = UA_Client_connectAsync(client, cfgUrl);
linkState = LINK_CONNECTING;
```

This tool does not connect by itself at boot. A test tool faces a different server every time, so redialling the last address is of little value — and stalling in the first few seconds after power-on would leave the panel unresponsive. Retries start only after CONNECT has been pressed, with a 1 → 2 → 5 → 10 s backoff, so the log is never flooded.

2.3 Uploading the Code

1. Check that `DEF_IP` and the other factory defaults match the subnet on site
2. Press **Upload** in the IDE (the → arrow)
3. Compile + upload takes about **30–90 seconds**
4. Open the Serial Monitor (**115200**); within a few seconds you should see:

```
=== QEC-M-070T OPC UA Client + HMI ===
QEC_OPCUA_CLIENT_HMI
QEC_OPCUA_CLIENT_HMI
config loaded from EEPROM
Ethernet.begin ret=1 QEC IP = 192.168.2.201
ready - check the target URL, then press CONNECT
```

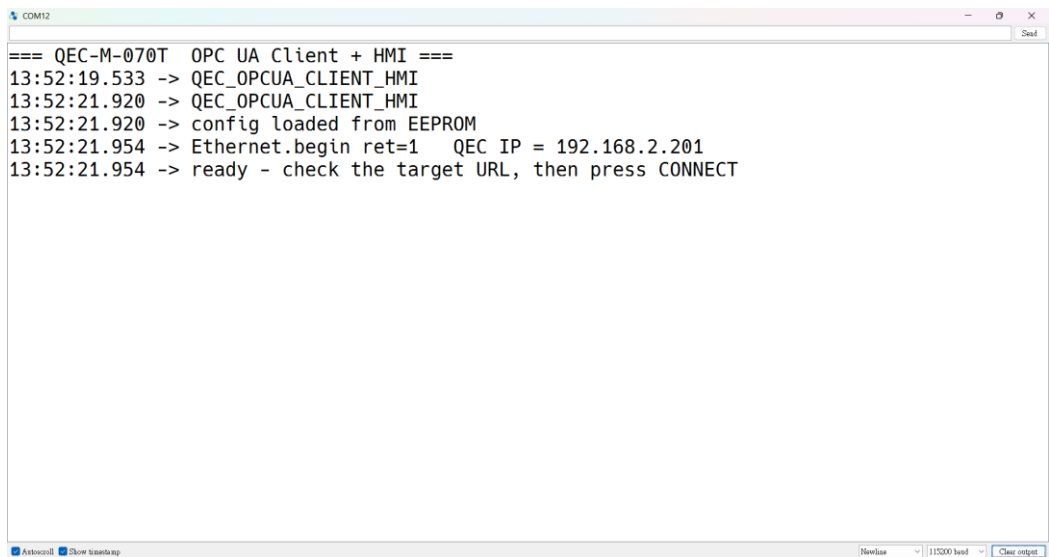


Figure 2-5 The boot messages in the Serial Monitor. Two lines matter: `Ethernet.begin ret=1` (0 means the network did not come up), and `QEC IP =` showing the configured address rather than `0.0.0.0`. Printing the sketch name twice is normal: once from `Serial.println()` in `setup()`, once from `setStatus()` mirroring the panel status bar to serial. The tool does not connect automatically — the last line prompts you to press CONNECT.

Check	Must be
<code>Ethernet.begin ret=</code>	1 (0 means the network did not come up — see §7)
<code>QEC IP =</code>	The IP you configured, not <code>0.0.0.0</code>
Last line	ready - check the target URL, then press CONNECT

Stopping here is deliberate. Nothing is connected yet; confirm the URL on the Home page first, then press CONNECT.

3. Connecting to a Server and Working with Nodes

3.1 The Endpoint

Item	Details
Where the endpoint is entered	The Target Server field on the Home page; stored in EEPROM
Format	<code>opc.tcp://<IP>:<port></code> — the host part must be an IP
SecurityPolicy	None (<code>UA_ENABLE_ENCRYPTION</code> is not compiled in)
MessageSecurityMode	None
UserTokenType	Anonymous
Discovery (FindServers)	Not compiled in — the endpoint URL has to be typed in
Connect	Asynchronous; the state machine is driven from <code>loop()</code>
Retry	Only after CONNECT has been pressed; backoff 1 / 2 / 5 / 10 s

`urlProblem()` validates the field before it is saved:

Check	Failure message
Starts with <code>opc.tcp://</code>	URL must start with <code>opc.tcp://</code>
At most 64 characters	URL too long (64 char max)
Has a host part	URL has no host part
Host contains only digits and dots	URL host must be an IP address, not a hostname

3.2 Browsing the Address Space

Browsing starts at **Objects** (`i=85`) and issues one **Browse** service call per level:

```
req.nodesToBrowse[0].browseDirection = UA_BROWSEDIRECTION_FORWARD;  
req.nodesToBrowse[0].referenceTypeId = UA_NODEID_NUMERIC(0, UA_NS0ID_HIERARCHICALREFERENCES);  
req.nodesToBrowse[0].includeSubtypes = true;
```

Limit	Value	Meaning
<code>BR_MAX</code>	60	Child nodes listed for one node. Above that the status bar shows <code>browse: showing <n> of N child nodes (BR_MAX limit)</code>
<code>BR_ROWS</code>	10	Rows per page; must match the layout
<code>BR_PATH</code>	8	Maximum drill-down depth

Values are shown directly in the list. After every browse, and on every page turn, the tool collects the `ReadValueId` of every Variable **on the current page** and sends them in **one** Read service call:

```
rq.nodesToRead = rvi;  
rq.nodesToReadSize = (size_t)n; /* ... n <= BR_ROWS ... */  
rq.timestampsToReturn = UA_TIMESTAMPSTORETURN_NEITHER;
```

That is **one round trip per page**, not one per node — and that is exactly what makes live values in the list possible.

3.3 Node Attributes

Opening a node reads four attributes plus the value:

Attribute	API
NodeClass	<code>UA_Client_readNodeClassAttribute()</code>
BrowseName	<code>UA_Client_readBrowseNameAttribute()</code>
DataType	<code>UA_Client_readDataTypeAttribute()</code> , then <code>UA_findDataType()</code> to map it to a name
AccessLevel	<code>UA_Client_readAccessLevelAttribute()</code> → R / W / RW
Value + two timestamps	<code>UA_Client_Service_read()</code> , <code>UA_TIMESTAMPSTORETURN_BOTH</code>

Reading the DataType attribute (rather than guessing the type from the value that comes back) is the key: it lets the tool write to a node it has never read, and it is what drives the input-widget switching in §4.3.

3.4 Supported Write Types

DataType	Input widget	Keyboard
Double / Float / Int16 / Int32 / UInt16 / UInt32 / Byte / SByte	NumberInput	Numeric
Boolean	NumberInput	Numeric (0 / 1)
String	TextInput	Full keyboard
Anything else	NumberInput	Numeric; WRITE returns <code>write: <type> not supported by this panel</code>

3.5 Subscription

Item	Details
Subscriptions	One at a time ; a new WATCH replaces the previous one
Publishing interval	<code>cfgPubMs</code> , 50–60000 ms, set on the Setup page
Sampling interval	<code>cfgSampMs</code> , 10–60000 ms, set on the Setup page
Timestamps requested	<code>UA_TIMESTAMPSTORETURN_BOTH</code>
Nodes that can be watched	Variables only — anything else returns <code>watch: only Variable nodes can be watched</code>

The revised values the server actually grants are shown in the status bar: `watching Double` `publish=500ms`
`sampling=250ms`.

4. HMI Page Reference

Five pages, Material / Light theme, designed for 800×480. Each page does exactly one job.

4.1 Home — the Connection

This page does one thing: connect or disconnect. The top half is the target endpoint and the session state, the bottom half is the node currently subscribed.

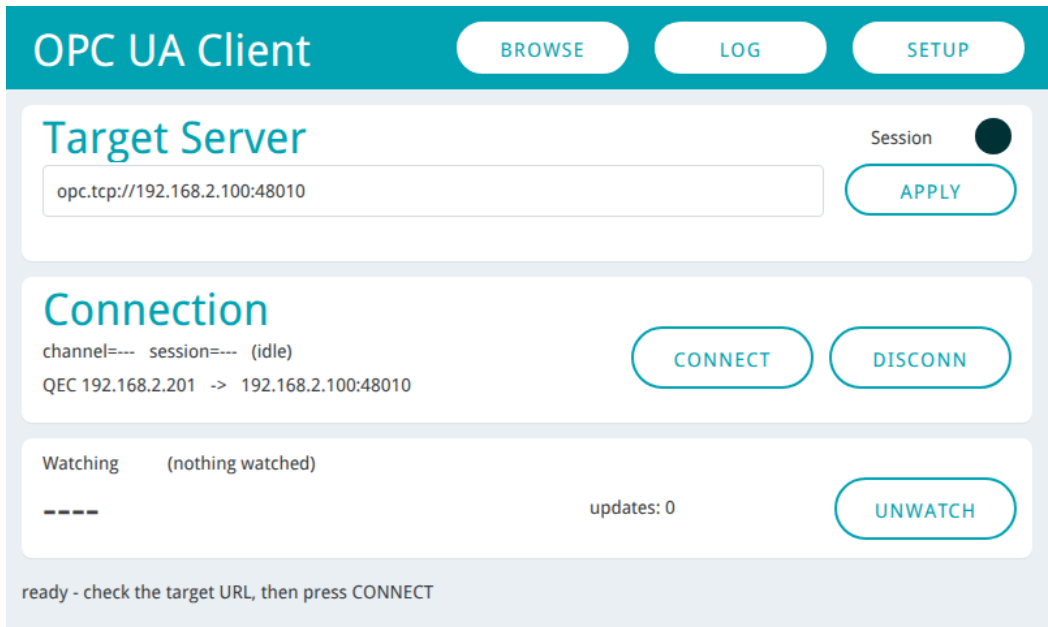


Figure 4-1 The Home page after boot: nothing connected yet, the Session LED dim, the status bar asking you to check the URL and press CONNECT.

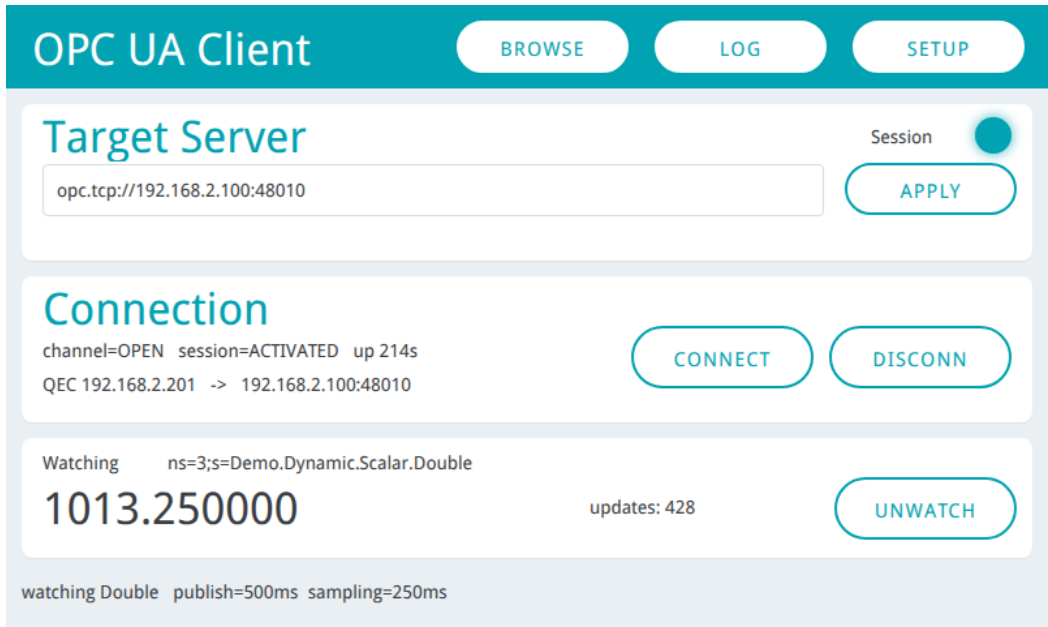


Figure 4-2 The Home page with a session running: LED lit, channel OPEN / session ACTIVATED, and one node subscribed.

Field / button	Function
Target Server field	The endpoint. Tap it for the full keyboard
APPLY	Validates and saves the URL. If a connection is already running, it switches to the new address immediately
Session + LED	Lit when the session is ACTIVATED, dim otherwise

Field / button	Function
channel= session= up	The raw SecureChannel and Session states, plus seconds since the session was established
QEC <ip> -> <target>	This device's address and the endpoint it is connected to
CONNECT	Starts connecting and enables automatic retry
DISCONN	Disconnects and stops retrying
Watching	The subscribed NodeId, its latest value, and the DataChange count
UNWATCH	Deletes the subscription
Status bar (bottom)	The most recent event, identical to the last line on the Log page
BROWSE / LOG / SETUP	Go to the Browse, Log and Setup pages

Example event messages

```
connecting opc.tcp://192.168.2.100:48010
connected
connect FAILED BadDisconnect  retry in 5 s
link lost  retry in 1 s
watching Double  publish=500ms  sampling=250ms
disconnected
```

Figure 4-3 A failed attempt: the reason and the retry countdown share one status bar, and the panel stays fully usable throughout.

4.2 Browse — Finding Nodes

Ten rows per page, with drill-down into folders and a return to the level above. The left column tells you what each row will do when tapped; the right column shows the current value of every Variable.

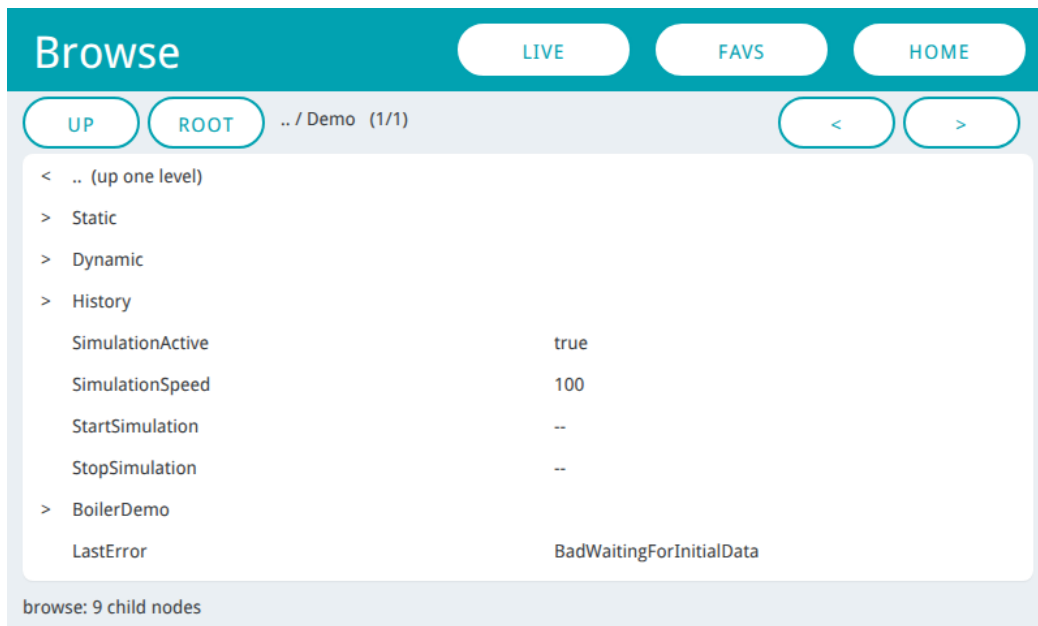


Figure 4-4 The Browse page one level inside Demo. The left column marks what each row does; folders show no value.

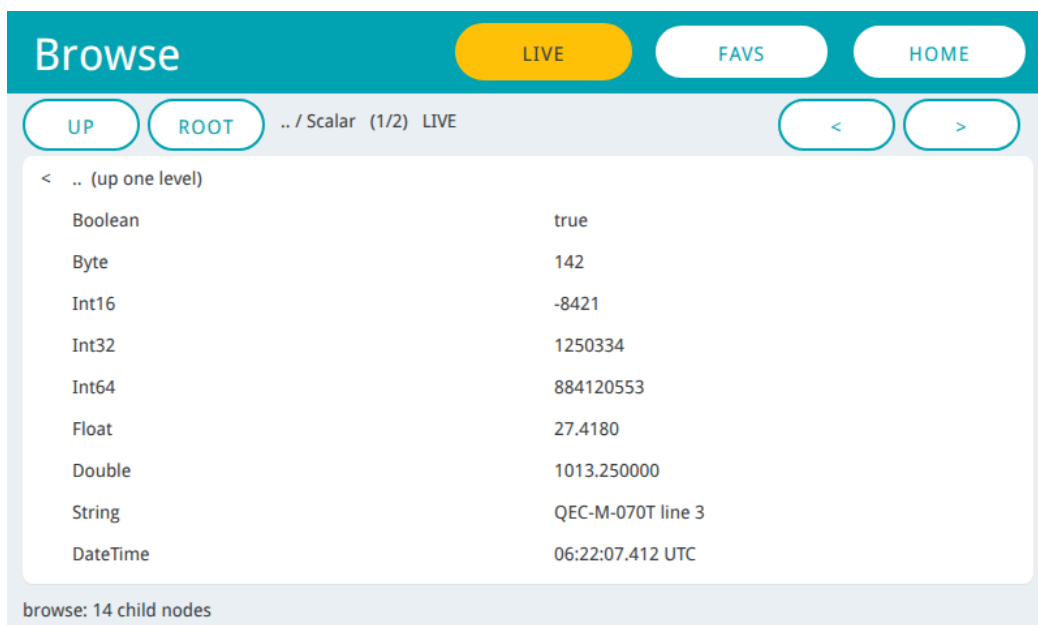


Figure 4-5 LIVE on: the whole page is re-read every second, which is ten nodes at once. The LIVE button turns amber and the breadcrumb carries a LIVE marker.

Field / button	Function
Left marker column	> this row drills down (an Object), < is .. up one level, blank is a leaf
Name column	BrowseName
Value column	The current value of a Variable; the StatusCode when it cannot be read
Breadcrumb	The last two path levels, the page number, and LIVE when LIVE is on
UP / ROOT	Up one level / back to Objects
< / >	Previous / next page. Turning a page re-reads the values
LIVE	Toggles the one-second refresh of the current page. Amber while on
FAVS	Shows the saved favourites instead of the browse result
HOME	Back to the Home page

Field / button	Function
Tapping a row	... goes up · an Object drills down · everything else opens the Node page

LIVE is off by default. Leaving the Browse page pauses the refresh, and coming back resumes it.

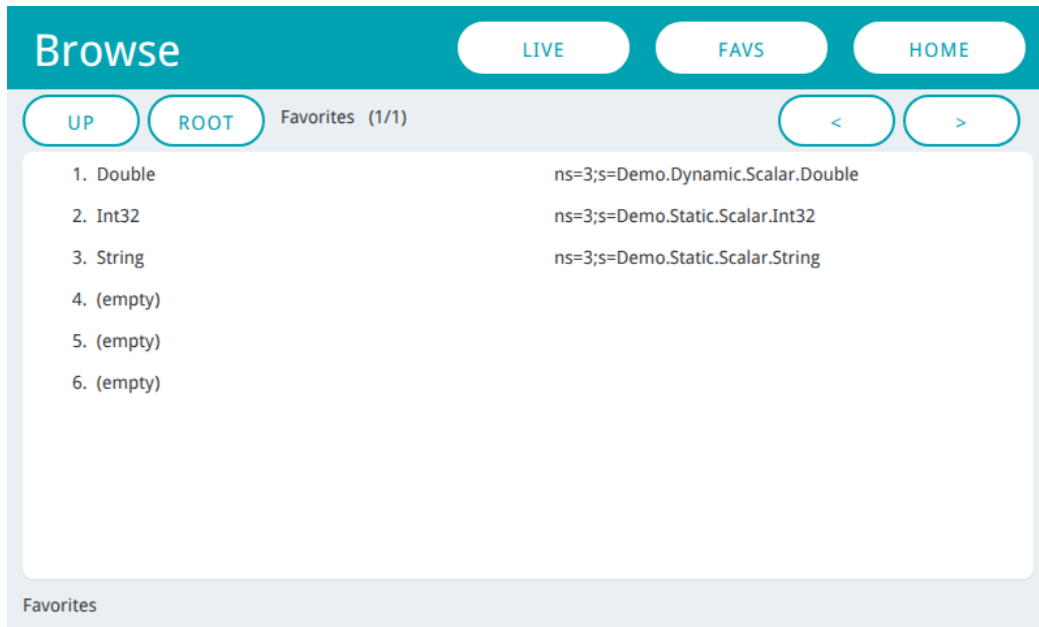


Figure 4-6 FAVS shows six slots: BrowseName on the left, the full NodeId on the right. Tap a row to open it on the Node page.

4.3 Node — Using a Node

One node's attributes, its current value and the write field all live on this page. The write field follows the node's declared DataType and becomes a numeric box or a text box accordingly.

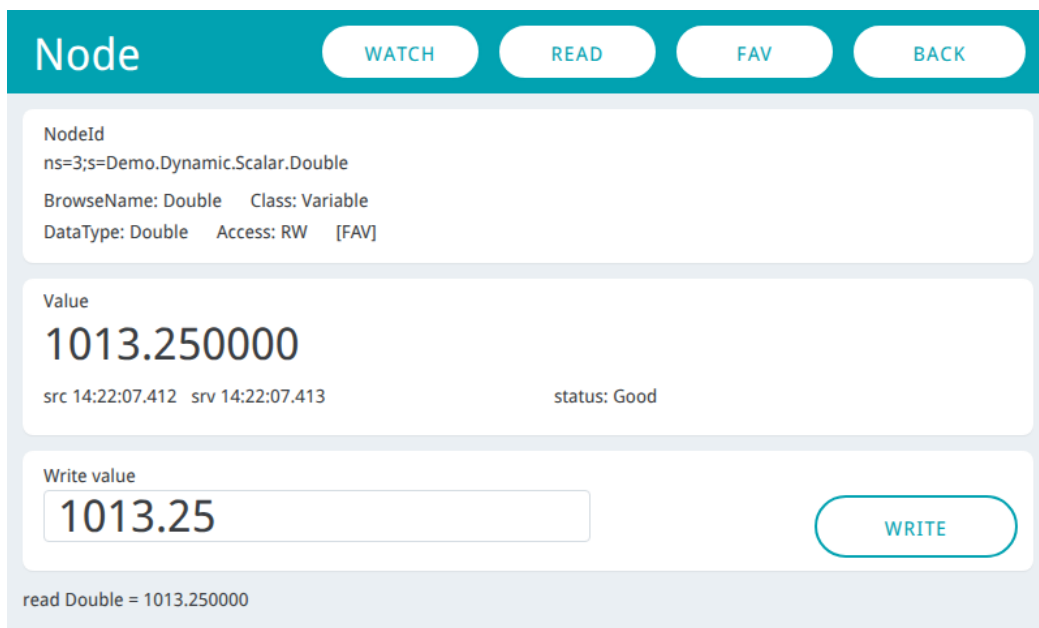


Figure 4-7 A Double node. The write field is a `NumberInput`; tapping it opens the numeric keyboard. The `[FAV]` at the end of the attribute row means this node is already saved as a favourite.

Node
WATCH
READ
FAV
BACK

NodeId
ns=3;s=Demo.Static.Scalar.String
BrowseName: String Class: Variable
DataType: String Access: RW

Value
QEC-M-070T line 3
src 14:23:11.008 srv 14:23:11.009 status: Good

Write value

WRITE

write String -> Good

Figure 4-8 A String node. The same position becomes a `TextInput`, and tapping it opens the full keyboard.

Field / button	Function
NodeId	The full NodeId of the selected node
BrowseName / Class	From the BrowseName and NodeClass attributes
DataType / Access	From the DataType and AccessLevel attributes. [FAV] appears when the node is a favourite
Value	The current value, formatted by type
src / srv	Source and Server timestamps
status	The StatusCode of this read
Write value	The input widget follows the DataType — numeric and boolean use the numeric keyboard, strings the full keyboard
WRITE	Writes the field to the node, then reads it back
READ	Re-reads all attributes and the value
WATCH	Subscribes to this node (Variables only)
FAV	Toggle: saves the node as a favourite, or removes it if already saved
BACK	Back to the Browse page

Node
WATCH
READ
FAV
BACK

NodeId
ns=3;s=Demo.Dynamic.Scalar.Int32
BrowseName: Int32 Class: Variable
DataType: Int32 Access: R

Value
1250334
src 14:24:02.180 srv 14:24:02.181 status: Good

Write value

write Int32 -> BadNotWritable

Figure 4-9 Writing to a node whose AccessLevel is **R**: the server returns *BadNotWritable* and the status bar reports it verbatim. Access was already shown before you tried.

Favourites have six slots. **FAV** saves the node in the first free slot, or removes it if it is already there. When all six are full the tool **refuses** rather than overwriting an older one: *favorites are full (6) - open one from FAVS and press FAV to remove.*

4.4 Log — a Live Console on the Panel

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

Log
CLEAR
HOME

1 14:20:58 QEC_OPCUA_CLIENT_HMI
2 14:20:58 config loaded from EEPROM
3 14:20:58 Ethernet.begin ret=1 QEC IP = 192.168.2.201
4 14:20:58 ready - check the target URL, then press CONNECT
5 14:21:44 connecting opc.tcp://192.168.2.100:48010
6 14:21:44 connected
7 14:21:44 browse: 3 child nodes
8 14:22:05 browse: 9 child nodes
9 14:22:31 read Double = 1013.250000
10 14:22:48 watching Double publish=500ms sampling=250ms

Figure 4-10 The Log page: boot, connect, browse, read, subscribe, and one rejected write — every line timestamped.

Field / button	Function
TextList (p4t11)	99 lines, numbered, auto-scrolled to the newest line
When it fills up	Cleared, with <i>----- log full - cleared and restarted -----</i> written on line 1
CLEAR	Clears it and restarts from line 1
HOME	Back to the Home page

The contents match 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 and there is no laptop, this page is the only debugging view available.

4.5 Setup — the QEC's Own Network and Subscription Intervals

This page holds the QEC's own address and the subscription intervals — the target server's URL is not here, it is on the Home page.

The Setup page has a teal header with the title 'Setup' and four buttons: REBOOT, DEFAULT, CONFIRM, and BACK. Below the header, there are two main sections. The first section is titled 'QEC Network (restart required)' and contains input fields for IP (192.168.2.201), Mask (255.255.255.0), and Gateway (192.168.2.1). Below these fields, it says 'Active now' followed by 'IP 192.168.2.201 Mask 255.255.255.0 GW 192.168.2.1'. The second section is titled 'Subscription' and contains input fields for Publish (ms) (500) and Sampling (ms) (250). At the bottom, it shows 'QEC_OPCUA_CLIENT_HMI target opc.tcp://192.168.2.100:48010' and a status message: 'Stored in EEPROM - network needs REBOOT; subscription applies on next WATCH'.

Figure 4-11 The Setup page. Only the QEC's own network lives here; the target URL is on the Home page — where it is actually used.

Field / button	Function
IP / Mask / Gateway	The QEC's own address. TextInputs; tap for the touch keyboard
Active now	The values actually in effect since boot. Different from the fields above → settings were changed but the device has not been restarted
Publish (ms)	The subscription's publishing interval, 50–60000
Sampling (ms)	The MonitoredItem's sampling interval, 10–60000
The version / target line	Which sketch is flashed and which endpoint is stored — useful with no serial cable attached
CONFIRM	Validates, then writes to EEPROM
DEFAULT	Returns to the sketch's <code>DEF_*</code> factory defaults and writes them. Press twice within 3 s
REBOOT	Soft-restarts the device so the network settings take effect. Press twice within 3 s
BACK	Back to the Home 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

Check	Failure message
Gateway is four octets 0–255	Gateway format error
The first octet of the IP must not be 0	IP address must not start with 0
The mask must be contiguous	Subnet mask must be contiguous, e.g. 255.255.255.0
The IP must not be the network address	IP is the network address of this subnet
The IP must not be the broadcast address	IP is the broadcast address of this subnet
Publish interval 50–60000	Publish interval must be 50 - 60000 ms
Sampling interval 10–60000	Sampling interval must be 10 - 60000 ms

Figure 4-12 CONFIRM after entering 255.0.255.0 as the subnet: the status bar explains why it was rejected, and the value is not saved.

Mask contiguity is checked bit by bit (255.0.255.0 is rejected — saved and restarted, it would take the whole device off the network); the network and broadcast addresses are derived **from the mask**, so 192.168.5.0 is still valid under a /16.

A gateway outside the subnet warns but does not block (internal test networks often need no gateway at all); the saved message becomes **Saved: ... - WARNING gateway not in this subnet.**

Double Confirmation

DEFAULT and REBOOT either overwrite on-site settings or drop the connection, so both need **two presses within 3 s**:

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

The two buttons are mutually exclusive: pressing REBOOT cancels a DEFAULT that has already been pressed once, and the other way round; CONFIRM cancels both. Entering the Setup page resets everything as well.

5. Running It Once: Your First Server Connection

5.1 What This Chapter Does

Find an OPC UA server to connect to and follow the shortest route: **connect** → **browse nodes** → **read once and write once** → **subscribe to a node that changes**. About ten minutes, with no code to write and no PC-side OPC UA client needed.

Once through, you have confirmed three things: the QEC can reach the other end, the panel shows you which nodes it has, and reads, writes and subscriptions all work. **Whether you go on to attach a demo server, a PLC on site or another QEC makes no difference to this tool** — it is the same standard interface every time.

5.2 Before You Start

1. The QEC MDevice is flashed with `QEC_OPCTUA_CLIENT_HMI` and serial shows **ready** - check the target URL, then press **CONNECT**
2. The network cable is in the **Gigabit RJ45** port (not one of the two 10/100 EtherCAT ports)
3. An OPC UA server is running and reachable — **any OPC UA server can be used**, see below. This chapter uses Unified Automation's demo server (installation steps in §2.1.3); note down its IP and port (48010 by default). The endpoint its console prints uses the PC's computer name — get the IP from `ipconfig` instead of copying that line
4. The PC and the QEC are on the **same subnet** (check with `ipconfig` on the PC)
5. The server accepts **Security None / Anonymous** — this client has no encryption compiled in

Choosing a Server to Test Against

Nothing in this tool is specific to the demo server. If there is already a server on the network, just point the QEC at it — no installation needed. Connecting, browsing and reading are identical in every case; only the node names in the screenshots differ:

Server to connect to	Endpoint to enter	Notes
A second QEC running <code>QEC_OPCTUA_Server_HMI</code>	<code>opc.tcp://<that QEC's IP>:4840</code>	The cleanest pairing: one panel serves, the other browses it, and each has its own log to compare. Its ten <code>QEC.*</code> nodes are documented in full in the Server manual, and <code>QEC.Counter</code> increments every second — exactly what LIVE and WATCH need
An existing PLC, gateway or SCADA server on site	Whatever address it publishes	Confirm first that it allows Security None / Anonymous, and that its nodes are safe to write. On running equipment, READ before you WRITE
Unified Automation's demo server	<code>opc.tcp://<the PC's IP>:48010</code>	What the rest of this chapter uses. Its <code>ns=3;s=Demo.*</code> tree covers every type the panel can write

Pairing two QEC units. One runs the Server sketch at `192.168.2.200:4840` and this one sits at `192.168.2.201`; enter `opc.tcp://192.168.2.200:4840` and press **CONNECT**. Browse to **Objects** ▶ **QEC** and you will see the server's ten nodes with live values; **Setpoint** and **Analog** are read-only by design, which makes them ready-made subjects for the **BadNotWritable** case in §5.5.

The one hard requirement is the host part of the endpoint. Whichever server you pick, the host part must be an **IP address** (§2.2.1).

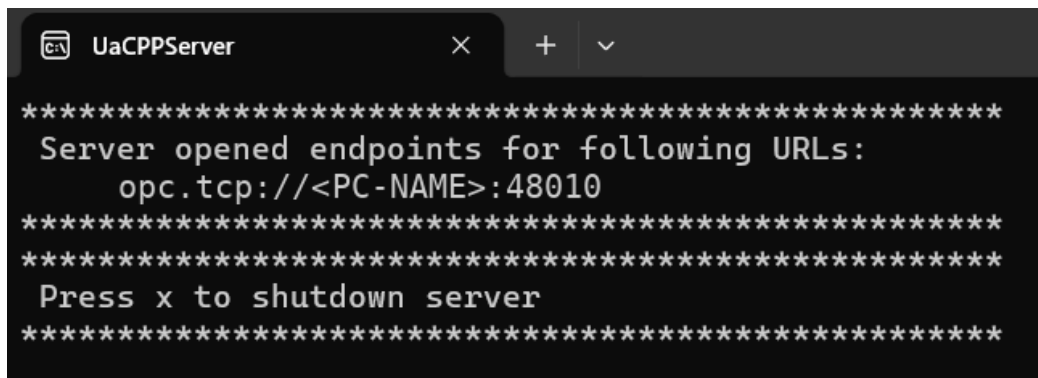


Figure 5-1 UaCPPServer running on a PC. *Server opened endpoints for following URLs* means it is listening, and the port at the end of that URL — **48010** — is the number to enter in the QEC's target URL. The host part it prints is the PC's computer name, and that is exactly what you cannot enter on the QEC* (see §2.2.1): run **ipconfig** on the PC and use the IP instead, i.e. **opc.tcp://192.168.2.100:48010**. The computer name is masked in this figure.*

5.3 Steps

#	Step	Action
1	Check the QEC's own address	SETUP → confirm Active now matches your subnet → BACK
2	Enter the endpoint	On Home , tap the Target Server field, enter opc.tcp://<server IP>:48010 , confirm
3	Save it	Press APPLY — the status bar shows target saved: ...
4	Connect	Press CONNECT
5	Confirm the session	The Session LED lights, the line reads channel=OPEN session=ACTIVATED , and the status bar shows connected
6	Browse	Press BROWSE . The list is already at Objects — the tool browses once automatically on connect
7	Drill down	Tap a row marked > to go deeper; tap .. or UP to come back
8	Watch ten at once	Find a folder full of Variables and press LIVE — the button turns amber and the whole page is re-read every second
9	Open a node	Tap an unmarked row; the Node page opens with the attributes and the value
10	Save it	Press FAV ; [FAV] appears on the attribute row

5.4 Once Connected

The Home page shows **channel=OPEN session=ACTIVATED** with the LED lit, and the Log page records **connecting ...** followed by **connected**. Press **BROWSE** and the children of **Objects** are already listed — **with the Variables' values already filled in, without opening a single node**.

That whole page of values came back in **one batched Read** (one round trip per page, not one per node), and that is what makes the browse list on the panel genuinely usable.

5.5 Four Things to Try

Tap an unmarked row on the Browse page and the Node page opens with that node's attributes. Then do four things:

1. Read once — press **READ**; the value and both timestamps update and the status bar shows **read <name> = <value>**. The **DataType** and **Access** on the attribute row are what the next two steps depend on.

2. Write a number — enter **1.2345** in the write field and press **WRITE**; the status bar returns **write <name> -> Good** and the value card becomes **1.234500**.

3. Write a string — open a String node instead. The write field in the same position **becomes a text box, pre-filled with the current value**, and tapping it opens the full keyboard rather than the numeric one. Change a character and press **WRITE**. The input widget switches on the node's declared **DataType**, not on a guess.

4. Subscribe to a node that changes — press **WATCH**, then **BACK** → **HOME**. The Watching block on the Home page shows the NodeId, a value that keeps changing, and a rising **updates:** count. The **publish=<n>ms** **sampling=<n>ms** line is the interval the **server actually granted**, which is often not the one you asked for.

Things You Will Notice Along the Way

What you see	What it means
Entering a folder full of Variables, the right column already has values	One batched Read fetched the whole page (§3.2)
Values are already there after turning a page	Each page turn re-runs the batched read
After LIVE the button turns amber and LIVE appears in the breadcrumb	The whole page is re-read every second; it pauses when you leave the Browse page and resumes when you return
WRITE on a node whose Access is R returns BadNotWritable	Access protection is working — this is expected
opc.tcp://MYPC:48010 is rejected	The host part must be an IP (§2.2.1); it is caught before saving
link lost retry in 1 s after unplugging the cable	Reconnection is automatic, with a 1 / 2 / 5 / 10 s backoff
The target URL, favourites and intervals survive a power cycle	All of it is in EEPROM (§6)

At this point the QEC's ability to act as an OPC UA client is confirmed. How to browse nodes, how attributes are read and which types can be written are all in §3; what to do next is in §9.

5.6 These Messages Are Normal

First-time users most often mistake the following for faults. None of them are:

What you see	What it means
connect FAILED BadDisconnect immediately	Wrong IP or port, the server is not running, or a firewall. Confirm the endpoint from the server side first
URL host must be an IP address, not a hostname	This is normal — enter the server's IP, not its computer name (see §2.2.1)
Connected, but BROWSE is empty	The server has nothing under Objects , or this session has no browse rights. Compare with a PC-side client on the same endpoint
A row you expected to expand opened the Node page instead	Expected in this version — only nodes whose NodeClass is Object drill down. See §7
The value column is blank on some rows	Those rows are not Variables (folders, Methods); there is no value to read
The value column shows a StatusCode	The node exists but cannot be read — what you see is the server's answer

6. Configuration Persistence

Everything needed to bring the tool back to where you left it lives in EEPROM: the QEC's own network, the target endpoint, the favourites and the subscription intervals.

Item	Purpose	How
The EEPROM config area	Keeps the QEC network, target endpoint, favourites and subscription intervals across power cycles	The first 676 bytes of bank 0, with magic + version + XOR checksum; any mismatch falls back to <code>DEF_*</code>
The Active now line	Separates "saved but not yet in effect" from "actually running"	On the Setup page; shows the values in effect since boot, refilled by <code>activeToUi()</code> each time the page is opened
The REBOOT button	Lets "save → take effect" be completed on the panel	Two presses within 3 s

6.1 What Needs a Restart and What Does Not

Setting	When it takes effect
The QEC's IP / mask / gateway	On restart — <code>Ethernet.begin()</code> runs once, in <code>setup()</code>
Target server URL	On the next CONNECT — press <code>APPLY</code> , and a running connection switches immediately
Publish / Sampling intervals	On the next WATCH
Favourites	Immediately

Only the QEC's own network needs a restart, which is why that is the only block on the Setup page marked (`restart required`).

6.2 Why the Target URL Lives on the Home Page

Pointing the tool at another server is the thing you do most often, so the field sits on the page that manages the connection rather than buried in a settings screen. `APPLY` validates, saves, and — if a session is already running — switches to the new address in one step.

6.3 EEPROM Layout Version Changes

The layout carries a version byte (`EE_VER`, currently 5). When a firmware update changes the layout, the version no longer matches and `cfgLoad()` falls back to the defaults **deliberately**, rather than reading old bytes as if they were new.

After a firmware update that changed `EE_VER`, the first boot shows no valid EEPROM config - using defaults. The IP and target URL have to be entered again. This is by design — misreading an old layout would be worse.

7. Troubleshooting

Problem	Likely cause	Fix
Compile error 'p3nin1' was not declared in this scope	The layout in <code>data_project.hpj</code> was changed without saving it in the 86HMI designer — <code>myhmi.h/cpp</code> are still the old ones and are missing the new widget	Open the designer, load <code>data_project.hpj</code> , save once to regenerate <code>myhmi.h/cpp</code> , then compile. You will not hit this unless you changed the layout (both files ship with the project)
Serial prints QEC IP = 0.0.0.0, Ethernet.begin ret=0	Cable in the wrong port / an IP address conflict with another device / invalid EEPROM contents	Confirm the cable is in the Gigabit port; pick an unused IP; on the panel SETUP → DEFAULT (×2) → REBOOT (×2)
URL must start with opc.tcp://	The scheme is missing	Enter the full endpoint including <code>opc.tcp://</code>
URL host must be an IP address, not a hostname	A computer name was entered	This is deliberate — DJGPP has no <code>getaddrinfo()</code> . Use the server's IP
connect FAILED BadDisconnect retrying forever	The server is not running, wrong port, a firewall, or a different subnet	Confirm the endpoint with a PC-side client first; check Active now on the Setup page against your subnet
Connected, then link lost a few seconds later	The server closed the session (security or user-token policy)	This client offers Security None / Anonymous only. Open an anonymous None endpoint on the server
A node that expands in a PC client opens the Node page here	Only nodes whose NodeClass is Object drill down. Variables with children (a DataAccess AnalogItem, for example) open as a node instead of expanding	A known limitation of this version. Those children are still reachable from a PC client, and reading and writing the node itself works normally here
browse: showing <n> of N child nodes (BR_MAX limit)	That node has more children than BR_MAX	Raise BR_MAX in the <code>.ino</code> and reflash, or enter through a folder with fewer children
The value column is blank on a row	That row is not a Variable	Expected — only Variables are read
write: data type unknown - press READ first	The DataType attribute could not be read	Press READ ; if it still fails, that server does not expose the attribute
write: <type> not supported by this panel	The node is not a supported scalar type	See §3.4
watch: only Variable nodes can be watched	WATCH was pressed on an Object or a Method	Expected
favorites are full (6)	All six slots are in use	FAVS → open one → FAV to remove
The panel pauses for a second or two when the link drops	Expected — Read / Write / Browse are synchronous in open62541 1.1	Worst case is SVC_TIMEOUT (2000 ms). The connection itself never blocks
REBOOT does nothing	This IDE version's port-IO behaviour differs	Power-cycle the device manually; nothing else is affected
no valid EEPROM config on every boot	The layout version changed, the EEPROM write did not complete, or the checksum does not match	Press CONFIRM once on the Setup page; if it still fails, the EEPROM may be faulty
Status-bar messages are truncated	That Label's width was shrunk to its text by the designer	See the warning in §8; check that the status-bar Label is still at least 700 px wide

8. Complete Code and Downloads

```
QEC_OPCUA_CLIENT_HMI\  
├── QEC_OPCUA_CLIENT_HMI.ino    Main program (~1,380 lines in 16 sections; edit this one)  
├── CHANGELOG.md               Design decisions and measurements behind the code  
├── myhmi.h                   Generated by the 86HMI designer (do not edit)  
├── myhmi.cpp                 Generated by the 86HMI designer (do not edit)  
└── data\  
    └── _project.hpj           HMI layout project (5 pages; edit in the designer)
```

File	Notes
QEC_OPCUA_CLIENT_HMI.ino	All of the application logic, in 16 numbered sections with an index at the top of the file. The five layers are in §2.2
data_project.hpj	The HMI layout. Change the layout in the designer, not by editing myhmi.* directly
CHANGELOG.md	Includes the field notes: why the connection had to become asynchronous, the browse pointer-aliasing bug and its fix, why the Node page stacks two input widgets

⚠ **Read this before changing the HMI layout: the designer overwrites Label widths.** Setting a Label's **Width** in the **.hpj** JSON has no effect — when you save in the 86HMI designer, it re-fits every Label's width to its **design-time text** (measured at about 8.15 px per character). So the placeholder text in this project is deliberately set to the **longest realistic case**: 94 characters for the status bar, 72 characters for each of the ten browse rows. **After changing the text of those Labels, go back and check that the width has not shrunk**, or long runtime messages will be truncated.

The Node page deliberately carries **two overlapping input widgets** (**p3nin1** and **p3tin1**). The designer preview shows them stacked, and that is correct. **Do not separate them** — the sketch hides whichever one does not apply.

Remember to save after changing the layout, or **myhmi.h** / **myhmi.cpp** will not be updated.

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

9. Next Steps

With the QEC confirmed as an OPC UA client, the next step is putting it to work in a real setting.

9.1 Against an Existing Server on Site

This usually requires no changes on the QEC side at all. Any standard OPC UA server — a PLC, a gateway, a upstream system — connects once you enter its endpoint on the Home page and press **APPLY** → **CONNECT**.

There are only three things to check beforehand:

Item	Requirement
Endpoint	<code>opc.tcp://<their IP>:<port></code> — the host part must be an IP , not a computer name (§2.2.1)
Security	They must offer SecurityPolicy None and Anonymous ; this version has no encryption compiled in
Nodes	Defined by them. Browse straight from the panel — no node list needed in advance

On running equipment, **READ** first to see **Access** and **DataType**, then decide whether to **WRITE**.

9.2 Two QECs Paired

One runs the Server tool, the other runs this Client tool, with no PC in between. It is the fastest way to verify a connection end to end, and with a Log page on each side it is easy to tell which end a problem is on.

The ten **QEC.*** nodes on the server side are documented in [QEC OPC UA Server Test Tool — User Manual](#); **QEC.Counter** increments every second, which makes it the best node to try **LIVE** and **WATCH** on.

9.3 The QEC as a Server Instead

If what you need is "the QEC exposes its own data and SCADA connects in", that is the other direction — see the Server manual above. Both roles can coexist in one sketch; the library supports either side.

9.4 Possible Extensions

Direction	What it involves
A filter on the browse list	Against a server with hundreds of children per folder, paging is slow. A filter field is the obvious next step, at the cost of one of the ten rows
Expanding Variables that have children	Extend the drill-down rule beyond Object so DataAccess structures expand the way they do in a PC client
More writable types	<code>doWrite()</code> currently covers the scalar built-ins. Arrays, LocalizedText and ByteString each need their own input handling
Adding EtherCAT	The non-blocking <code>loop()</code> already leaves room: update EtherCAT after each <code>UA_Client_run_iterate()</code> and turn subscribed node values into target positions

9.5 Security Before You Go Live

This version is built without **UA_ENABLE_ENCRYPTION**, so it can only reach servers offering SecurityPolicy None with Anonymous authentication — which also means **the traffic is plaintext on the wire**. That is fine for testing and for a closed production segment, but before it reaches an office network or the outside world, isolate it behind a firewall or on a separate segment.

10. Contact Us

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

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