Arduino Opta · 文件中心

MES I/O Gateway

工業級物聯網閘道器 · 完整技術手冊

文件版 2026.06

目 次

00 快速上手

快速上手

> 📅 最後更新:2026-06-26 | 📌 對象:第一次拿到 MES Gateway 的客戶

這份是**起手式**:跟著做,你會在約 30 分鐘內讓設備上線、接上一台 RS485 量測設備、並把它的數值透過 MQTT 發出去。全程用**瀏覽器**操作,不需要寫程式。

> 想深入每個功能,做完這份後看〈使用者操作手冊〉。安裝接線細節看〈硬體安裝與接線〉。卡住了看〈疑難排解〉。

01

你會用到

- MES Gateway 設備(Arduino Opta)一台,已接電源 + 網路線。
- 一台同網段的電腦/筆電(開瀏覽器)。
- 一台 RS485 量測設備(電表 / 磅秤 / 計米器…)+ A/B 兩線接到設備的 RS485 端子。
- 設備的雲端帳號已開通(向供應商索取)。

02

步驟 1:開機 + 找到設備網頁

1. 設備通電,等綠燈穩定(約 30 秒)。
2. 在同網段電腦的瀏覽器輸入設備 IP(預設由 DHCP 取得;不確定問網管,或看〈新裝置配置〉)。

3. 看到 Dashboard 即代表設備已上線。

✅ **過關**:瀏覽器出現 MES Gateway 儀表板(不是「應用載入失敗」)。

> 若出現「應用載入失敗」→ 見〈疑難排解〉(多半是韌體用裸 .bin 燒、缺網頁資產,要用 .mesb 整包)。

03

步驟 2：申請授權 + 等雲端核准

設備出廠未授權,首次上線會自動向雲端**申請**。

1. Dashboard 上會看到「等待授權 / 待審核」狀態。
2. 請供應商在雲端管理後台**核准**這台設備(用設備的 UID 對應)。
3. 核准後約 1～2 分鐘(設備心跳週期),Dashboard 轉為 **ACTIVE(已授權)**。

✅ **過關**:狀態變 ACTIVE,沒有紅色「授權已撤銷」橫幅。

> MQTT 連線**出廠已預設好官方 broker(TLS 加密)**,你不需要自己填帳密。

04

步驟 3：掃描並加入 RS485 設備(選「型號樣板」自動帶暫存器)

1. 進「設備 / RS485」頁 → 先按「**通訊埠設定**」把該埠的 **鮑率 / 同位** 設成跟你的設備一致(常見 9600 / 8N1;部分磅秤是 8N2)→ 儲存。

> 出廠預設 19200,不改成設備實際值會掃不到。

2. 按「🔍 **掃描匯流排**」(約 30–60 秒)。
3. 逐一按「**加入**」,在表單裡**選「設備類型(型號樣板)」** → 選到對應樣板後,**系統會自動帶好該型號的暫存器**(下表),你幾乎不用手 key。沒有對應樣板的,選「**自訂**」(見下節)。
4. 記下每台的 **Slave ID**(設流程要用)。

✅ **過關**:總覽頁出現設備卡片,在線、即時值跳動。

> 掃不到 → 見〈疑難排解〉(baud/parity/接線/Slave ID 重複)。

目前支援的型號樣板

樣板	適用設備	自動帶的暫存器 / 欄位
Finder 6M(電力分析儀)	Finder 6M 系列電表	電壓 voltage / 電流 current / 功率 power / 無效功率 reactive / 視在 apparent / 功因 pf / 頻率 freq / THD / 電能 kwh(內建,免設)
SF965(計數器)	SF965 計數器	計數值(@4,÷256)等
TDA08B(磅秤)	TDA08B 秤重模組	實時重量(@11,÷100)/ 穩定狀態 / 操作通道(@107)/ 標定操作(@113)
自訂	上面以外的任何 Modbus RTU 設備	無 — 自己照設備暫存器表新增(見下節)

> 套用樣板後仍可在設備編輯視窗按「**套用型號預設暫存器**」重帶,或微調(例如磅秤小數點不同時改 scale)。

自訂設備:照你設備的暫存器表設定

樣板沒涵蓋的設備,選「**自訂**」,然後**對照你設備的「Modbus 暫存器對應表」**(在設備原廠手冊 / datasheet 裡)逐一新增暫存器。你需要從手冊查到每個量測值的:

要查的資訊	在 UI 對應欄位	說明
暫存器位址	**位址**(可填十進位或 `0x` 十六進位)	例:重量在 0x000B → 填 `11` 或 `0x0B`
資料型別	**資料型別**	16-bit / 32-bit、有號 / 無號、float;不確定先試 16-bit 有號
位元組順序	**Endianness**	32-bit 值跨兩個暫存器時的高低位順序(big / little);讀出來亂跳就換
單位 / 小數點	**倍率(scale)** + 乘/除	例:raw 1234 代表 12.34 → scale 100、除;raw×256 → scale 256、除
讀或寫	**模式**	讀(輪詢顯示)/ 寫(事件觸發,如標定歸零)

步驟:加入設備選「自訂」→ 在「自訂暫存器」區按「+」→ 填上面欄位 + 取個**名稱**(中文可,例「實時重量」;這個名稱就是模板裡 `${rs485.<SlaveID>.名稱}` 用的)→ 啟用 → 儲存。

> 🔎 **驗證位址抓對沒**:存好後到總覽頁看該暫存器即時值是否合理;或用設備頁的「讀取」對某位址試讀(進階:`POST /api/modbus/probe`)。值不對 → 多半是位址、資料型別或 endianness 要調。

05

步驟 4:設一條「MQTT 發佈流程」

目標:把某個量測值(例如重量、電力、計數)定時或變化時發到 MQTT。一條流程 = **閘(輸入群組)** + **發佈通道(Converter)**。

4a. 先建「閘」(輸入群組) — 決定何時發

- 進「輸入群組」→ 新增 → 名稱例 `發佈閘`。

- 觸發類型 **Always**、來源「📨 MQTT 命令訊號」、索引填一個你要用的通道號(例 `1`)、運算子 **≥**、門檻 **1**、**啟用** → 儲存。
- 意義:之後對 `…/cmd/signal/1` 送非 0 = 開始發、送 `0` = 停。

4b. 再建「發佈通道」(TCP 通道 / Converter)

- 進「TCP IO 通道」→ 新增 → 模式選 **Converter**、協議 **MQTT**。
- 名稱例 `發重量`(**不要和閘同名**)。
- Topic: `mes/gateway/<你的設備UID>/weight`(UID 在系統資訊頁看)。
- **JSON 模板**:用 `${rs485.<SlaveID>.<欄位>}` 取值。
  - 例(磅秤 Slave ID = 3,自訂暫存器名「實時重量」): `{"weight":${rs485.3.實時重量}}`
  - 電力設備欄位用英文名: `voltage/current/power/pf/freq/kwh` 等,例 `{"v":${rs485.1.voltage}}`
  - **N 就是設備的 Slave ID**(不是「第幾台」),刪/重排設備都不會跑掉。
- **發佈時機**(三選一,見下表)。
- 🔒 **門控信號**:選剛剛建的「發佈閘」。
- 啟用 → 儲存。

發佈時機 — 三種模式

模式	怎麼設	行為	適合
固定間隔	推送間隔填秒數(例 5)	每 5 秒發一次	一般定時上報
變更時才發	勾「🔁 變更時才發」、間隔填 0	值變才發、沒變不重複	計數器、狀態
由命令決定間隔	(進階,API)送到閘的數字=秒數	送 5→每 5 秒、送 0→停	遠端動態調整

✅ **過關**:配置頁出現「發佈閘」+「發重量」兩項。

06

樣板設備 — 三個現成案例(照抄即可)

下面是三種常見樣板設備的完整流程設定。**把 `<Slave ID>` 換成你掃到的實際 Slave ID**,Topic 裡的 `<UID>` 換成你的設備 UID(系統資訊頁)。每個案例都是「先建閘 → 再建發佈通道」。

案例 A：電力即時上報(Finder 6M)

項目	值
閘(輸入群組)	名稱 `電力閘`、Always、來源 MQTT 命令訊號 索引 `1`、≥、門檻 `1`
發佈通道	名稱 `電力發佈`、MQTT、Converter、Topic `mes/gateway/<UID>/power`
發佈時機	**固定間隔**,推送間隔 `5`
JSON 模板	`{"v":${rs485.<Slave ID>.voltage},"i":${rs485.<Slave ID>.current},"p":${rs485.<Slave ID>.power},"pf":${rs485.<Slave ID>.pf},"hz":${rs485.<Slave ID>.freq},"kwh":${rs485.<Slave ID>.kwh}}`
門控信號	電力閘

案例 B：計數上報(SF965)— 變更時才發

項目	值
閘	名稱 `計數閘`、Always、MQTT 命令訊號 索引 `2`、≥、門檻 `1`
發佈通道	名稱 `計數發佈`、MQTT、Converter、Topic `mes/gateway/<UID>/count`
發佈時機	🔁 **勾「變更時才發」**、推送間隔 `0`(計數值有變才發、沒變不重複)
JSON 模板	`{"count":${rs485.<Slave ID>.計數值}}`
門控信號	計數閘

案例 C：磅秤即時重量(TDA08B)

項目	值
閘	名稱 `磅秤即時閘`、Always、MQTT 命令訊號 索引 `3`、≥、門檻 `1`
發佈通道	名稱 `磅秤即時發佈`、MQTT、Converter、Topic `mes/gateway/<UID>/scale/weight`
發佈時機	**固定間隔**,推送間隔 `5`
JSON 模板	`{"weight":${rs485.<Slave ID>.實時重量},"stable":${rs485.<Slave ID>.穩定狀態}}`
門控信號	磅秤即時閘

> 💡 模板變數**直接從通道對話框的「可用變數參考」面板複製**最保險(面板已用你這台的 Slave ID 列好)。

案例 D(進階):磅秤標定後再發重量

比案例 C 多「先歸零」的動作 —— 多一個**輸出群組**+一條**規則**:

1. **閘**:名稱 `磅秤標定閘`、觸發類型 **Rising**(只在命令當下觸發一次)、MQTT 命令訊號 索引 `0`、≥、門檻 `1`。
2. **輸出群組(動作)**:名稱 `磅秤標定`、動作=對 Modbus 寫入 → 目標「標定操作」暫存器、寫入值 `1`。
3. **規則**:當「磅秤標定閘」成立 → 執行動作「磅秤標定」(少了這條,命令不會觸發歸零)。
4. **發佈通道**:同案例 C 的模板,門控信號選「磅秤標定閘」。

07

步驟 5：觸發,看資料發出

1. 進「工作流程」頁,找到你的閘,把「發送開關」打開(等同對 `…/cmd/signal/1` 送 1)。
2. 用任一 MQTT 工具訂閱你的 Topic(`mes/gateway/<UID>/weight`)。

3. 應看到 JSON 持續(或變化時)送出。

✅ **過關**:訂閱端收到 `{"weight":1.00}` 之類的資料。關掉開關 → 停止發送。

🎉 完成!設備已上線、授權、接上 RS485、並把資料發上 MQTT。要設更多流程(多台設備、標定、計數),重複步驟 4 即可;細節見〈使用者操作手冊〉。

00 快速上手

首次部署指南

📅 最後更新：2026-06-16 | 📌 負責人：KC

本指南帶你從零開始，在 30 分鐘內將 MES Gateway 部署到工廠產線。

01

前置準備

項目	說明
Arduino Opta	已有韌體（或 USB 線 + 電腦用於燒錄）
24V DC 電源	工業級穩壓電源
Ethernet 網線	連接至工廠 LAN
電腦/手機瀏覽器	進入 Web 管理介面

02

Step 1：接線與上電（5 分鐘）

1. 連接 24V DC 電源至 Opta 電源端子
2. 連接 Ethernet 網線至 Opta RJ45 埠
3. 上電，等待 3 秒自檢完成（LED 閃爍 → 常亮 = 正常）

> ⚠ **若 LED 持續紅色快閃** = 韌體異常，參見 故障排除

03

Step 2：找到設備 IP（2 分鐘）

方式 A：查看路由器 DHCP 列表

設備會透過 DHCP 自動取得 IP，在路由器管理頁查找主機名 `MES-Gateway` 或 MAC 開頭為 `00:60:...`

方式 B：使用 AP 模式（僅 Opta WiFi 8320）

如果沒有 Ethernet 或需要初始設定：

1. **按住 USER 按鈕** → 同時 **按一下 RESET**
2. 繼續按住 USER，觀察 LED（**8320 三段**）：
   - 0-5 秒常亮 → 5-10 秒慢閃（500ms，放開 = AP 模式）→ 10 秒+ = 原廠重置
   - **8310 無 WiFi 為兩段**：0-5 秒正常 / 5 秒+ 即原廠重置（無 AP 段）。
   - 「原廠重置」**只刪設定、留授權憑證、不清分割區**（型號感知 WI-152，詳見 新機初始化 附錄）。
3. 用手機連接 WiFi：`MES-Gateway-Setup` / 密碼 `12345678`
4. 瀏覽器打開 `http://192.168.3.1`

> ⚠ **Opta RS485 (8310) 無 WiFi，進不了 AP 模式。** 8310 請改走有線網路，或 USB 燒「拓荒包（Pioneer）」用自包設定頁開荒（見 新機初始化）。心跳燈：8320 藍燈、8310 琥珀燈（皆為正常運作）。

04

Step 3：首次組態（10 分鐘）

進入 Web 管理介面後：

3.1 網路設定

- 導航到 **設定 → 網路**
- 確認 Ethernet IP 正確
- 若需 WiFi 備援：填入 WiFi SSID/密碼

3.2 MQTT 連線

- 導航到 **設定 → MQTT**
- 填入 Broker 位址（例：`mqtt://192.168.0.100:1883`）
- 設定 Client ID（建議：`mes-gw-{產線編號}`）
- 點擊「測試連線」確認

3.3 I/O 命名

- 導航到 **配置 → I/O**
- 為每個 DI/DO/AI 通道設定有意義的名稱
- 例：DI-1 = `壓力開關`、DO-1 = `警報燈`

3.4 規則引擎（選配）

- 導航到 **規則**
- 新增規則：例「DI-1 觸發 → DO-1 動作」
- 設定觸發條件和動作

05

Step 4：驗證（5 分鐘）

1. **總覽頁面** — 確認所有 I/O 卡片顯示即時數值

2. **MQTT** — 在 Broker 端確認收到 Topic 發佈
3. **規則** — 手動觸發 DI 確認 DO 動作

06

Step 5：持久化與備份（3 分鐘）

- 所有設定會自動儲存至 QSPI Flash
- 建議：透過 Config Server 建立第一次備份

```
curl -X POST http://{gateway-ip}/api/config/push \
  -H "Authorization: Bearer admin-token"
```

07

故障排除

症狀	原因	處理
LED 紅色快閃	Stack Overflow / 韌體異常	雙擊 RESET 進入 DFU，重新燒錄
Web 介面空白	QSPI 缺網頁資產（`app.js`/CSS/語系）	重灌官方 `.mesb`（帶齊 fw＋6 資產），或燒拓荒包 `mes-gateway-bootstrap.bin` 後上傳 `.mesb`（`upload_web.py` 已廢除）
MQTT 連不上	Broker 位址錯誤或防火牆	檢查 IP/Port，確認 1883 開放
WiFi AP 看不到	QSPI 未釋放	重新按 USER+RESET 流程
設定消失	config.json 損壞	從 Config Server 還原備份

08

Step 6：進階功能佈建（選配）

以下為依現場需求的一次性佈建，設定皆持久化於 QSPI（隨設備保留）。

6.1 TDA-08B 秤重校正引導精靈

若現場有 TDA-08B 秤重模組，可佈建**引導式校正精靈**（4 步驟 + 燈號提示，純設定）。需建立寫暫存器 + 狀態通道 + signals/actions/rules。

> 📖 設定建構步驟：`guide-calibration-wizard.md` §5；操作流程 §3。

6.2 MQTT Watchdog（強烈建議）

broker 重啟 / 網路抖動後，設備可能 MQTT 顯示已連線但收不到命令（已知 lib 行為）。佈建外部 watchdog 自動偵測 + 復原：

1. **設備端佈建 `wdprobe` signal**（一次性，隨 NVS 持久化）：

```bash
curl -X POST http://{gateway-ip}/api/signals \
  -H "Authorization: Bearer admin-token" \
  -H "Content-Type: application/json" \
  -d '{"index":4,"enabled":true,"name":"wdprobe","trigger":0,"debounceMs":0,"s
```

> ⚠️ JSON 必須 compact（冒號後無空格），Content-Length 用 byte 數。

2. **在 always-on 主機常駐 watchdog**：

```bash
python3 scripts/opta_watchdog.py --interval 30 --fails 3
```

> 📖 完整原理、參數、systemd 部署：`guide-failover-resilience.md` §4。

09

下一步

- 配置 TCP I/O 通道
- 邏輯與虛擬裝置操作
- MQTT-Chain 工作流引擎
- 備援機制與 Watchdog

01 使用者

使用者操作手冊

更新於 Tue Jun 16 2026 08:00:00 GMT+0800（台北標準時間）

適用版本：v5.9.214+（2026-06-05）
適用對象：現場工程師、設備管理員、系統整合人員

v6 更新摘要（v5.9.197 → v5.9.214）

- **發佈間隔由 MQTT payload 設定（WI-143，v5.9.214）**：定時發佈類流程（用電量、磅秤發重量）啟動命令的 **payload 數值 = 發佈間隔秒數**。送 `5` → 每 5 秒發；送 `2` → 改每 2 秒（即時生效，不用重設）；送 `0` → 停。秒數夾 1~3600。例：`{prefix}/cmd/signal/1` 送 `5`。（舊版 ≤5.9.213：payload 只當開關，間隔固定。）
- **RS485 型號模板 + 一鍵套用（WI-139/143）**：RS485 設備編輯視窗挑「設備類型」（Finder 6M / SF965 計數器 / TDA08B 磅秤 / 自訂）即帶出正確 Modbus 暫存器；既有設備可按 **「↴ 套用型號預設暫存器」**一鍵補齊。Finder 設備另有 **「⚡ 套用「用電量發佈」流程範本」**一鍵建立整套發佈流程（閘訊號 + 通道 + 模板），免手動逐項設定。
- **SF965 計數器暫存器圖修正**：當前計數值在 `0x0004`（非 0x0000）；32-bit 值需 ÷256 解碼。
- **MQTT topic 前綴 = 設備 UID**：所有命令/發佈走 `mes/gateway/{設備UID}/...`（UID 見 Settings → 系統資訊；非 MQTT 設定頁顯示的 topicPrefix）。

v5 更新摘要（v5.9.128 → v5.9.197）

- **新增「工作流程」頁（WI-128/129/130）**：把規則鏈以白話卡片呈現，操作員從畫面就能 觸發 / 開關 / 監看流程與即時狀態，無需下 MQTT 命令。詳見 §10.5。
- **對外 MQTT 設備探索/資訊（WI-131，v5.9.197）**：第三方現場管理端可發免前綴廣播 `mes/gateway/whoami`（或 `<prefix>/cmd/info`）→ 設備回 `<prefix>/info` 摘要 + 各通道 `<prefix>/info/io/<id>`（含可控制的 MQTT topic）。整合合約見 `../specs/spec-mqtt-integration.md`。
- **開關型流程（WI-130，v5.9.196）**：持續性的「閘門信號」（如磅秤的 `發重量開關`）會自成一張 **on/off 開關卡**，附即時「發送中 / 待命」狀態 —— 一鍵開始 / 停止，畫面看得出在不在動。
- **磅秤發重量 100% 通用化（WI-130）**：移除磅秤特製韌體，改用通用 Converter＋命令門控；舊主題 `cmd/scale/active` 已停用，啟停改用「發重量開關」（內部 `cmd/signal/7`）。

v4 更新摘要（v5.9.64 → v5.9.82）

- **登入流程強化（v5.9.64+）**：不再信任下拉「權限級別」選項，登入時實際 probe `/api/tokens` 判定 admin/viewer 真實權限。即使下拉誤選，UI 角色仍以 token 真實權限為準
- **/api/system auth 補上（v5.9.76）**：亂打密碼會被 401 擋下，不會誤導用戶以為登入成功

- **登入 lwIP race 修補（v5.9.75）**：兩個連續 fetch 加 250ms 間隔 + 3 次 retry，避免 單線程 HTTP 短時間內被打爆出現「網路錯誤：Failed to fetch」
- **瀏覽器 cache ETag 機制（v5.9.65）**：升級韌體後 ETag 變動 → 強制 re-fetch，不再卡 24h max-age 拿到舊 UI bundle
- **OTA 連續升級穩定（v5.9.66~78）**：修補多項 race 問題（QSPI partition 殘留、`localConfig.remount()` 干擾、`reboot` delay 中其他 task 寫 QSPI），現在連續 OTA 升級不再 brick 進 DFU
- **Settings → 使用者管理 改稱「使用者/密碼」**（v5.9.77）：UI 字眼從 token 改成 更直覺的「密碼」，admin 可在 Settings 頁直接新增/刪除/變更使用者，三語完整

v3 更新摘要（v5.9.63）

- WI-112/115：Web 登入改密碼制，admin/viewer 兩級權限
- WI-113：UI 三語切換（繁中 / 簡中 / EN，右上 🌐 自動偵測 + 強制切換）
- 雲端 admin UI 韌體列表改 card 排版（最新版高亮 + changelog details）
- Viewer 可瀏覽 + 切換語系，無法變更設定 / I/O / 規則
- OTA 加入同版號 + size 安全 guard（>780KB 強制改用 USB DFU，避免 brick）

本手冊說明 MES I/O Gateway 的核心設計理念，以及如何透過 Web Dashboard 完成安裝、設定與日常操作。

01

1. 產品概述

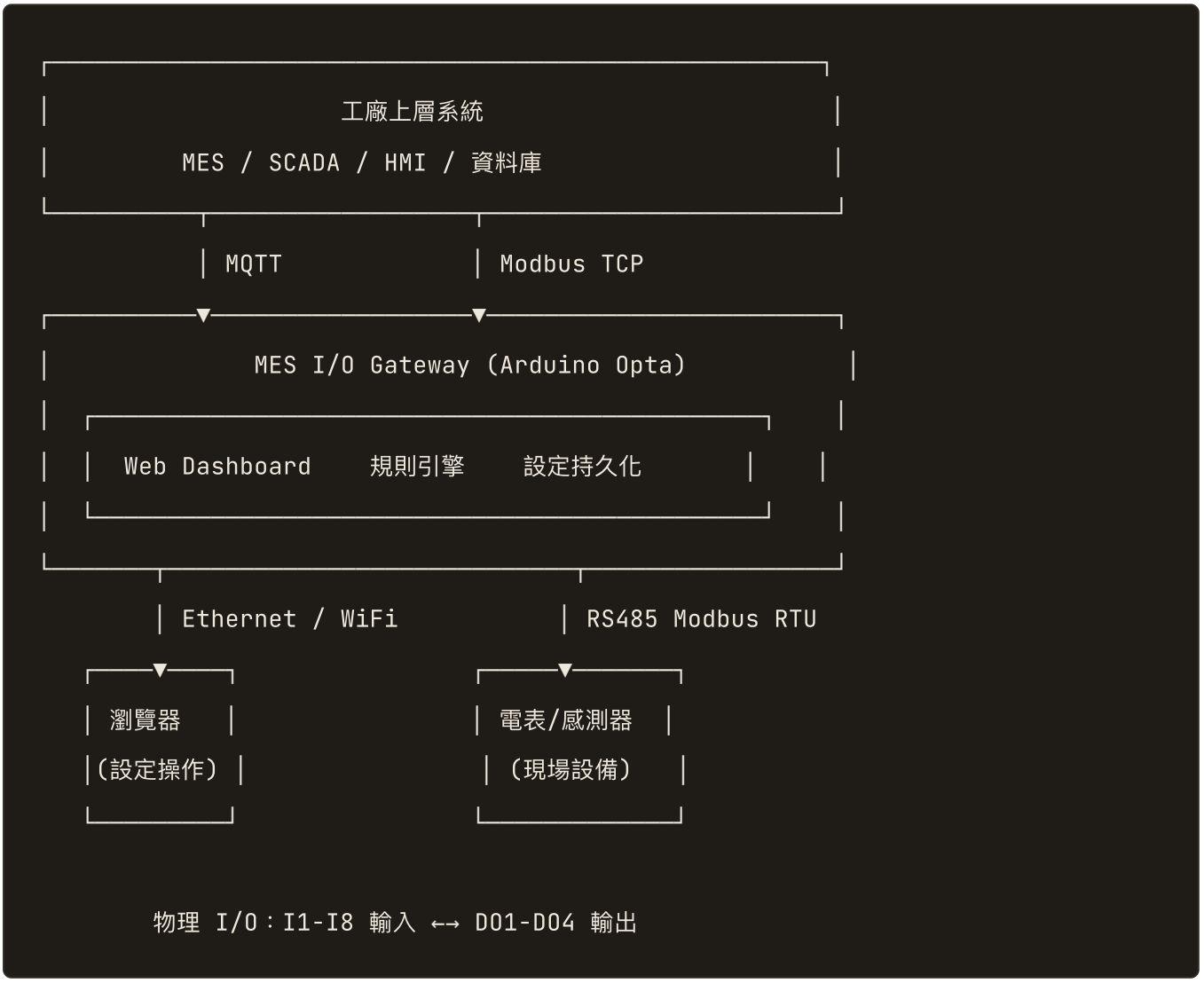
MES I/O Gateway 是一台安裝在工廠現場的智慧型 I/O 閘道器。它以 Arduino Opta 工業控制器為核心，將產線上的感測器、開關、電表等設備，透過內建的規則引擎與通訊協議，串接到工廠的 MES、SCADA 或其他上層管理系統。

一句話定位：把現場的物理訊號，變成工廠系統能理解的數位資訊，並且根據規則自動執行控制動作。

1.1 它能做什麼？

能力	說明
收集現場訊號	8 路輸入（數位或類比）、RS485 電表/感測器
執行自動控制	4 路繼電器輸出 + 擴充模組輸出
連接上層系統	MQTT、Modbus TCP 雙向通訊
現場邏輯運算	計數器、計時器、類比換算、規則引擎
遠端管理	Web Dashboard、OTA 韌體更新、Config Server 備份

1.2 系統組成



02

2. 核心設計理念

> 在開始操作 Dashboard 之前，請先花幾分鐘理解以下核心概念。掌握這些設計理念，你會發現後續的操作設定變得直覺而且有邏輯。

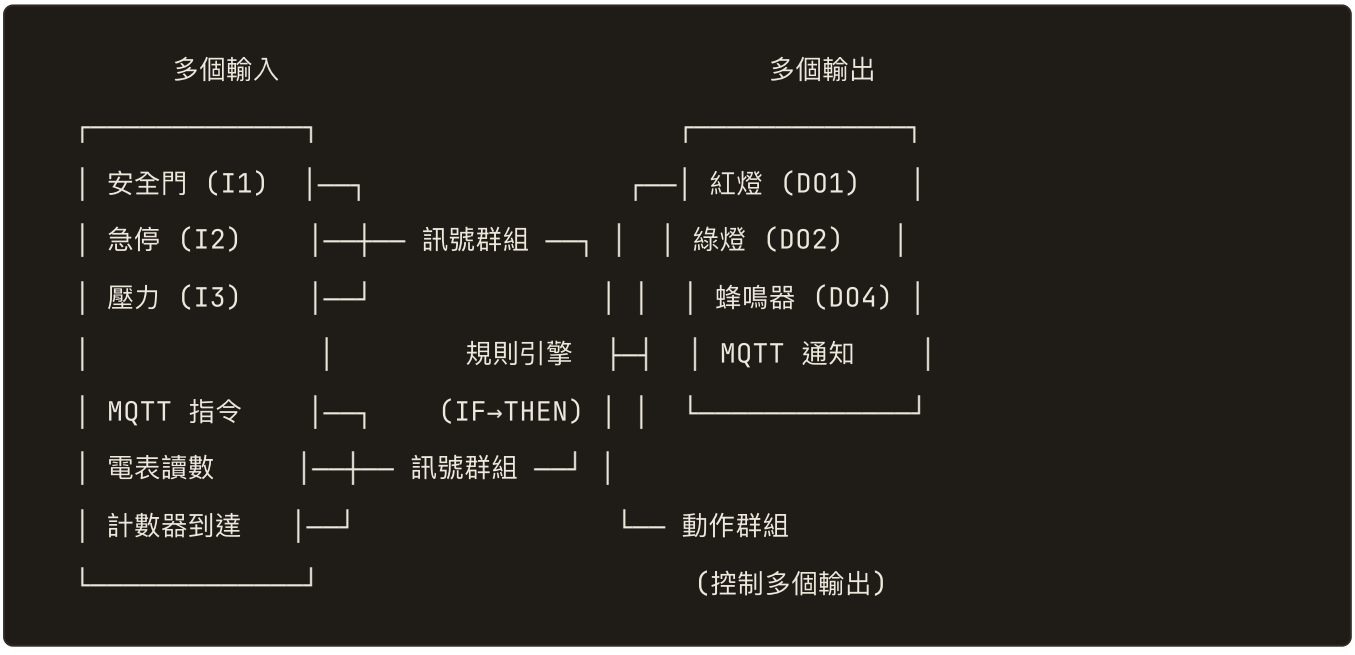
2.1 多對多架構：為什麼不是一對一？

傳統的 I/O 控制通常是「一個輸入控制一個輸出」。例如：按下按鈕 → 燈亮。

但在工廠現場，真實需求往往更複雜：

- **多個輸入 → 一個判斷**：安全門關閉 AND 急停未按下 AND 壓力正常 → 才算「可以啟動」
- **一個判斷 → 多個輸出**：「可以啟動」→ 綠燈亮 + 蜂鳴器短響 + 發送 MQTT 訊息通知 MES
- **同一個輸入 → 參與多個判斷**：「安全門」同時用於「啟動條件」和「緊急停止條件」

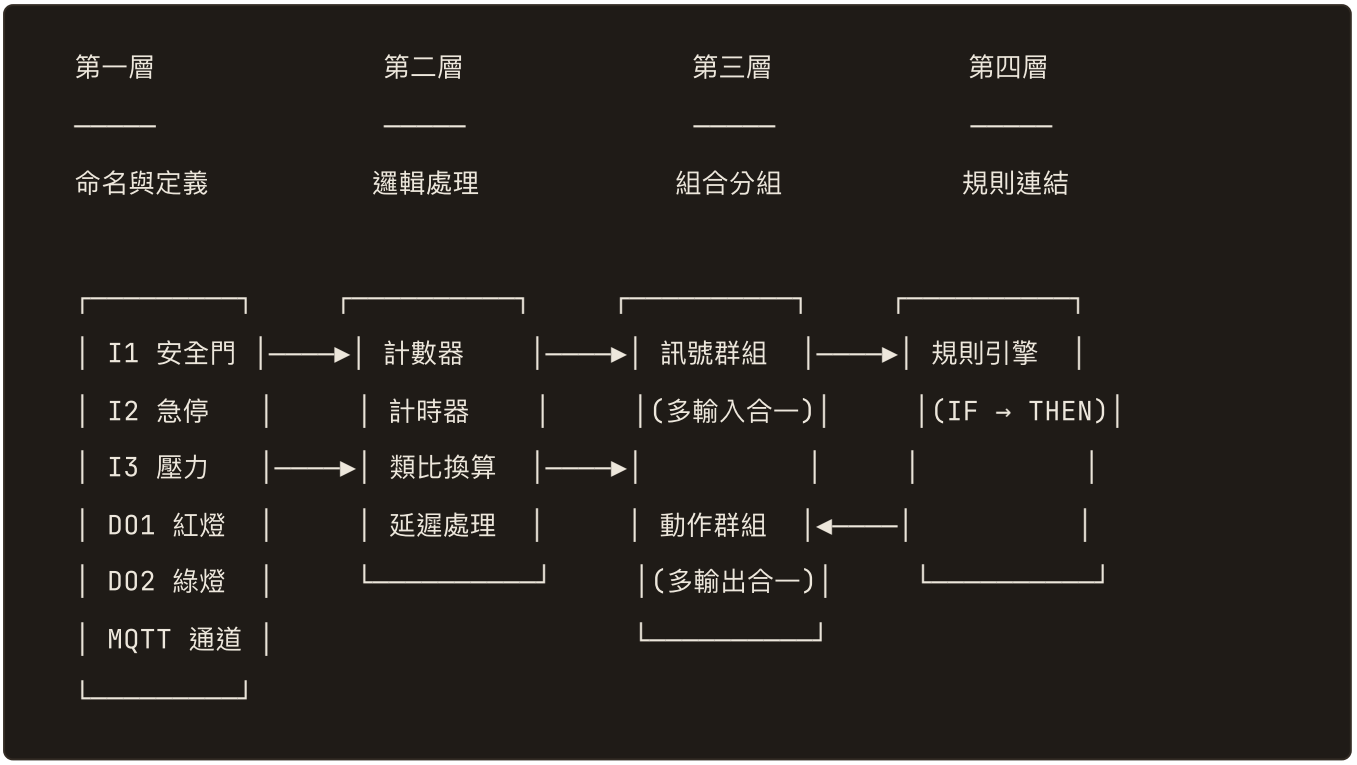
這就是 **多對多 (Multi-to-Multi)** 的設計理念：



> **關鍵好處**：你可以自由組合任意輸入和輸出，不受硬體接線的限制。改接線不需要改規則，改規則不需要改接線。

2.2 四層邏輯模型

為了實現多對多，系統將訊號處理分成四層。每一層各司其職，互不干擾：



第一層：命名與定義（你有什麼？）

把硬體端子和網路通道取一個人看得懂的名字。

原始名稱	你取的名字	模式
I1	安全門	數位
I3	壓力感測器	類比
DO1	紅燈	—
DO4	蜂鳴器	—
MQTT Topic	MES 啟動指令	Parser

> **為什麼要命名？** 之後建立群組和規則時，選單裡會顯示你取的名字而不是 I1、I3。這讓設定更直覺，也讓其他工程師一眼就知道每個通道的用途。

第二層：邏輯處理（需要計算嗎？）

有些訊號不能直接用，需要經過處理：

處理器	用途	範例
計數器	計算某個訊號觸發了幾次	I1 每觸發一次 = 生產一件，累計到 100 件就通知
計時器	測量某個狀態持續多久	I2 從 ON 到 OFF 的時間 = 這次的加工時間
類比換算	把 0-10V 電壓轉成工程單位	I3 的 0-10V 對應 0-10 bar，超過 6.5 bar 就觸發
延遲處理 ➡ SOON	訊號需持續一段時間才生效，或關閉後延長作用	安全門打開超過 3 秒才報警；排風扇停機後再轉 30 秒

> **不是每個通道都需要第二層。**如果你只是要偵測「門開了沒」，I1 的數位訊號可以直接跳到第三層使用。

第三層：組合分組（怎麼組合判斷？）

把多個訊號來源組合成一個有意義的條件（訊號群組），或把多個輸出組合成一個動作（動作群組）。

訊號群組範例（📡 輸入側）：

群組名稱	組合條件
安全條件成立	安全門 = 關閉 **AND** 急停 = 未按下
壓力異常	壓力感測器 > 6.5 bar
MES 啟動	MES 啟動指令 = "START"

動作群組範例（⚙️ 輸出側）：

群組名稱	包含的動作
警報模式	紅燈 ON + 蜂鳴器脈衝 500ms
正常模式	綠燈 ON + 紅燈 OFF
通知 MES	發送 MQTT JSON 訊息

第四層：規則連結（什麼時候做什麼？）

規則名稱	當…	就…
壓力警報	壓力異常 **成立時**	執行 **警報模式**
壓力恢復	壓力異常 **解除時**	執行 **正常模式**
MES 控制	MES 啟動 **成立時**	執行 **正常模式** + **通知 MES**

> **為什麼要分這麼多層？** 因為解耦。假設你把壓力感測器從 I3 換到 I5，你只需要在第一層改名字，第二到四層的設定完全不用動。

2.3 三大通道類型

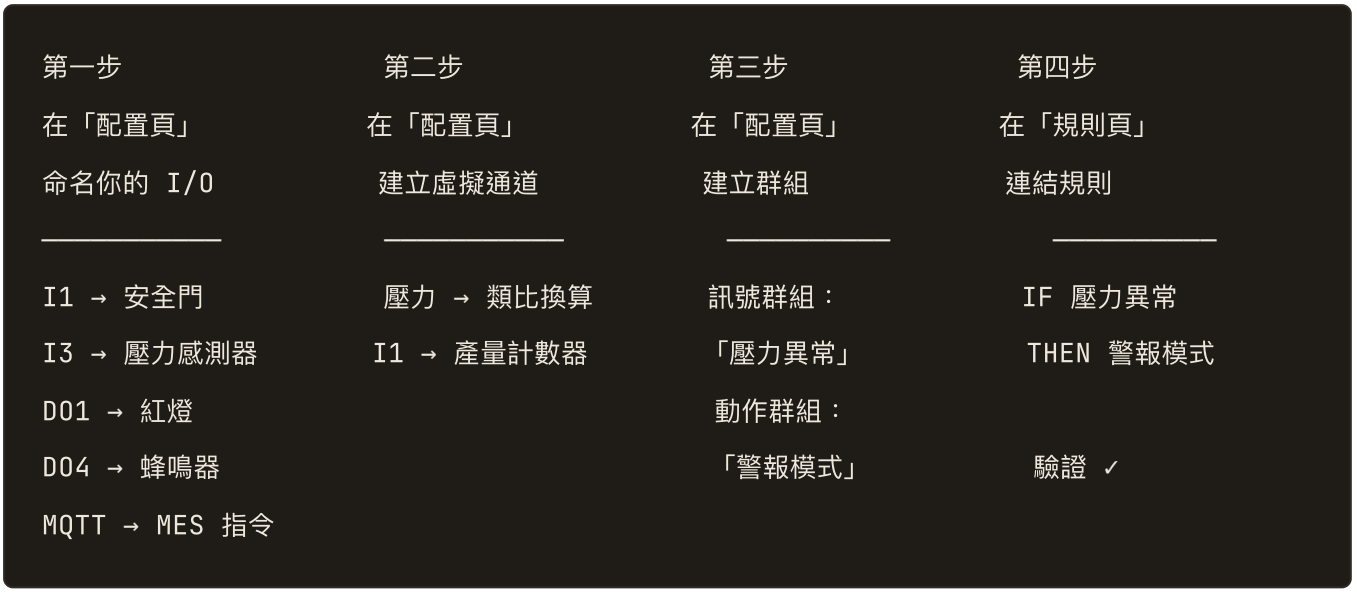
在多對多架構中，「多」不只是物理的開關和繼電器。系統支援三種通道，它們都可以自由混合在群組裡：

通道類型	輸入方向	輸出方向
物理通道	DI 數位輸入、AI 類比輸入	DO 繼電器輸出
虛擬通道	計數器到達值、計時器到時	計數重置、變數更新
網路通道	MQTT 訂閱、Modbus TCP 讀取	MQTT 發佈、Modbus TCP 寫入

混合範例：一個訊號群組可以同時包含「I1 安全門（物理）」+「計數器到達 100（虛擬）」+「MES 發來 START 指令（網路）」。這三個條件全部成立時，才觸發動作。

2.4 設定流程總覽

理解了四層模型後，實際操作的順序就是依序完成每一層：



> 後續章節將按照這個順序，帶你在 Web Dashboard 上完成每一步的操作。

03

3. 端到端範例：壓力監控場景

> 在進入各頁面的詳細操作說明之前，先用一個完整的現場情境，示範從接線到規則觸發的全過程。

情境說明

工廠有一台沖壓機，現場配置如下：

設備	接線位置	用途
壓力感測器	I3（類比 0-10V）	偵測油壓，0V = 0 bar，10V = 10 bar
安全門磁簧開關	I1（數位）	門關閉 = HIGH，門打開 = LOW
三色燈 紅	DO1	異常時亮
三色燈 綠	DO2	正常時亮
蜂鳴器	DO4	異常時短響
MES 系統	MQTT Broker	接收產線狀態 JSON

需求：壓力超過 6.5 bar 且安全門關閉時 → 亮紅燈 + 蜂鳴器響 + 通知 MES。壓力恢復正常時 → 切回綠燈。

設定步驟對照表

步驟	對應層	Dashboard 位置	操作
1	L1 命名	配置頁 → 物理通道	I1 命名「安全門」，模式 = 數位
2	L1 命名	配置頁 → 物理通道	I3 命名「壓力感測器」，模式 = 類比
3	L1 命名	配置頁 → 物理通道	DO1 命名「紅燈」，DO2「綠燈」，DO4「蜂鳴器」
4	L2 邏輯	配置頁 → 虛擬通道	新增類比邏輯「油壓監控」：I3，0-10V → 0-10 bar，閾值 > 6.5
5	L3 群組	配置頁 → 輸入群組	新增訊號群組「壓力異常」：油壓監控 = 觸發 AND 安全門 = HIGH
6	L3 群組	配置頁 → 輸出群組	新增動作群組「警報模式」：DO1 ON + DO4 脈衝 500ms
7	L3 群組	配置頁 → 輸出群組	新增動作群組「正常模式」：DO2 ON + DO1 OFF
8	L4 規則	規則頁	新增規則：IF 壓力異常 成立 → THEN 警報模式
9	L4 規則	規則頁	新增規則：IF 壓力異常 解除 → THEN 正常模式
10	驗證	總覽頁	確認 I/O 狀態、手動觸發測試

> 每個步驟的詳細畫面操作說明，請參見第 7 章（配置頁）和第 8 章（規則頁）。

04

4. 安全與操作注意事項

在接線、設定與更新前，請先確認以下事項：

- 請使用穩定的 24V DC 工業電源。
- 接線或更換 RS485 設備前，先關閉相關設備電源。
- RS485 匯流排上的每台設備必須使用唯一 Slave ID。
- WiFi 僅支援 2.4GHz 網路。
- 執行 OTA 或工廠重置前，先確認設備不在關鍵生產控制時段。
- 不要在公開文件中記錄管理員密碼、Token、雲端 Registry 密碼或 VPN 密碼。
- 若使用 USB Serial Monitor 除錯，拔除 USB 前請先關閉 Serial Monitor，避免 Mbed OS Serial 阻塞影響主迴圈。

05

5. 系統組成

MES I/O Gateway 由以下元件組成：

元件	用途
Arduino Opta	現場 Gateway 主機
Web Dashboard	使用者設定與監控介面
QSPI LittleFS	保存設定、統計與暫存資料
MQTT Broker	與 MES / SCADA / 上層系統交換訊息
RS485 / Modbus RTU	連接電表、感測器、控制模組等現場設備
Modbus TCP	透過 TCP 與 HMI / PLC / SCADA 整合
Config Server	多設備管理、設定備份、OTA 與授權

06

6. 首次上線流程

6.0 全新設備開荒（USB 拓荒包，僅第一次）

> 📦 **剛出廠 / 變磚的設備請先做這一步；已在運作的設備可直接跳到 6.1。**

全新 Opta 出廠時還沒有本專案的完整韌體與網頁。請先用 USB 燒一次**拓荒包**把設備帶起來：

1. **USB 首燒拓荒包** `mes-gateway-bootstrap.bin`（**無版號、固定一顆**，雲端公開頁「🔧 全新設備開荒」藍框可直接下載，連結永不變）：

   ```bash
   arduino-cli upload -b arduino:mbed_opta:opta -p <PORT> -i mes-gateway-bootstra
   ```

   （詳細燒錄步驟見 `guide-cli-opta.md`。）
2. **開機即見開荒設定頁**：顯示 🆔 裝置 UID（供雲端綁定/授權）+ 🌐 網路設定 + 🚀「上傳 .mesb」大按鈕。8310 會自動隱藏 WiFi 欄（型號感知）。
3. **上傳官方完整版** `.mesb` → 設備自動寫入 QSPI 並重開 → **完整版上線**（韌體 + 完整 UI 一次到位）。

> 💡 拓荒包只負責「開荒」這一次，**極少更新**；之後所有功能 / 版本更新一律走完整版 `.mesb`（見 §13）。發佈端的程序分工見 `guide-ota-embedded-web.md`。

6.1 接線與上電

1. 將 Arduino Opta 接上 24V DC 電源。
2. 若現場有有線網路，接上 Ethernet 網線。
3. 上電後等待設備完成啟動。
4. 確認狀態燈不是紅色錯誤狀態。

常見 LED 判讀：

主機 Status LED（L1-L4）= DO1-DO4 輸出狀態指示

LED	對應	亮 = ON	滅 = OFF
L1	DO1	DO1 輸出中	DO1 關閉
L2	DO2	DO2 輸出中	DO2 關閉
L3	DO3	DO3 輸出中	DO3 關閉
L4	DO4	DO4 輸出中	DO4 關閉

USER LED = 系統狀態燈號

⚠️ **心跳燈顏色依機型不同（v5.9.244+ / WI-152）**：

- **Opta WiFi (8320)**：心跳是**藍燈**（與下表「藍色慢閃」相同）。
- **Opta RS485 (8310)**：藍燈是 BLE 模組燈，官方明文「僅 WiFi 版能亮」，8310 點不亮 → 心跳改用**琥珀色（紅+綠同亮）**。看到 8310 跳琥珀慢閃 = 系統正常，不是錯誤。

模式	意義	處理說明
🔵 藍色慢閃 (1Hz)〔8320〕/ 🟡 琥珀慢閃〔8310〕	系統正常運作	設備運作正常，可進入 Web Dashboard。8310 用琥珀色取代藍色。
🔵 藍色快閃 (5Hz)〔8320〕/ 🟡 琥珀快閃〔8310〕	MQTT 斷線警告	設備已啟用 MQTT 但無法連線至 Broker，請檢查網路或設定。
🟢 綠燈呼吸閃爍	DFU 韌體更新中	正在替換底層韌體，**絕對請勿斷電** (需時 2~3 分鐘)。
🟠 橙色閃爍	AP Mode 網路設定中（僅 8320）	設備處於配網模式，請用手機連線至 `MES-Gateway-Setup` 熱點。8310 無 WiFi 進不了此模式。
🔴 紅色常亮	系統嚴重錯誤 (Panic)	系統當機，請嘗試重啟或依維修流程重新燒錄韌體。
🔴 紅色慢閃	硬體初始化失敗	WiFi 或 QSPI 模組錯誤，請依維修流程處理。

6.2 找到設備 IP

優先使用 Ethernet：

1. 將設備接上工廠 LAN。
2. 到路由器或 DHCP Server 查找設備 IP。
3. 在瀏覽器開啟 `http://<gateway-ip>/`。

若無法取得 IP，可使用 AP 設定模式（**僅 Opta WiFi 8320**）：

開機時按住 USER 按鈕，依燈號分段放開（門檻**依機型自動切換**，WI-152）：

設備	0–5s 放開	5–10s 放開	10s+ 放開
8320（有 WiFi）三段	正常開機	**AP 設定模式**	**原廠重置**
8310（無 WiFi）兩段	正常開機	**原廠重置**（5s+，無 AP 段）	原廠重置

進 AP 設定模式(僅 8320)後:

1. 使用手機或電腦連接 WiFi:`MES-Gateway-Setup`。
2. 預設密碼:`12345678`。
3. 瀏覽器開啟 `http://192.168.3.1`。

> ⚠️ **「原廠重置」只刪設定、留授權憑證、不清分割區**(僅 `config.json` / `config.bak`,重開機)。要連 QSPI 分割區一起清(從無到有)請用 QSPIFormat,見 新機初始化「從無到有」。

> ⚠️ **Opta RS485 (8310) 無 WiFi,進不了 AP 模式**(嘗試會 fallback)。8310 請改走有線網路設定,或用「拓荒包(Bootstrap)」自包設定頁(見 安裝手冊 §4.2 與 新機初始化)。韌體會用 OTP 自動分辨機型,型號感知過濾見下方「網路設定差異」。

> **網路設定差異(型號感知,WI-151)**:8310(無 WiFi)開啟「設定」頁的網路卡片時,會自動**隱藏 WiFi 帳密區與整個介面模式選單**(強制乙太網路,只留 Ethernet IP 的 DHCP/靜態設定);8320 顯示完整 WiFi + 介面模式(自動/僅有線/僅 WiFi)。前端依 `/api/system` 的 `hasWifi` 欄位判斷。

6.3 首次登入與帳號

進入 Web Dashboard 後會跳出 🔐 登入框:

欄位	說明
權限級別	🔴 管理員 (admin,全部功能) / 👁️ 唯讀 (viewer,只能查看) — **僅作提示**,實際權限以密碼對應的真實 token 為準(v5.9.64+)
密碼	出廠預設見下表,**首次登入後請立即更換**

出廠預設密碼

多數設備(含目前在線的 v5.9.82)都是 `change-me-admin` / `change-me-viewer`:

帳號	預設密碼	權限
admin	`change-me-admin`	全部功能（修改設定、規則、I/O 控制）
viewer	`change-me-viewer`	只能查看 Dashboard，無法修改

> **為什麼不是 `smmsadmin`？** WI-115（v5.9.54）把預設值「程式碼裡」改成了 `smmsadmin` / `smmsviewer`，但 `seedDefaults()` 只在 **`tokenCount == 0`** 時執行（首次 boot / factory reset）。任何升級上來的設備，QSPI LittleFS 已有舊 token 紀錄（`tokenCount > 0`），不會 reseed，所以仍是 `change-me-admin` / `change-me-viewer`。
>
> **新出貨的全新設備**（QSPI 從未寫過 config）才會看到 `smmsadmin` / `smmsviewer`。
>
> 想確認你的設備：USB 接上 → serial 115200 baud → 開機 log 會看到 `[Token] Loaded N tokens from config`（升級設備）或 `[Token] Seeded defaults (admin / viewer). PLEASE CHANGE PASSWORDS IMMEDIATELY.`（首次 boot 才會 seed 新預設值）。

首次登入後請立即更換密碼（見下方「變更密碼」流程）。

登入流程說明（v5.9.64+）

1. submitLogin 發 `GET /api/system` 帶 Bearer token → 401 = 密碼錯誤；200 = token 有效
2. 250ms 後再發 `GET /api/tokens` probe → 200 = admin；401 = viewer/control（最多 retry 3 次防 lwIP busy）
3. 三次 probe 全失敗 → fallback 用 token 字面比對預設值（`smmsadmin` / `change-me-admin` → admin）
4. 登入成功 → localStorage 存 token + 真實 perm，body class 套 `perm-admin` 或 `perm-readonly`

> **下拉「權限級別」誤選不會卡死 UI**：v5.9.64 之前若下拉選 viewer + 用 admin 密碼登入，UI 會被 perm-readonly CSS 鎖死所有按鈕。現在以 `/api/tokens` probe 為準，下拉只是 UI 提示。

變更密碼（admin only）

1. 以 admin 登入
2. 進「設定」頁 → 「👥 使用者管理」卡片
3. 點 admin 那組旁邊的「💾」儲存（先在密碼欄輸入新密碼 4-31 字元，留空＝保留原值）
4. 也可按「🗑」刪除（系統會防呆，不允許刪掉最後一個 admin token）
5. 「➕ 新增」可加最多 4 組使用者，每組可獨立設定權限（viewer / control / admin）
6. 變更後重新登入確認新密碼可用

切換語系

右上角下拉「🌐 自動」可選 🇹🇼 繁中 / 🇨🇳 簡中 / 🇺🇸 EN。

- **自動**：依瀏覽器設定（zh-TW/zh-HK → 繁中、其他 zh → 簡中、其他 → EN）
- 強制切換後存於 localStorage，下次開啟保留設定
- viewer 也可切換語系（純 UI 偏好，不需 admin 權限）

登入後檢查

- 左側顯示的設備名稱與 IP
- 總覽頁是否能顯示系統狀態
- 設定頁的網路狀態是否符合現場需求
- 若需要 MQTT，先設定 Broker 後再測試連線
- 右上角徽章：🔴 admin / 👁️ viewer — 確認與你登入的帳號一致

> **權限差異**：viewer 進站後會看到所有「儲存 / 切換 / 刪除 / 新增」按鈕呈灰階（pointer-events:none），可瀏覽但點不到。**例外**：語系下拉、輪詢頻率下拉、登出按鈕、關閉 Login modal 按鈕仍可用（純 UI 偏好不算寫入）。

> **權限不足提示**：若 viewer 不小心觸發了需要 admin 的功能（如 API 直接呼叫），會跳出 ⛔「權限不足，此操作需要 admin 權限」toast，**不會**重新跳出登入框（避免誤導用戶以為密碼過期）。

瀏覽器 cache 提示（v5.9.65+）：升級韌體後若還看到舊 UI 行為，按一次 Cmd+Shift+R（Mac）/ Ctrl+Shift+R（Windows）強制重整即可。之後 ETag 機制會自動帶你抓最新 bundle，不再需要手動清 cache。

07

7. Web Dashboard 導覽

Dashboard 主要頁面如下：

頁面	用途
總覽	監看即時 I/O、系統狀態、MQTT、RS485 與統計資料
配置	命名 I/O、設定輸入模式、管理 TCP IO 與邏輯裝置
規則	建立訊號群組、動作群組與自動化規則
工作流程	把規則鏈以白話卡片呈現，畫面直接觸發 / 開關 / 監看流程（見 § 10.5）
設備	管理 RS485 / Modbus 設備與匯流排工具
設定	管理網路、MQTT、Config Server、OTA、重置等系統設定

在 AP Mode 下，Dashboard 會聚焦在 WiFi / 網路設定，部分工業控制頁面會隱藏，避免設定模式下誤操作。

08

8. 總覽頁操作

總覽頁面 (Dashboard)

總覽頁是系統的入口，提供所有實體與虛擬 I/O 的即時狀態監控。

可確認的內容：

- Gateway 是否在線。
- 當前網路介面：Ethernet 或 WiFi。
- MQTT 是否已連線。
- I1-I8 輸入狀態。
- DO1-DO4 與擴充輸出狀態。
- RS485 設備是否在線。
- 生產計數、類比數值、虛擬裝置狀態。

操作建議：

1. 班前先確認所有關鍵設備顯示在線。
2. 若 I/O 命名仍是預設名稱，先到「配置」頁補上現場名稱。
3. 若某個通道呈現灰色或未使用，表示它沒有被納入目前的訊號或動作群組。
4. 重置統計前，先確認是否已備份或記錄本班次資料。

09

9. 配置頁操作

配置頁面 (Config)

配置頁用於設定 I/O 節點的功能與通訊。畫面分為 Host（主機）與 Expansion（擴充模組）兩大區塊。建議依序完成 Layer 1 到 Layer 3：

1. Layer 1：定義實體 I/O 與設備名稱。
2. Layer 2：建立訊號群組與動作群組。
3. Layer 3：在規則頁連接條件與動作。

9.1 設定 I1-I8 輸入

I1-I8 是共用輸入，可依現場用途設定為數位或類比。

操作步驟：

1. 進入「配置」頁。
2. 找到 I1-I8 通道列表。
3. 為每個通道輸入易懂名稱，例如 `安全門`、`壓力感測器`、`入料偵測`。
4. 選擇模式：
   - Digital：用於開關、接點、光電、近接等 0/24V 訊號。
   - Analog：用於 0-10V 類比訊號。
5. 儲存後回到總覽頁確認顯示是否正確。

9.2 設定 DO1-DO4 輸出

操作步驟：

1. 進入「配置」頁。
2. 找到輸出通道。
3. 為 DO1-DO4 命名，例如 `紅燈`、`綠燈`、`蜂鳴器`、`排風扇`。
4. 若輸出要由規則控制，後續需在「規則」頁加入動作群組。

9.3 類比邏輯處理

Analog Logic 可用於比例換算、單位顯示、門檻判斷與濾波。

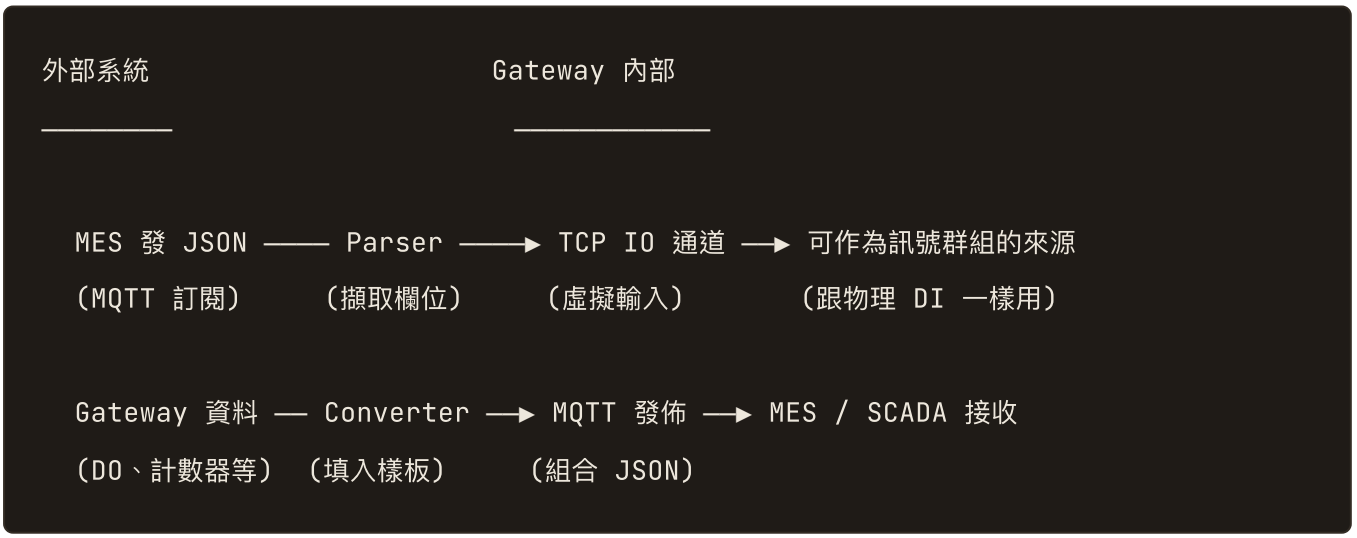
常見設定：

欄位	說明
名稱	顯示於 Dashboard 與規則選單
AI / Input	綁定的類比輸入
Min / Max	0-10V 對應的工程值範圍
Unit	單位，例如 `bar`, `kg`, `V`, `%`
Operator	判斷條件，例如大於、小於、區間內
Threshold	觸發門檻
Deadband	防止臨界點抖動
Delay	需持續多久才判定觸發

9.4 TCP IO 通道（網路通道）

TCP IO 是讓 Gateway 和外部系統（MES、SCADA、其他設備）雙向溝通的橋樑。它把網路上的資料變成規則引擎可以使用的「虛擬輸入」，也可以把 Gateway 的即時資料組合成訊息發送出去。

運作概念



簡單來說：

- **Parser（解析器）** = 從外面收資料進來，變成 Gateway 看得懂的訊號
- **Converter（轉譯器）** = 把 Gateway 的資料包裝好送出去

三種模式說明

模式	方向	用途	適用場景
⚡ Legacy（收取觸發）	輸入	收到特定字串就觸發	MES 發 `"START"` 指令啟動產線
📩 Parser（擷取數值）	輸入	從 JSON 中擷取特定欄位的值	MES 發來含多個欄位的 JSON，需要解析
📤 Converter（組合發佈）	輸出	把 Gateway 資料組合成 JSON 送出	定時回報產線狀態給 MES

Parser 詳細操作：從 MQTT JSON 擷取數值

場景：MES 透過 MQTT 發送以下 JSON 到 Topic `factory/line1/status`：

```JSON
{
  "machine_id": "CNC-001",
  "pressure": 6.8,
  "temperature": 42.5,
  "door_open": true,
  "status": "RUNNING"
}
```

你想把 `pressure` 和 `door_open` 擷取出來，讓規則引擎可以用。

操作步驟：

1. 進入「配置」頁 → 右側「TCP 通道」卡片 → 點擊 **+ 新增**
2. 設定基本資訊：

欄位	填入
通道名稱	油壓數值
協議	🔵 MQTT
模式	📩 收取並擷取數值 (Parser)
MQTT Topic	`factory/line1/status`

3. 新增擷取欄位：

JSON 欄位名	說明
`pressure`	系統會自動從 JSON 中找到 `"pressure": 6.8`，擷取出 `6.8`
`door_open`	系統會擷取出 `true`，對應為 HIGH

4. 點擊「儲存」。系統會自動訂閱該 MQTT Topic。
5. 回到「配置」頁左側的「虛擬通道」，你會看到新建的 TCP IO 通道已經出現在列表中，可以在建立訊號群組時選取它作為來源。

> **多欄位擷取**：一個 MQTT Topic 可以同時擷取多個欄位。Web UI 提供「新增擷取欄位」按鈕，每個欄位會自動建立一個獨立的 TCP IO 通道。

> **安全機制**：如果 MQTT 斷線或 JSON 格式錯誤，系統會維持「最後有效值」，不會讓規則因為收不到資料而亂跳。你也可以設定「過期時間」，超過 N 秒沒收到更新就自動標記為失效。

Converter 詳細操作：組合 JSON 發佈到 MQTT

場景：你希望 Gateway 每次規則觸發時，自動發送一則 JSON 到 MES 系統的 Topic `factory/line1/gateway_report`：

```JSON
{
  "device": "Gateway-A",
  "pressure_bar": 6.8,
  "door_closed": 1,
  "production_count": 42,
  "uptime_sec": 3600
}
```

操作步驟：

1. 進入「配置」頁 → 右側「TCP 通道」卡片 → 點擊 + 新增
2. 設定基本資訊：

欄位	填入
通道名稱	產線報告
協議	🔵 MQTT
模式	📤 組合並發佈訊息 (Converter)
MQTT Topic	`factory/line1/gateway_report`

3. 編寫 JSON 樣板（Template）：

```JSON
{
  "device": "Gateway-A",
  "pressure_bar": ${io.I3},
  "door_closed": ${io.I1},
  "production_count": ${vc.CNT1},
  "uptime_sec": ${sys.uptime}
}
```

4. 樣板中的 `${...}` 是變數標籤，Gateway 在發送時會自動替換成即時數值。

5. 儲存後，在「規則」頁的動作群組中，就可以選取這個 Converter 通道作為輸出動作之一。

可用的變數標籤一覽

在 Converter 樣板中，你可以使用以下標籤。Dashboard 的樣板編輯器右側會提供可點擊複製的標籤列表。

分類	標籤格式	範例	說明
物理 I/O	`${io.I1}` ~ `${io.I8}`	`${io.I3}` → `6.8`	輸入值（數位=0/1，類比=工程值）
	`${io.DO1}` ~ `${io.DO4}`	`${io.DO1}` → `1`	輸出狀態
虛擬通道	`${vc.名稱}`	`${vc.CNT1}` → `42`	計數器/計時器的目前值
RS485 設備	`${rs485.SlaveID.欄位}`	`${rs485.1.V}` → `220.5`	RS485 設備的暫存器讀數
系統資訊	`${sys.uptime}`	`${sys.uptime}` → `3600`	開機秒數
	`${sys.ip}`	`${sys.ip}` → `192.168.1.100`	設備 IP
TCP IO	`${tcpio.名稱}`	`${tcpio.油壓數值}` → `6.8`	其他 TCP IO 通道的值
原始封包	`${raw_payload}`	（完整 JSON）	最近一次收到的 MQTT 原始訊息

> **小數位數修飾詞（v6.0.5+）**：數值變數後可加 `:N` 指定小數位數 —— `${rs485.1.計數值:0}` → `107`（**整數**，四捨五入）、`${rs485.1.計數值:2}` → `107.00`（兩位），不加 `:N` = 預設 2 位。**計數器建議用 `:0` 發整數。**需韌體 6.0.5+（舊韌體會把 `:0` 當成暫存器名的一部分而讀不到值）。

進階用法：Process-and-Forward 如果你想讓 Gateway 收到一則 MQTT 訊息後，加上 Gateway 自己的資料再轉發出去，可以在 Converter 樣板中使用 `${raw_payload}` 把原始訊息嵌入：

```JSON
{
  "original": ${raw_payload},
  "gateway_ip": "${sys.ip}",
  "local_door": ${io.I1}
}
```

Modbus TCP 雙向整合

除了 MQTT，TCP IO 也支援 Modbus TCP 協議，適用於需要與 PLC、HMI 或 SCADA 直接整合的場景。Gateway 同時扮演 **Server** 和 **Client** 兩種角色：

情境 A + B：Gateway 作為 Modbus TCP Server

Gateway 開機後自動在 Port 502 監聽。外部 PLC 或 HMI 可以連線讀寫，Register Map 如下：

Modbus 功能碼	對應	位址範圍	Gateway 資源
FC02 Discrete Inputs	讀取	0-7	DI (I1-I8 狀態)
FC01/05 Coils	讀寫	0-3	DO (DO1-DO4 控制)
FC04 Input Registers	讀取	0-7	AI (類比輸入值，0-65535)
FC03/06 Holding Registers	讀寫	0-1023	自訂變數區

> 在設定頁可以調整 Modbus TCP Server 的 Slave ID（預設 = 1）。

情境 C：Gateway 作為 Modbus TCP Client（讀取外部 PLC）

這部分的設定整合在「TCP 通道」中，因為它與 MQTT 一樣都是走乙太網路通訊。

操作步驟：

1. 進入「配置」頁 → 右側找到「TCP 通道」卡片 → 點擊 + 新增。
2. 在彈出的視窗中，將「協議 (Protocol)」下拉選單從預設的 MQTT 改為 **Modbus TCP**。
3. 此時下方的欄位會動態改變，請設定目標 PLC 的 IP 和 Port。
4. 設定要讀取的暫存器位址 (Register) 與觸發條件 (大於、等於閾值等)。
5. 儲存後，此通道的值可在訊號群組中作為來源使用

情境 D：Gateway 寫入外部 PLC ➡ SOON

> 此功能目前規劃中。完成後可透過規則引擎在 Action 中選擇 Modbus TCP Write，主動將數值寫入外部 PLC 的暫存器。

10

10. 規則頁操作

規則頁面 (Rules)

規則引擎是 Gateway 實現「邊緣運算」與「自動控制」的核心。你可以透過圖形化介面，將多個輸入條件組合，觸發對應的輸出動作。

規則引擎的基本概念是：

```text
IF 訊號群組成立
THEN 執行動作群組
```

10.1 建立訊號群組

訊號群組用來描述「什麼情況算成立」。

操作步驟：

1. 進入「規則」頁。
2. 新增或編輯訊號群組。
3. 選擇來源，例如 I1、類比邏輯、RS485 數值、TCP IO。
4. 設定 AND / OR 關係。
5. 設定 debounce 或延遲，避免接點抖動。
6. 儲存。

範例：

名稱	條件
安全條件成立	`安全門 = 關閉` AND `急停 = 未按下`
壓力過高	`壓力感測器 > 6.5 bar`
MES 啟動命令	`tcpio.startCommand = START`

10.2 建立動作群組

動作群組用來描述「成立時要控制哪些輸出」。

操作步驟：

1. 新增或編輯動作群組。
2. 選擇輸出來源，例如 DO、擴充 DO、TCP IO、MQTT 發佈。
3. 設定每個輸出的值。
4. 若是脈衝輸出，設定 pulse 時間。
5. 儲存。

範例：

名稱	動作
啟動警報	DO1 開啟、DO4 蜂鳴器脈衝
正常燈號	DO1 開啟、DO1 關閉
發佈 MES 訊息	TCP IO Converter 發佈 JSON

10.3 建立規則 (端到端實戰教學)

> **實戰情境**：我們要將設定好的「I1 安全門」與「DO1 紅燈」綁定。當門打開時，亮紅燈。

步驟一：建立訊號群組 (定義 IF)

1. 進入「規則」頁，在左側「訊號群組」卡片點擊 +。
2. 群組名稱輸入：`門被打開`。
3. 點擊「新增條件」按鈕。
4. 在出現的下拉選單中，來源選擇 `I1 安全門`。
5. 條件選擇 `等於`，數值選擇 `LOW` (假設門開是 LOW)。
6. 點擊「儲存」。

步驟二：建立動作群組 (定義 THEN)

1. 在中間「動作群組」卡片點擊 +。
2. 群組名稱輸入：`亮起紅燈`。
3. 點擊「新增動作」按鈕。
4. 來源選擇 `DO1 紅燈`。
5. 動作選擇 `開啟 (ON)`。
6. 點擊「儲存」。

步驟三：將兩者綁定為一條規則

1. 在右側「規則引擎」卡片點擊 +。
2. 規則名稱輸入：`安全防護規則`。

3. **當 (IF)** 下拉選單，選擇剛建立的 `門被打開`。
4. 選擇觸發時機為 `成立時`。
5. **就 (THEN)** 下拉選單，選擇剛建立的 `亮起紅燈`。
6. 確認「啟用」開關有打開，點擊「儲存」。

步驟四：硬體驗證 7. 回到「總覽」頁，手動將接在 I1 的磁簧開關移開（模擬開門）。8. 你會立刻看到總覽頁的 DO1 燈號亮起，同時機器上的 L1 紅色指示燈亮起。恭喜你完成第一次端到端設定！

驗證規則時建議先使用低風險輸出，例如指示燈，再連接真正設備。

11

10.5 工作流程頁操作（白話流程 + 一鍵操作）

「規則」頁是工程師建立邏輯的地方；「**工作流程**」頁則是把這些規則鏈，以**白話卡片**呈現給 **現場操作員** —— 不必懂訊號 / 規則 / 動作，從畫面就能「執行 / 開關 / 監看」，全程**不需下 MQTT 命令**。

每張卡片把一條（或一串接在一起的）流程寫成一句話：「**當 X → 就 Y**」，並依其性質歸成四種角色，卡片右上角有對應徽章與**即時狀態燈**：

角色	卡片長相	怎麼操作	範例
觸發型（可執行）	「▶ 執行」按鈕（可填參數）	按一下＝送一次命令	磅秤「命令-免校正 / 需校正」
開關型（待命／發送中）	**on/off 開關** + 即時狀態	切 ON 開始、切 OFF 停止	磅秤「**發重量開關**」
武裝監控型（監控中）	啟用／停用開關 + 即時條件值	先啟用，之後由實體事件自動觸發	「重量>5kg 亮燈，但需先啟用」
純被動（自動）	只有狀態、無操作鈕	只觀察，由實體訊號自動觸發	實體開關接點

操作要點

- **執行一個觸發型流程**：在卡片上（必要時）填參數 → 按「▶ 執行」。送出後會短暫顯示「⟳ 送出中」。
- **開關一個持續流程（開關型）**：直接撥卡片上的 on/off 開關。
  - 切 **ON** → 徽章變「🟢 發送中」、卡片發亮，下游開始持續動作（如每 2 秒送一筆重量）。
  - 切 **OFF** → 徽章變「⚪ 待命」、停止。
  - 狀態由設備即時回報，**多人 / 多裝置看到的開關狀態一致**（不是只反映你按的那下）。
- **即時值**：武裝監控型卡片會顯示條件當前值與「達成 / 未達成」，方便現場判斷。

> **範例：磅秤發重量**「發重量開關」是一張開關型卡片。切 ON 即開始每 2 秒送 `scale/weight`；切 OFF 即停。這取代了舊版要記主題、用 MQTT 工具發 `1` / `0` 的做法。詳見《磅秤測試指引》第 3 節。

> **小提醒**：側邊欄切換頁面是即時的；但若剛升級韌體 / UI，請整頁重新整理（Cmd/Ctrl+R）一次，確保載入最新畫面。卡片資料偶爾顯示「載入中…」數秒屬正常（設備單執行緒輪詢中），稍候即顯示。

12

11. 設備頁與 RS485 操作

設備頁面 (Devices)

設備頁用於管理 RS485 / Modbus RTU 設備。RS485 設備分為兩種用途：

用途	方向	範例
讀取設備	Gateway ← 設備	電表（電壓/電流/功率）、溫濕度感測器
寫入設備	Gateway → 設備	光警器（三色燈控制）、繼電器模組、馬達控制器

> 一個 RS485 設備可以同時有讀取和寫入暫存器。例如光警器可以讀取「目前狀態」，也可以寫入「切換燈色」。

11.1 RS485 接線原則

- 使用菊花鏈接線，不建議星狀接線。
- A 接 A，B 接 B，GND 共地。
- 每台設備使用唯一 Slave ID。
- 匯流排長度較長時，於最遠端加 120Ω 終端電阻。
- 所有設備需使用相同 Baud Rate、Parity、Stop Bits。

> ⚠ **同型號多台（如兩台 SF965）**：出廠常為同一 Slave ID → 匯流排實體撞包 +「第一台勝出」遮蔽第二台，兩台全掛。務必**先各設不同 Slave ID**（一次只接一台上線，用 Modbus Probe 寫位址暫存器或機身按鍵改址，再接下一台），且都 **≤ 32**（掃描上限）。v6.0.2 起 register↔device 綁定以 Slave ID 持久化，設好不同號後跨重開機不會串台。

建議通訊參數：

參數	建議值
Baud Rate	38400 或依設備規格
Parity	依設備規格，Finder 6M 常見為 8E1
Protocol	Modbus RTU

11.2 新增 RS485 設備（讀取）

操作步驟：

1. 進入「設備」頁。
2. 開啟 RS485 匯流排工具。
3. 設定全局通訊埠參數。
4. 使用掃描工具尋找設備，或手動新增。
5. 輸入設備名稱、Slave ID、設備類型與輪詢間隔。
6. 如果是已知設備（如 Finder 6M 電表），選擇設備模版自動帶入暫存器。
7. 如果是未知設備，選擇「自訂 Modbus 設備」，手動新增讀取暫存器。
8. 儲存後確認設備狀態為在線。

讀取到的數值會出現在：

- **總覽頁** 的 RS485 區塊（即時顯示）
- **Converter 樣板** 中可用 `${rs485.SlaveID.欄位名}` 引用
- **訊號群組** 中可作為條件來源

11.2a 自訂 Modbus 設備完整教學

當你遇到一台**不在已知模版清單中的 RS485 設備**，需要手動設定暫存器。以下是完整的流程指引。

步驟一：從設備說明書收集通訊參數

打開設備隨附的說明書（或從廠商官網下載），找到以下資訊：

你需要找到的資訊	在說明書中通常叫做	範例
Slave ID（站號）	通訊地址 / Station Address	1 ~ 247
Baud Rate（波特率）	通訊速率 / Baudrate	9600、19200、38400
Parity（校驗位）	校驗方式 / Parity Check	None (8N1) / Even (8E1)
暫存器位址表	Register Map / 暫存器對照表	0x0000 = 電壓, 0x0001 = 電流
Function Code（功能碼）	FC / 讀取方式	FC03 = Holding, FC04 = Input

> 💡 **小技巧**：如果設備說明書只有紙本，可以搜尋設備型號 + "Modbus" + "通訊協議" 來找 PDF 版。

步驟二：設定通訊埠參數

1. 進入「設備」頁 → 點擊「通訊埠設定」
2. 將 Port 1 的 **Baud Rate** 和 **Parity** 設為與設備一致
3. ⚠️ 同一條匯流排上的所有設備必須使用相同參數

步驟三：手動新增暫存器

1. 新增設備 → 類型選「未知 / 自訂 Modbus 設備」
2. 填入 Slave ID（與設備 DIP 開關一致）
3. 在「自訂暫存器」區域，逐一新增：

欄位	說明
✅ 啟用	勾選 = 在總覽頁顯示此暫存器
名稱	自訂名稱，如「溫度」「計數值」
📖 Read / ✏️ Write	Read = 從設備讀取 (FC03)；Write = 寫入控制 (FC06)
Addr	暫存器位址，直接填數字或 `0x` 開頭十六進位
資料類型	INT16 / UINT16 / INT32 / FLOAT 依設備規格選擇
× / ÷ 倍率	原始值的換算，如「÷ 10」表示原始值 250 → 顯示 25.0
Offset	偏移量，如 `-40` 用於某些溫度感測器

步驟四：使用 Modbus Probe 驗證

新增暫存器後，建議用「設備」頁底部的 **Modbus Probe** 工具做即時測試：

1. 填入 Slave ID、起始暫存器、讀取數量
2. 點「📡 發送 Probe」
3. 確認回應有數值（而非 timeout）
4. 若成功，表示通訊參數正確，可以儲存設備設定

倍率 (Scale) 與偏移 (Offset) 的常見用法

設備情境	原始值	設定	顯示值
溫度感測器回傳 250 代表 25.0°C	250	÷ 10	25.0
計數器 32-bit 編碼，後 8 位為小數	6400	÷ 256	25.0
原始值需加偏移	200	× 1, Offset = -40	160
原始值為百分比 (0-1000 → 0-100%)	500	÷ 10	50.0

實際範例：YX523R 光警器

YX523R 是一款 RS485 光警器，同時有「讀取」和「寫入」暫存器：

新增設備 - YX523R 模版

快速套用：在設備編輯畫面，從「樣板匯入」下拉選擇「YX523R 光警器 (含寫入)」，系統自動帶入：

- **設備名稱**：`YX523R 光警器`
- **Slave ID**：`4`（QA 治具預設；如有不同實機請手動改）
- **Baud / Parity**：使用 Bus 0 設定（建議 9600 / 8N1）

名稱	位址	模式	資料類型	說明
狀態	0x10	📖 Read	UINT16	讀取目前狀態 (FC03)
燈色	0x01	✏️ Write	UINT16	1=紅 2=綠 3=藍 4=黃 5=白 6=紫 7=青
模式	0x02	✏️ Write	UINT16	0=常亮 1=閃爍 2=漸變

> 套用模版後，你仍可以修改或新增暫存器（例如再加 0x03 蜂鳴器）。

11.2b 手動寫入操作（Dashboard）

設定好寫入暫存器的設備後，在總覽頁 (Dashboard) 的設備卡片中，寫入暫存器會顯示在獨立的「✏️ 寫入暫存器」區塊，每個暫存器旁邊有數值輸入框和「▶ 寫入」按鈕。

Dashboard 寫入操作

操作方式：

1. 在數值框輸入要寫入的值
2. 點擊「▶ 寫入」按鈕
3. 成功時會出現綠色 Toast：`✅ 寫入成功: Slave X Reg 0xYYYY = Z`
4. 失敗時會出現紅色 Toast：`❌ 寫入失敗: Slave X 無回應`

> 💡 手動寫入適合用在：調試新設備、臨時控制、出廠測試。正式生產建議透過「規則引擎」自動觸發寫入。

API 觸發（curl 範例）

除了 UI 按鈕，韌體還提供 `POST /api/rs485/:deviceIdx/write` 端點，可從外部腳本一次性寫入：

```bash
curl -X POST http://<gateway-ip>/api/rs485/0/write \
  -H "Authorization: Bearer admin-token" \
  -H "Content-Type: application/json" \
  -d '{ "registerIdx": 1, "value": 2 }'
# → { "ok": true, "wroteValue": 2, "elapsedMs": 38 }
```

- `:deviceIdx` 為 `cfg.modbusDevices[]` 索引（0、1、…）
- `registerIdx` 為該設備自己的 write registers 列表索引（從 0 開始，依設備卡顯示順序）
- 失敗時回 `{ "ok": false, "error": "rs485_timeout", "message": "RS485 bus 無回應" }`

11.3 新增 RS485 寫入暫存器

當你需要透過 RS485 **控制**設備（而不只是讀取），需要設定寫入暫存器。

場景：工廠有一台 YX523R 光警器，接在 RS485 匯流排上，Slave ID = 2。你想透過規則引擎控制它的燈色和閃爍模式。

操作步驟：

1. 進入「設備」頁 → 新增或編輯 RS485 設備
2. 在「自訂暫存器」區塊，新增暫存器：

名稱	暫存器位址	模式	說明
燈色	0x01	**Write**	1=紅 2=綠 3=藍 4=黃 5=白 6=紫 7=青
模式	0x02	**Write**	0=常亮 1=閃爍 2=漸變
狀態	0x10	Read	讀取目前狀態

3. 選擇模式為 **Write** 時，系統會：
   - 自動將 Function Code 設為 **FC06**（Write Single Register）
   - 不會在背景輪詢中讀取此暫存器（避免衝突）
   - 此暫存器會出現在規則 Action 的輸出選單中
4. 儲存設備設定。

在輸出群組中使用 RS485 寫入（WI-118 端到端範例）：

1. 進入「規則」頁 → 編輯**輸出群組** (Action)
2. 點「+ 新增輸出」
3. 第一個下拉「類別」選 📡 **Modbus 寫入 (RS485)**
4. 第二個下拉「設備」選對應的暫存器，例如 `YX523R 光警器 → 燈色 (0x01)`
5. 右側橘色框輸入「寫入值」（例如 `2` = 綠燈）
6. 注意值欄旁的 hint：例如 `💡 1=紅 2=綠 3=藍 4=黃 5=白 6=紫 7=青`，會自動從 `DEVICE_TEMPLATES.yx523r.registerLabels` 讀取
7. 儲存。再到規則頁綁好觸發信號（例如「按鈕按下」）即可

```
規則範例（WI-118 預設端到端）：
  輸入群組「按鈕按下」   : I1 == 1
  輸出群組「亮綠燈」     : YX523R 燈色 ← 寫入 2
  規則：按鈕按下 → 亮綠燈

實機驗證：
  按下接到 I1 的按鈕 → YX523R 燈變綠 → toast「✅ 寫入成功」
```

💡 寫入值是 `0~65535` 的 uint16；rule 觸發時韌體用 `outputMode=1 + mappings[0]` 模式把固定值寫入 register。Edge-trigger + 5 秒 refresh 防止 bus 洪水。

⚠️ RS485 寫入會佔用匯流排，系統已內建匯流排鎖定機制，確保寫入不會和背景讀取輪詢衝突。但如果同時有大量寫入操作，可能會略微增加響應延遲。

11.4 TDA-08B 稱重／測力控制器完整教學

TDA-08B 是單通道 24-bit 高精度 Modbus-RTU 稱重控制器，支援負載傳感器（四線制）直接接入，透過 RS-485 輸出實時量測值、峰值、谷值及穩定狀態。

11.5.1 TDA-08B 接線

① 負載傳感器 → TDA-08B（四線制標準配色）

傳感器導線顏色	接 TDA-08B 端子	說明
紅 (Red)	**E+**	激勵正
黑 (Black)	**E-**	激勵負
綠 (Green)	**S+**	訊號正
白 (White)	**S-**	訊號負

⚠️ 上述顏色為一般標準配色，實際請以傳感器原廠說明書為準。

② TDA-08B → Arduino Opta RS-485 端子

TDA-08B 端子	Opta Bus 0 端子	說明
A (+)	**A+**	RS-485 資料正
B (-)	**B-**	RS-485 資料負

> 💡 若匯流排線長超過 50m，或同一匯流排上有多台設備，建議在匯流排兩端各接 120Ω 終端電阻。

③ TDA-08B 電源

端子	說明
DC+	DC 18–30V 電源正
DC-	電源負 / GND

11.5.2 TDA-08B 通訊參數設定（E-Group）

在 TDA-08B 面板操作：**長按** `MENU` **3 秒**進入主選單 → 找到 **E GROUP** → 短按 `MENU` 進入。

選單代碼	參數	建議設定	說明
E0 bdL	波特率	**9600**	出廠預設
E1 SJC	資料格式	**8N1**	出廠預設
E2 tFC	通訊協議	**0 (Modbus)**	出廠預設
E3 Adr	通訊地址	**1**	確認與 Gateway 設定一致

> 💡 設定完後短按 `ZER` 返回，設備自動儲存。

11.5.3 Gateway 設定（套用模版）

1. 進入「設備」頁 → 點擊「**＋ 新增 RS485 設備**」
2. 在「從模版套用」下拉選擇 **「TDA-08B 稱重控制器」**
3. 點擊「**套用模版**」，系統自動填入：
   - 設備名稱：`TDA-08B 稱重控制器`
   - Slave ID：`1`

- Baud Rate：`9600` / 格式：`8N1`
- 5 個讀取暫存器（見下表）

4. 確認 Slave ID 與設備面板 E3 設定一致後，按「**儲存**」

11.5.4 暫存器說明

套用模版後自動建立以下讀取暫存器：

名稱	地址	資料型別	說明
實時值	11 (0x000B)	INT32	第 1 通道當前量測值（單位由設備 A-Group 小數點設定決定）
穩定狀態	9 (0x0009)	UINT16	Bit 0 = 1 表示第 1 通道量測值穩定；0 表示動態
線上狀態	10 (0x000A)	UINT16	Bit 0 = 1 表示第 1 通道感測器在線；0 表示離線
峰值	43 (0x002B)	INT32	第 1 通道歷史最大值（寫 0 可清零）
谷值	75 (0x004B)	INT32	第 1 通道歷史最小值（寫 0 可清零）

> 💡 **實時值解讀**：TDA-08B 回傳 32-bit 有符號整數。若面板設定 A1 小數點 = 2，則顯示值 = 實時值 ÷ 100。Gateway 目前以整數儲存，請依現場標定結果設定倍率（× / ÷）。

11.5.5 規則整合範例

場景：當量測值超過 5000（kg）時，觸發警報 DO1

1. 「**規則**」頁 → 新增規則
2. 訊號群組 → 選擇「TDA-08B 稱重控制器」→「實時值」→ 條件：`> 5000`
3. 動作群組 → 選擇「DO1 (DO1)」→ 輸出高電平
4. 啟用規則 → 儲存

場景：穩定狀態 = 1 才執行下游動作

1. 訊號群組 → 選「穩定狀態」→ 條件：`= 1`（Bit 0 = 1）
2. 結合實時值條件，使用 AND 邏輯

11.5.6 故障排除

症狀	可能原因	處理
實時值一直顯示 `0`	設備未上電或傳感器未接好	確認 E+/E-/S+/S- 接線，面板顯示 ERR 代碼
線上狀態 = 0	Slave ID 不符或 RS-485 接線錯誤	確認 E3 Adr = Gateway 設定的 Slave ID；用 Modbus Probe 測試
值不穩定	濾波係數太低或接地噪音	在 TDA-08B 面板 Ab FIL 增大濾波係數（建議 6~20）
量測值偏移	未清零 / 標定失效	在面板按 ZER 清零，或透過 Modbus 向暫存器 11 寫入 0
ERR01 過載	傳感器超載	確認負載未超過傳感器額定量程
ERR02 感測器故障	接線錯誤或傳感器損壞	逐一確認 E+/E-/S+/S- 接線是否正確、牢固

11.5.7 校正引導精靈（v5.9.130+，含燈號提示）

TDA-08B 有砝碼校正是**缺一不可的 4 步驟**，Gateway 提供**引導式精靈 + 燈號提示**，避免操作員漏步驟：

步驟	動作	燈號
1	放**空籃**在秤上 → 推進「標零」	🔵 第一顆燈亮
2	籃內放**砝碼**（如 5kg）→ 推進「標增益」	🟡 第二顆燈亮
3	完成（要重來按「重置」）	🟢 / 全滅

精靈以 **MQTT-Chain 工作流**實作（純設定、無專屬韌體）：推進按鍵發到 `…/cmd/signal/N`，狀態經 MQTT loopback 強制順序（缺一不可）。

> 📖 **完整操作流程、MQTT 指令、設定建構步驟**見：`guide-calibration-wizard.md` 📖 **工作流引擎原理 + API schema**見：`../specs/spec-mqtt-chain-workflow.md`

11.5.8 用 MQTT 觸發任何 RS485 量測值的「定期回報」

任何在輪詢的 RS485 量測值（磅秤重量、Finder 用電量、SF965 計米數…）都能用**同一套通用機制**做「外部下命令 → 設備每 N 秒回報」：一個 **MQTT 閘訊號** + 一個 **CONVERTER 輸出通道**，發 `{prefix}/cmd/signal/N = 秒數` 即開始週期上報、送 `0` 停（payload 即間隔，WI-143）。**純設定、無專屬韌體。**

> 💡 **發佈時機共三種模式**：除了上述「payload 當間隔」（WI-143），另有**固定間隔**（`converterIntervalSec`，每 N 秒固定發）與**變更時才發**（`converterOnChange`，v5.9.263；值有變才發、沒變不重複）。三種對照詳見〈觸發式定期回報〉。

> 📖 通用機制與最小骨架、各裝置 recipe（磅秤 / Finder 電力 / SF965 計米器）見：`guide-triggered-periodic-report.md`

11.5 RS485 故障排除

症狀	可能原因	處理
掃描不到設備	Slave ID、Baud、接線錯誤	單台直連測試，確認通訊參數
多台同時離線	Slave ID 衝突或匯流排品質不良	逐台斷開排查
數值一直是 0	現場沒有負載或 register 對應錯誤	確認設備端量測狀態與 register map
間歇斷線	線長、雜訊、終端電阻問題	使用屏蔽雙絞線並加終端電阻
寫入無反應	暫存器位址錯誤或設備不支援 FC06	用 Modbus Probe 工具手動測試寫入
寫入後設備狀態不對	數值格式（十進位 vs 十六進位）或 Endianness	查閱設備通訊手冊，確認暫存器值定義

13

12. 設定頁操作

設定頁面 (Settings)

設定頁面包含網路連線、通訊伺服器與系統管理等進階參數。MQTT、Config Server、系統維護與重置。

12.1 網路模式與斷線備援 (Failover)

Gateway 擁有強大的雙網路備援能力，確保資料回報不中斷。可用模式如下：

模式	用途
自動偵測 (Auto)	**強烈建議使用**。系統會優先使用有線 Ethernet。若現場網路線被拔除或斷線，Gateway 會自動啟用 WiFi 連線。當網路線插回時，會自動切換回有線網路。
僅 Ethernet	固定使用有線網路，關閉 WiFi 模組以節省資源。
僅 WiFi	固定使用無線網路，適用於無法拉實體網路線的機台。

建議：

- 初次部署使用 Ethernet + DHCP。
- 正式上線後可改為固定 IP。
- 若需 WiFi 備援，先在 Ethernet 連線下儲存 WiFi SSID / 密碼。
- 需要掃描 WiFi 時，可切換到 WiFi 設定模式。

12.2 WiFi 設定模式

當設備進入 AP Mode：

1. 連接 SSID `MES-Gateway-Setup`。
2. 開啟 `http://192.168.3.1`。
3. 掃描或手動輸入 2.4GHz WiFi。
4. 儲存認證。
5. 依畫面提示重啟或返回一般模式。

12.3 MQTT 設定

> 💡 **v5.9.261 起出廠已預設官方 TLS broker + 帳密**（port 8883），客戶**免自行設定** broker / 帳號密碼即可上雲；只有要改用自家 broker 時才需動以下欄位。

設定欄位：

欄位	說明
啟用 MQTT	控制是否啟動 MQTT client
Broker	MQTT broker IP 或 host
Port	常見為 1883
Client ID	建議使用唯一設備 ID
Username / Password	依現場 broker 設定
定期 IO / RS485 推送	是否將現場資料定期送往上層系統

操作步驟：

1. 填入 Broker 與認證資料。
2. 先按「測試連線」。
3. 測試成功後再儲存。
4. 回到總覽頁確認 MQTT 狀態。

若 MQTT 連續失敗，系統可能自動停用 MQTT，以避免網路堆疊被重試佔滿。修正設定後重新儲存即可再啟用。

> 💡 **關於「測試連線」(v6.0.4)**：按下後會顯示「測試中」再回結果（非同步，TLS 握手需時）。若設備**已連到相同 broker**，測試會直接以現有連線判定通過——避免對同一 broker 開第二條 TLS（那在單執行緒設備會握手失敗，即舊版「TLS 測試一直錯、直接存檔卻可以」的根因）。**密碼欄留空 = 用已儲存密碼**；要測新密碼才填入。改了 broker/帳密才會真的重連測試。

12.4 Config Server

Config Server 用於多設備管理、備份、OTA 與授權。

使用建議：

1. 設定 Config Server URL。
2. 確認設備能回報 heartbeat。

3. 在 Config Server Dashboard 檢查設備是否在線。
4. 完成現場設定後，建立一份配置備份。

注意：裝置端 QSPI Flash 中的 `/fs/config.json` 是實際運行設定。Config Server 保存的是設備上傳的快照。若兩者不一致，現場運行以裝置端為準。

12.5 設備授權啟用 (License Activation)

當你拿到一台全新的 Gateway，或者剛為設備重新燒錄韌體時，設備可能會處於未授權狀態。在未授權狀態下，部分功能（如規則引擎儲存、雲端連線）可能會被限制。

如何啟用授權：

1. 進入「設定」頁面，查看頂部的 **「授權狀態 (License Status)」**。
2. 如果狀態顯示為 `ACTIVE` (綠色)，代表設備已經授權，無需任何操作。
3. 如果狀態顯示為 `PENDING` 或 `UNLICENSED` (黃色/紅色)：
    - 確認設備已經連上網際網路 (Ethernet 或 WiFi)。
    - 點擊旁邊的 **「啟動授權 (Activate)」** 按鈕。
    - 系統會向 Config Server 發起驗證請求，請等待約 3~5 秒。
    - 成功後，畫面上方會跳出綠色提示，狀態變更為 `ACTIVE`。

> **注意**：授權綁定於硬體的唯一識別碼 (MAC / CPU ID)。只要連上網啟動一次，授權會永久保存在本機的 QSPI Flash 中，之後即使斷網也可以正常運作所有工業控制功能。

14

12.6 遠端診斷：設備日誌與開機原因 (v6.0.1+)

設備在現場、只能經 VPN 進入、無法插 USB 時，不必到現場也能看 serial log 與重開原因。

- **即時日誌**：設定頁「📁 裝置識別」卡 →「📟 設備日誌」鈕，或直接開 `http://<設備IP>/log.html`。需 admin 權限（與設定頁同一組 token，預設 `smmsadmin`）。每 5 秒自動刷新，可看 RS485 輪詢、MQTT、開機序列；可下載/暫停/清畫面。敏感值（密碼/token）已自動遮罩。

- **為何重開**：日誌頁頂部橫幅顯示「為何重開 / 上次撐多久 / 開機次數 / 目前運行」。`SOFTWARE` = 有意重開（OTA/設定/使用者）；`UNEXPECTED` = 非預期（看門狗/斷電/當機）——**開機次數快速上升 + 上次撐很短 = 重開迴圈**，要查電源/網路/韌體。這些欄位也隨心跳上雲，設備隨後失聯仍可在雲端查上次開機原因。
- **限制**：此頁需要設備網路/網頁活著；若設備 reboot loop 且網路也死，網頁進不去 → 改看雲端上次 `bootReason` 或現場 USB。詳見〈疑難排解 → 📟 遠端診斷〉。

15

13. OTA 韌體更新

> **v5.9.155+ 架構更新**：為釋放 Flash 讓 binary 跌破 780KB OTA 上限，**`app.js`**、**`workflow.html`**、**CSS、三套語系字典 (zh-TW/en/zh-CN) 已從韌體 PROGMEM 搬到 QSPI** runtime 服務（v5.9.192 起 app.js 也外置）。因此 OTA 需同時更新韌體 + 這 6 個資產，**`.mesb` 單一檔格式已復活**（一檔內含 firmware + 全部 web 資產，一次投遞）。

兩種更新檔：

- **`.mesb` (建議)** — 單一檔內含 firmware + CSS/語系資產，OTA 一次帶齊，保證韌體與 前端資產同版本。由 `scripts/build_ota_bundle.py` 打包（見 13.6）。
- **`.bin` (純韌體)** — 只更新 firmware，不帶資產。適用於資產未變動、或走乙太網 raw OTA。（注意：純 `.bin` over **WiFi** 的 raw 路徑可靠性較弱，WiFi 設備建議走 `.mesb`。）
- `index.html`（載入器）仍嵌入韌體 PROGMEM；但 `app.js` 已外置 QSPI（v5.9.192+）→ **QSPI app.js 缺檔時 UI 會卡「載入中」**（index 的 bootstrap 會重試，但無 inline app fallback）。所以**升級一律走 `.mesb`**（帶齊資產），別只燒 raw `.bin`。
- v5.9.78+ 連續 OTA 已驗證穩定；`.mesb` over WiFi 端對端已驗證（v5.9.170/172）

13.1 線上 OTA（雲端 → 設備）

由 Config Server 發起，整段流程約 90 秒：

1. 管理員上傳 **`.mesb` (建議) 或 `.bin`** 韌體到 Config Server（Web UI「📦 韌體」頁拖放上傳）。雲端會依檔案 magic 自動辨識（前 4 bytes `MESB` → 命名 `.mesb`，否則 `.bin`）。

2. 在 Config Server 選擇目標設備按「🚀 部署」
3. 設備下次 heartbeat（30s 一次）取得 OTA 指令（含下載 URL），加入本地 pending 清單
4. 設備在「設定」頁「韌體更新 (OTA)」可看到「☁️ 伺服器韌體 v X.Y.Z 可用」
5. 用戶按「📥 選擇版本後更新」開始下載（admin 強制部署可跳過此步自動觸發）
6. 設備依 URL 副檔名分流寫入：
   - `.mesb` → `handleMesbUpload`：firmware 先串流收到 P4 暫存 → 關 WiFi → **強制 reformat P2** → copy 到 P2 `UPDATE.BIN`、web 資產寫入 P4 `/cfg/web/`（mbed File API）
   - `.bin` → `handleOtaUpload`：直接寫 P2 `UPDATE.BIN`（pre-unlink + 強制 sync）
7. 設置 RTC backup register 後**立即** `NVIC_SystemReset()` 重啟（不留 delay window）
8. Bootloader 讀 RTC magic → 將 UPDATE.BIN 從 QSPI 複製到內部 flash → boot 新韌體

13.2 離線 OTA（USB DFU）

當以下情況發生時改用 USB DFU：

- 韌體 binary > 780KB（v5.9.78 起 OTA guard 會回 413 自動拒絕）
- 設備離雲端且網路無法連接 Config Server
- OTA swap 失敗 fallback 到 DFU 模式（綠燈呼吸閃爍且無法重啟）

DFU 流程：

1. USB 線連接電腦
2. 連點兩下 OPTA 上的 RESET 按鈕 → 進入 DFU 模式（綠燈呼吸閃爍）
3. 在電腦上執行 `pio run -e opta -t upload`，或用 `dfu-util -d 2341:0364 -a 0 -D firmware.bin`
4. 燒錄約 30 秒，自動重啟到新韌體

13.3 OTA 安全機制（v5.9.78 累積）

自動 reject 條件（`POST /api/ota/trigger` 回 409 / 413）：

狀態	HTTP 回應	原因
同版號	409 Conflict	target == current，可能 heartbeat stale，等 30s 或重啟設備
Binary > 780KB	413 Payload Too Large	接近 partition 限制（786KB），改 USB DFU 更穩

v5.9.66 → v5.9.78 OTA brick 修補歷程

之前連續 OTA 偶爾會 brick 進 DFU。經過多輪除錯找到根因 + 修補：

問題	修補（從 v5.X 開始）
`localConfig.remount()` 在 OTA 寫完後 re-init QSPI device，擾亂 partition 2	移除 OTA-V2 path 的 remount（v5.9.66）
Reboot 前 3000ms delay 中其他 task 寫 QSPI 觸發 cache flush	RTC set 完立刻 NVIC_SystemReset()，不留 delay（v5.9.69）
Partition 2 殘留前次 UPDATE.BIN + FAT metadata 影響新寫入	寫前 remove() + unmount/remount 強制 sync（v5.9.72）
Background tasks（RS485 / MQTT / heartbeat）干擾 OTA write	loop() 偵測 `g_otaInProgress=true` 立即 return（v5.9.78）

連續 OTA 兩次 timing 對比已驗證一致（v5.9.81 → v5.9.82 連測）：

- Download → Content-Length: ~2.4s
- handleOtaUpload START → Pre-unlink: ~240ms（FAT mount）
- 整段下載 + reboot：約 90 秒

13.4 OTA 更新燈號與安全機制

當設備透過線上或離線成功下載新韌體後，會進入重啟並進入底層 Bootloader 模式來進行韌體的替換。這個階段的特徵如下：

- **燈號狀態**：設備面板上的 **綠燈會呈現緩慢的呼吸閃爍 (DFU 模式)**

- **處理時間**：Bootloader 將 UPDATE.BIN 從 QSPI partition 2 搬移到內部 flash。約需 **30~60 秒**（不像舊版要 2~3 分鐘那麼長）
- **安全還原 (Rollback) 機制**：Opta 內建了防磚防護。如果在 DFU 過程中斷電，或是新版本韌體存在嚴重 Bug 導致無法正常開機，可以**重新 USB DFU 燒入舊版**還原
- ⚠️ **嚴格禁忌**：在綠燈呼吸閃爍期間，**絕對禁止拔除電源**

13.5 OTA 診斷（v5.9.78+）

OTA 各階段都有 `[OTA t=Nms]` 時間戳前綴方便診斷。可從 USB serial（115200 baud）即時觀察：

```
[OTA-V2 t=78896ms] Downloading from: https://...
[OTA-V2 t=81330ms] Got Content-Length: 747 KB
[OTA t=81434ms] === handleOtaUpload START ===
[OTA t=81441ms] Mounting MBR partition 2 (OTA)
[OTA t=81677ms] Pre-unlink UPDATE.BIN: OK (removed)
[OTA] Progress: 10%~90%
[OTA t=NNNms] Upload OK; total 765744 bytes (XXXXX ms)
[OTA t=NNNms] RTC: DR0=0x07AA DR1=0xA4 DR2=2 DR3=765744
[OTA t=NNNms] Rebooting immediately via NVIC_SystemReset()...
```

若懷疑 OTA brick 規律：

1. `pio device monitor --port /dev/cu.usbmodem* --baud 115200` 接 serial
2. 觸發 OTA #1 + 記錄時間戳序列
3. 連續 OTA #2 + 比對時間差異
4. 任一階段卡住超過 5 秒都應該回報

`.mesb` OTA 的 serial 特徵（與 `.bin` 不同）：

```
[OTA t=Nms] === handleMesbUpload START (.mesb v2) ===
[OTA t=Nms] bundle: fwSize=726996 assets=6
[OTA t=Nms] FW received → P4 temp + CRC OK
[OTA t=Nms] asset OK: app.js.gz / workflow.html.gz / styles.css.gz / i18n-*.json
[OTA t=Nms] all received; shutting down WiFi before P2 copy
[OTA t=Nms] WiFi off; copy FW P4→P2 (alternating mount)
[OTA t=Nms] FW copied → P2 + CRC OK → arming bootloader & rebooting
```

13.6 .mesb 打包與資產投遞

> 📌 發佈端完整 SOP(一般版本 `.mesb` vs 拓荒包 `bootstrap.bin` 兩條程序分工)見 `guide-ota-embedded-web.md`;本節為手冊內的快速編排。

`.mesb` v2 是單一檔容器(32B header `MESB` + firmware + 每個 web 資產的 manifest),一次帶齊 firmware 與 QSPI 資產。**沒有單一腳本一鍵完成,是多步驟手動編排**,順序如下。

> ⚠️ **重點(v5.9.192+):必須打包「全部 6 個 QSPI 資產」**。WI-129 之後 `app.js` 與 `workflow.html` 也外置到 QSPI(不再 inline 在韌體)。**漏包 app.js/workflow.html → OTA 帶不過 UI 改動**(設備留舊資產)。早期文件只列 4 個(CSS+三語系)是 WI-129 之前的舊狀態,**勿照舊抄**。

```bash
# ── 步驟 0：先產生 web 資產的 .gz（必做，否則 web/build/ 是舊的）──
python3 scripts/build_webpage.py      # → app.js.gz / styles.css.gz / i18n-*.jso
( cd web && node build-workflow.js )  # → workflow.html.gz

# ── 步驟 1：編譯韌體 ──
pio run -e opta                                          # 授權版(production)
# 免授權版(暫時，授權穩定後將取消)：前置 build flag
# PLATFORMIO_BUILD_FLAGS='-D BYPASS_LICENSE_CHECK' pio run -e opta

# ── 步驟 2：打包 .mesb（fw + 全部 6 個 QSPI 資產）──
python3 scripts/build_ota_bundle.py \
  --fw .pio/build/opta/firmware.bin \
  --asset web/build/app.js.gz \
  --asset web/build/workflow.html.gz \
  --asset web/build/styles.css.gz \
  --asset web/build/i18n-en.json.gz \
  --asset web/build/i18n-zh-TW.json.gz \
  --asset web/build/i18n-zh-CN.json.gz \
  --out release/mes-gateway-v<版本>.mesb

# ── 步驟 3：驗證（務必看到 assets=6 + 各 CRC OK）──
python3 scripts/build_ota_bundle.py --verify release/mes-gateway-v<版本>.mesb

# ── 步驟 4a：本機直接投遞到設備（admin token，會重開機 ~90s）──
curl -X POST http://<device-ip>/api/ota/bundle \
  -H "Authorization: Bearer <admin-token>" \
  -H "Content-Type: application/octet-stream" \
  --data-binary @release/mes-gateway-v<版本>.mesb

# ── 步驟 4b：或上傳雲端 Config Server，再「部署」（設備主動 pull）──
#   ⚠️ 用 https:// 直連（http 會 301→POST 降級 GET→404）；channel=dev 是安全預設(不自動
CL=$(printf '%s' "v<版本> 變更說明…" | base64 | tr -d '\n')
curl -X POST 'https://opta.smms.com.tw/api/firmware/upload' \
```

```
  -H 'Content-Type: application/octet-stream' \
  -H 'X-Firmware-Version: <版本>' \
  -H "X-Firmware-Changelog-B64: $CL" \
  -H 'X-Firmware-Channel: dev' \
  -H 'Authorization: Bearer admin-token' \
  --data-binary @release/mes-gateway-v<版本>.mesb
#   免授權版多帶：  -H 'X-Firmware-No-License: 1' -H 'X-Firmware-Variant: nolicense'
```

設備收到後：firmware → QSPI P2 `UPDATE.BIN`（reboot 後 bootloader 搬入內部 flash）、web 資產 → QSPI P4 `/cfg/web/`（runtime 由 `serveFile` 服務，ETag 304 快取）。

> **離線/沒網路的現場**：用步驟 4a（本機 `POST /api/ota/bundle`），同網段直接投遞，免雲端。完全無網路時走 §13.2 USB DFU（但 USB 只燒 firmware、**不帶 QSPI 資產** → UI 不會更新；資產需另用 `POST /api/web?name=<檔>` 逐一補上，或改用 .mesb 路徑）。

> **雙變體（過渡期）**：目前同時發「授權版 / 免授權版」。**免授權版是暫時措施**（因授權機制尚未穩定），授權機制穩定後將停止產出免授權版、並把雲端已公開的 `-nolicense.mesb` 撤下。

注意：OTA 寫 QSPI 仍有 soft-brick 風險（可 USB DFU 救回）；正式批量部署前務必先單機驗證。

16

14. 備份、還原與工廠重置

14.1 備份

建議在以下時機備份：

- 初次上線完成後。
- 修改規則或通訊設定前。
- OTA 更新前。

- 更換設備前。

可備份的內容：

- 裝置端配置。
- Config Server 的 `config-server/data/`。
- 韌體版本與 manifest。

14.2 還原

還原方式：

1. 從 Config Server 選擇指定設備備份。
2. 還原到同一台或替換設備。
3. 設備重啟後檢查配置是否正確。
4. 驗證網路、I/O、規則、RS485、MQTT。

14.3 工廠重置

工廠重置會清除自訂配置與儲存資料，設備將重新啟動。

執行前請確認：

- 已匯出或備份目前設定。
- 現場人員知道設備會暫時離線。
- 若依賴 WiFi，重置後可能需要重新進入 AP Mode 設定。

17

15. 日常巡檢表

建議每班或每日巡檢一次：

項目	判定
Gateway 在線	總覽頁可開啟
網路狀態正常	Ethernet / WiFi 顯示符合現場設定
MQTT 正常	總覽頁顯示已連線或已依需求停用
I/O 狀態正常	重要通道名稱與狀態正確
RS485 設備在線	設備頁或總覽頁顯示連線
規則有效	重要規則啟用且可觸發
統計正常累計	生產計數沒有異常歸零
Config Server 同步	Dashboard 顯示設備在線與最新 heartbeat

18

16. 常見問題

無法開啟 Web Dashboard

處理順序：

1. 確認電源與網路線。
2. 從 DHCP Server 查詢 IP。
3. 嘗試 ping Gateway IP。
4. 若使用 WiFi，確認設備是否進入 AP Mode。
5. 若仍無法連線，使用 USER 按鈕進入安全設定模式。

設備改 IP 後找不到

可能原因是裝置端 QSPI 已保存舊設定，重新燒錄韌體不會覆蓋已保存的 IP。

處理方式：

1. 使用舊 IP 進入 Web Dashboard。

2. 在設定頁修改網路。
3. 若舊 IP 無法進入，使用 AP Mode 或工廠重置。

MQTT 顯示連線失敗

檢查：

- Broker IP / Host 是否正確。
- Port 是否開放。
- Username / Password 是否正確。
- Client ID 是否重複。
- 網路是否能連到 Broker。

MQTT 顯示已連線，但命令／規則沒反應

症狀：Web 正常、MQTT 顯示已連線、設備也能 publish（遙測還在送），但**設備收不到任何 MQTT 命令**（wizard 推進按鍵、`cmd/signal/N` 都沒反應）。

原因：broker 重啟或網路抖動後，連線雖恢復但訂閱（subscribe）失效——這是已知 MQTT lib 行為，設備自我偵測不可靠。

處理：

- 立即恢復：`POST /api/system/reboot`（或斷電重開），重開後訂閱即恢復。
- 長期自動化：部署外部 **MQTT Watchdog**（`scripts/opta_watchdog.py`），偵測到連續探測失敗會自動觸發 reboot。詳見 `guide-failover-resilience.md` §4。

RS485 設備不穩定

檢查：

- Slave ID 是否唯一。
- 所有設備 Baud / Parity 是否一致。
- A/B 是否接反。
- 線材是否過長或受干擾。
- 是否需要終端電阻。

OTA 後畫面或功能異常

處理順序：

1. 強制重新整理瀏覽器。
2. 確認設備版本。
3. 檢查 Config Server 是否部署正確 `.bin`。
4. 若設備無法啟動，進入 DFU / 維修流程重新燒錄。

19

17. 文件來源

本手冊依下列文件整理：

- `docs/references/ref-document-authority.md`
- `docs/README.md`
- `docs/roadmap.md`
- `docs/api/api-firmware.md`
- `docs/api/api-config-server.md`
- `docs/specs/spec-config-system.md`
- `docs/specs/spec-connectivity.md`
- `docs/specs/spec-tcp-io-channels.md`
- `docs/specs/spec-ota-system.md`
- `docs/guides/guide-first-deployment.md`
- `docs/guides/guide-calibration-wizard.md` — TDA08B 校正引導精靈操作手冊
- `docs/guides/guide-failover-resilience.md` — 備援機制 + MQTT Watchdog 部署
- `docs/specs/spec-mqtt-chain-workflow.md` — MQTT-Chain 工作流引擎 + API
- `docs/hardware/guide-installation.md`
- `docs/hardware/rs485/guide-setup-sop.md`
- `docs/sprints/sprint-23-plan.md`
- `docs/sprints/sprint-24-plan.md`

若後續文件或程式碼顯示功能狀態已更新，請同步修訂本手冊。

01 使用者

秤重校正精靈

📅 最後更新：2026-07-28 | 🛠️ 對應韌體：v5.9.130+ | 📌 負責人：KC

本手冊說明 TDA08B 秤重模組的**引導式校正流程**（4 步驟 + 燈號提示），以及背後的 MQTT 命令、設定建構方式。校正以 MQTT-Chain 工作流引擎 實作，**全部設定、無專屬韌體**。

01

1. 為什麼是「引導精靈」而不是單一命令

TDA08B 的有砝碼校正（依原廠手冊 §5.4.1）是**缺一不可的 4 步驟**：

步驟	暫存器寫入	使用者動作	燈號提示
1 選通道	reg107 = 1	（自動）	—
2 標零點（tare）	reg113 = 1	**放空籃**上秤	第一顆燈亮（進行中）
3 寫砝碼值	reg108 = 砝碼值	**籃內放砝碼**	第一顆燈持續亮
4 確認增益	reg113 = 2	（自動）	第二顆燈亮（完成）

⚠️ **燈號沒有固定顏色。**韌體只控制「第幾顆燈亮/滅」，顏色完全取決於現場把哪一路 DO 接到哪一顆燈泡。舊版本文件在這裡標了藍/黃/綠三種顏色，那是**錯的** —— 實際只有兩顆燈的 亮滅，而且藍色容易被誤認成 Opta 板載的藍色 LED（完全不同的東西，見 §7）。接線與設定方式見 **§7 號誌燈搭配**。

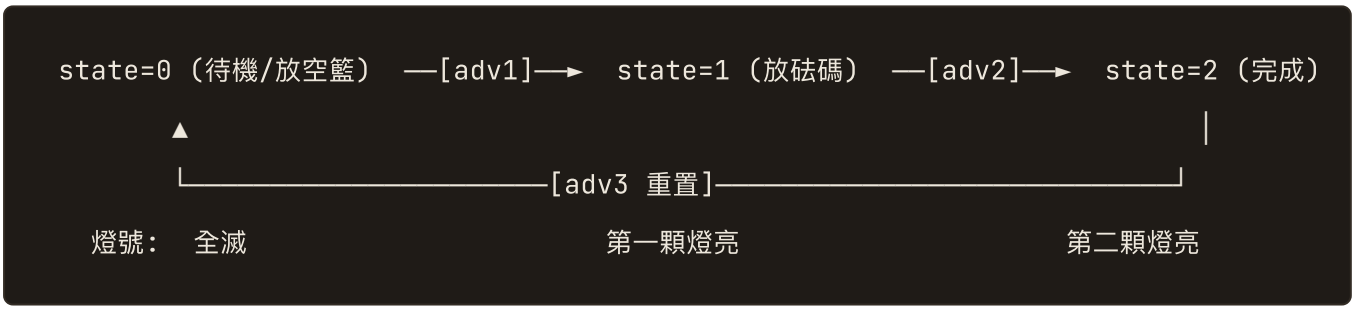
> **籃子（basket-as-tare）流程**：放空籃標零點（扣掉籃重）→ 籃內放砝碼標增益 → 之後秤籃裡的東西就是淨重。**砝碼值換算**：TDA08B A1 dot=2（小數兩位），5kg → `reg108 = 500`（顯示值 ×100）。

每一步需要使用者在環（放空籃/放砝碼）+ 燈號回饋，否則操作員不知道現在該做什麼。這是**有狀態的精靈**，不是 fire-and-forget 命令。

02

2. 狀態機設計

以 MQTT topic `…/cmd/signal/0` 當「目前步驟狀態」（state），由 gateway 發佈；操作員以「推進按鍵」逐步前進。



推進按鍵	MQTT topic	條件（AND）	動作
adv1 標零	`…/cmd/signal/1`	state==0 ∧ adv1≥0.5	reg107=1, reg113=1, state→1, 第一顆燈亮
adv2 標增益	`…/cmd/signal/2`	state==1 ∧ adv2≥0.5	reg107=1, reg108=500, reg113=2, state→2, 第二顆燈亮
adv3 重置	`…/cmd/signal/3`	state==2 ∧ adv3≥0.5	state→0, 全滅

順序強制：每步用 2-source AND（`state==N` ∧ `advN`），只有在前一步 state 正確時才會觸發 → 缺一不可。

03

3. 操作流程（現場操作員）

> 前提：磅秤已接 TDA08B（slave 3, bus0, 9600/8N1），MQTT 已連線。推進按鍵以**瞬時**方式送（發 `1` 後立即發 `0`，模擬實體按鈕）。

用 MQTTX / mosquitto_pub 操作

```bash
PRE="mes/gateway/<UID>"   # UID 見 /api/poll 或設備標籤

# 步驟 1：放空籃在秤上 → 推進標零
mosquitto_pub -h <broker> -p 1883 -u smms -P <pw> -t "$PRE/cmd/signal/1" -m 1
mosquitto_pub -h <broker> -p 1883 -u smms -P <pw> -t "$PRE/cmd/signal/1" -m 0
#  → 第一顆燈亮，state=1（觀察 $PRE/cmd/signal/0 = 1）

# 步驟 2：籃內放 5kg 砝碼 → 推進標增益
mosquitto_pub ... -t "$PRE/cmd/signal/2" -m 1
mosquitto_pub ... -t "$PRE/cmd/signal/2" -m 0
#  → 第二顆燈亮，state=2，weight 應校到 5.00

# 完成。要重新校正：
mosquitto_pub ... -t "$PRE/cmd/signal/3" -m 1
mosquitto_pub ... -t "$PRE/cmd/signal/3" -m 0
#  → 全滅，state=0，回到待機
```

觀察校正結果

```bash
# 訂閱即時重量（若有設 Converter 遙測通道）
mosquitto_sub -h <broker> -p 1883 -u smms -P <pw> -t "$PRE/scale/weight" -v
# {"weight":5.00,"stable":1,...}  ← 校正成功後放 5kg 砝碼應顯示 5.00
```

> 七色燈可用三色燈或任意兩路 DO 模擬：實際只有「整顆亮/滅」，用第一顆＝進行中、第二顆＝完成。**哪一路 DO 對應哪顆燈由現場接線決定**，索引寫法與實測步驟見 §7.3。

04

4. 燈號對照

state	意義	第一顆燈	第二顆燈
0	待機 / 放空籃	滅	滅
1	放砝碼	**亮**	滅
2	完成	滅	**亮**

05

5. 設定建構（工程/部署）

整套 wizard 以 API 建構（compact JSON，序列化送）。完整 schema 見 spec-mqtt-chain-workflow.md。建構順序：

1. **寫暫存器**（`POST /api/config` modbusRegisters，連同既有讀暫存器一起送）：

- reg107（cal_ch）、reg108（cal_wt）、reg113（cal_trig），全部 `mode=1`（Write）、`functionCode=6`。

2. **TCP IO 狀態通道**（`POST /api/tcpio`，CONVERTER/OUTPUT）：
   - `wz-st1` payload `1`、`wz-st2` payload `2`、`wz-st0` payload `0`，topic 都指 `…/cmd/signal/0`。
3. **Signals**（`POST /api/signals`，2-source AND，trigger=CHANGE）：校正標零 / 標增益 / 重置。
4. **Actions**（`POST /api/actions`）：標零 / 標增益 / 重置，含 CH_MODBUS 寫 + CH_TCP 發 state + CH_DO 燈號。
5. **Rules**（`POST /api/rules`）：3 條 signal→action。

> ⚠️ 每個 POST 後 GET 驗證（特別是 signal 的 `sourceCount` 要等於 2）。JSON 必須 compact、Content-Length 用 byte 數。詳見規格文件「已知雷區」。

06

6. 驗證紀錄（2026-05-26, 72.77）

機制層已端到端驗證 PASS（磅秤暫拆，TDA08B 控制器在線）：

```
adv1 → rules/2/trigger 校正標零 → cmd/signal/0=1 → 第一顆燈 ON
adv2 → rules/3/trigger 校正標增益 → cmd/signal/0=2 → 第二顆燈 ON
adv3 → rules/4/trigger 校正重置 → cmd/signal/0=0 → 全滅
```

底層每環節確認：MQTT 輸入、2-source AND 順序強制、CHANGE 邊緣、rule fire、CH_MODBUS 寫命令、CH_DO 燈號切換、CH_TCP state loopback。

待驗：接回磅秤後跑完整校正，確認 reg108 增益生效、weight 校到 5.00（需實體 load cell）。

07

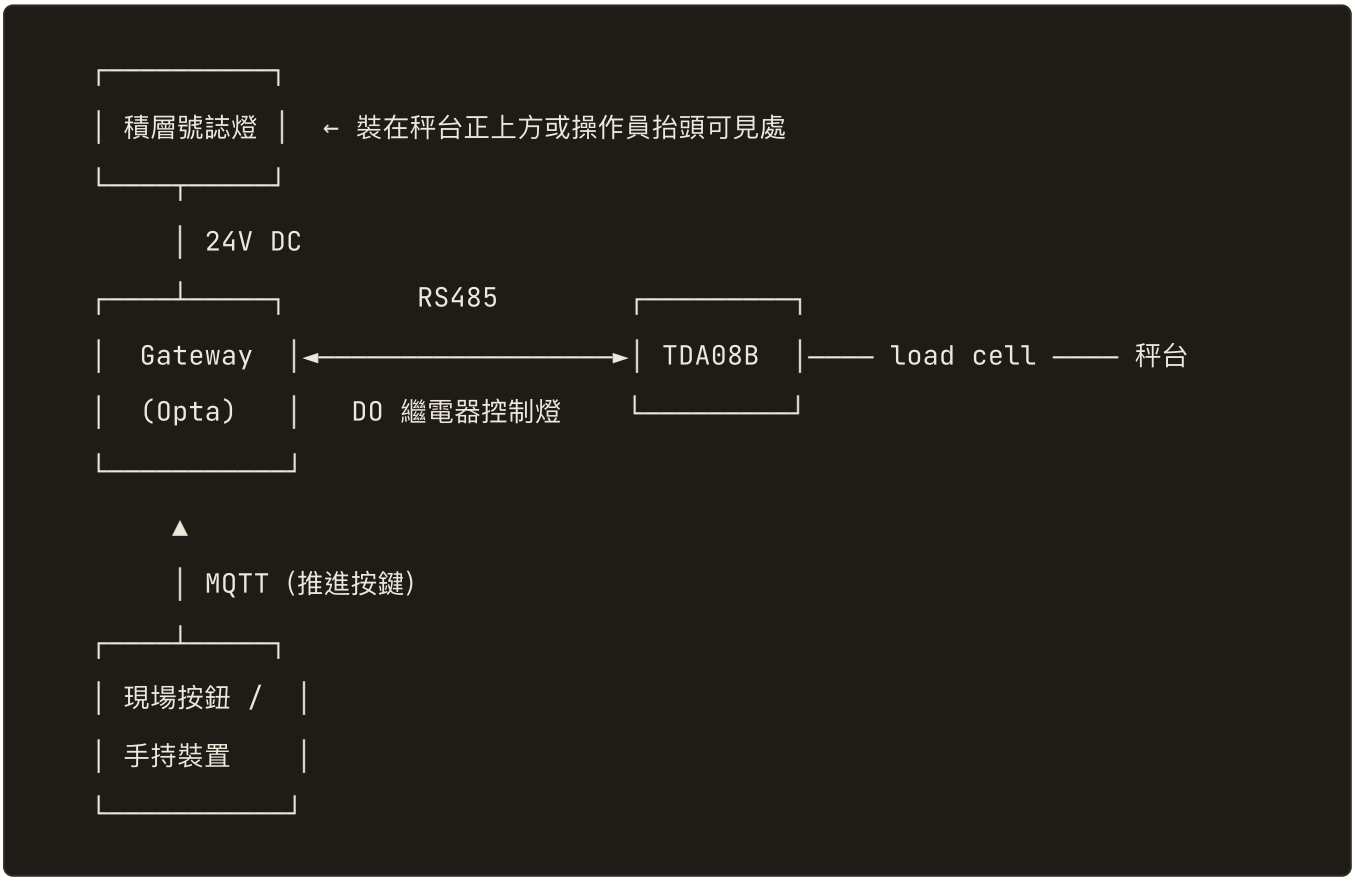
7. 號誌燈搭配（三色燈 / 積層燈）

> **磅秤和號誌燈是兩件各自獨立的設備**，彼此不知道對方存在，也不一定裝在一起。把它們串起來的是 Gateway 的 Rule Engine：校正精靈推進到某個狀態時，觸發一個 Action 去切換 DO 繼電器，繼電器再點亮外接的燈。**號誌燈不是磅秤的配件，也不是必要的** —— 沒有燈一樣能完成校正，只是操作員少了現場提示。

7.1 應用情境：為什麼要用燈，而不是看螢幕

校正是**雙手都被佔用**的工作：一手扶籃、一手放砝碼，人站在秤台前，眼睛盯著秤盤。這種情況下「回頭去看電腦螢幕確認現在第幾步」是不切實際的。

典型現場長這樣：



操作員視角的完整流程：

燈況	操作員知道要做什麼	做完按推進鍵
全滅	待機。把**空籃**放上秤台	按「推進」→ 標零
第一顆燈亮	零點已抓。在**籃內放砝碼**	按「推進」→ 標增益
第二顆燈亮	**校正完成**，可以開始秤料	（不用按）
全滅（從第二顆燈熄滅）	已重置，要重新校正	回到第一步

重點是**操作員完全不需要知道 reg107/reg113 是什麼**，也不需要看螢幕 —— 抬頭看燈就知道現在在哪一步、下一步該做什麼。這就是把它做成「引導精靈 + 燈號」而不是一道命令的原因。

常見場所：食品／五金／電子廠的分裝秤重站、產線旁的計重工位、需要定期複校的計量點。這些地方共通點是：環境吵（聽不到語音提示）、戴手套（不方便操作螢幕）、Gateway 通常裝在 配電箱裡（現場看不到）。

> 💡 **不裝燈也可以**：若操作員本來就在電腦前作業（例如實驗室環境），直接看 MQTT 的 `…/cmd/signal/0` 狀態值或網頁即可，本章可整章略過。

7.2 先釐清：這裡講的燈不是 Opta 板載 LED

	外接號誌燈（本章）	Opta 板載 LED
位置	機台上的積層燈 / 三色燈塔	Opta 本體面板
由誰控制	Rule Engine → DO 繼電器	韌體內部（心跳、開機狀態）
顏色	**由現場接線決定**	硬體固定
跟校正的關係	本章要設定的東西	**無關**

⚠️ 板載那顆藍燈（LEDB / PE_5）**只有 8320（WiFi 版）能亮**；8310 沒有 WiFi 模組，藍燈硬體 不存在，韌體改用紅+綠同亮的琥珀色代替。這跟校正燈號完全無關，但若文件用藍色圓點標示校正 步驟，很容易讓人誤以為看的是板載燈 —— 特此區分。

7.3 需要幾顆燈

精靈只有三種狀態，**兩顆燈就夠**：

state	意義	第一顆燈	第二顆燈
0	待機 / 該放空籃	滅	滅
1	該放砝碼	**亮**	滅
2	校正完成	滅	**亮**

⚠️ **韌體沒有顏色的概念**，它只控制「第幾顆燈亮/滅」。哪顆燈是什麼顏色，完全取決於現場 把哪一路 DO 接到哪顆燈泡。

用三色燈塔的話，建議這樣配：

燈	建議用途	理由
黃	第一顆（校正進行中）	「注意／進行中」是黃燈的通用語意
綠	第二顆（校正完成）	「完成／可作業」
紅	**不接精靈**，留給故障告警	紅燈被佔用會失去告警意義

七色積層燈（如 PATLITE NE-M1ATB-M）接法相同，只是可選顏色更多。一顆燈也可以由**多路 DO 一起驅動**（某些燈塔一個顏色需要兩路訊號），Action 裡多加一個 output 即可。

7.4 找出 DO 索引（最容易接錯的一步）

Action 的 `CH_DO` output 用一個 byte 編碼位置：

```
index = (expIdx << 4) | ch
  expIdx : 0 = 本機 Opta，1 = 第一台擴充模組，2 = 第二台…
  ch     : 該裝置上的通道，從 0 起算
```

⚠️ **`ch` 是 0-based，但面板和網頁上的標示是 1-based。**韌體內部 `DO_PINS[] = {D0,D1,D2,D3}`，`writeDO(ch)` 直接索引，所以：

面板標示	內部 ch	index（本機）
DO1	0	`0`
DO2	1	`1`
DO3	2	`2`
DO4	3	`3`

第一台擴充模組的 ch2、ch3 → index `18`、`19`（`(1<<4)|2`、`(1<<4)|3`）。

🔧 實測步驟（強烈建議，兩分鐘）

接線後**不要只照表填**，用實測確認對應關係：

1. 打開 Gateway 網頁 → **I/O 頁**。
2. 逐一手動切換 DO1 → DO2 → DO3 → DO4，每切一路就看現場**哪顆燈亮了**。
3. 記錄下來，例如：

面板切換	現場亮的燈	要填的 index
DO2	黃燈	`1`
DO3	綠燈	`2`

4. 用**實測記錄**去填 §7.5 的 Action，不要用猜的。

> 這一步能省掉現場「燈亮錯顆」來回改設定的時間。若接的是擴充模組，同樣在擴充模組頁逐路測。

7.5 精確設定：建立三個 Action

每個狀態一個 Action。**關鍵：該亮的設 ON、該滅的要明寫 OFF** —— 只設 ON 的話上一顆燈不會熄，會變成兩顆一起亮。

Action	對應 state	第一顆燈	第二顆燈
校正標零	1	`outputMode=0` (ON)	`outputMode=4` (OFF)
校正標增益	2	`outputMode=4` (OFF)	`outputMode=0` (ON)
校正重置	0	`outputMode=4` (OFF)	`outputMode=4` (OFF)

`outputMode` 對照：`0`=直接輸出(ON)、`1`=映射輸出、`2`=脈衝(配 `pulseMs`)、`4`=**強制 OFF**、`5`=toggle。

完整 API 範例

以 §7.4 實測結果「黃燈=index 1、綠燈=index 2」為例（`type: 2` 即 `CH_DO`）：

```bash
TOKEN=smmsadmin        # 設備 admin token
GW=http://192.168.10.55

# Action 0：校正標零 → 黃燈亮、綠燈滅
curl -X POST "$GW/api/actions" -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" -d \
'{"index":0,"enabled":true,"name":"校正標零燈號","outputs":[{"type":2,"index":1,"ou

# Action 1：校正標增益 → 黃燈滅、綠燈亮
curl -X POST "$GW/api/actions" -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" -d \
'{"index":1,"enabled":true,"name":"校正完成燈號","outputs":[{"type":2,"index":1,"ou

# Action 2：校正重置 → 兩顆全滅
curl -X POST "$GW/api/actions" -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" -d \
'{"index":2,"enabled":true,"name":"校正重置燈號","outputs":[{"type":2,"index":1,"ou
```

必讀注意事項：

- ⚠️ `"enabled":true` **一定要明寫**。不帶這個欄位會**靜默停用**，Action 建起來卻不動作，而且不會報錯 —— 這是最常見的卡關原因。
- `index` 是**陣列位置**（0 起算），不是 `actionId`。`actionId` 由韌體自動指派，建立後用 `GET /api/actions` 查。
- 規則（Rule）裡引用 Action 時用的欄位是單數 **`actionId`**，不是 `index`。
- JSON 必須 **compact**（不能有空格）—— 韌體用 `indexOf("\"key\":value")` 比對，有空格會靜默失敗。`Content-Length` 要用 **byte 數**（中文名稱是多位元組）。
- 每個 POST 後用 `GET /api/actions` 驗證 `outputCount` 是否等於預期（本例為 2）。

建完 Action 後，在 §5 的 Rules 步驟把三條規則分別指向這三個 `actionId` 即可。

7.6 加蜂鳴器（選配）

吵雜環境可加一顆蜂鳴器，在「完成」的 Action 裡多加一個 output。**用脈衝模式**，否則會一直叫：

```JSON
{"type":2,"index":3,"outputMode":2,"pulseMs":500}
```

（`index:3` = 面板 DO4，依 §7.4 實測結果調整。）

7.7 驗收

設定完成後，依序送三個推進命令，對照燈號：

送出	預期燈況
`cmd/signal/1`	第一顆燈**亮**、第二顆**滅**
`cmd/signal/2`	第一顆**滅**、第二顆**亮**
`cmd/signal/3`	兩顆**全滅**

三步都對，燈號就跟精靈綁好了。

排錯：

症狀	原因	處理
燈亮錯顆	DO index 填錯（多半是 0-based/1-based 搞混）	回 §7.4 重新實測
兩顆一起亮	該滅的沒明寫 `outputMode=4`	補上 OFF output
完全沒反應	Action 沒帶 `"enabled":true`	重送並確認
燈一直亮不熄	蜂鳴器/燈用了 `outputMode=0` 而非脈衝	改 `outputMode=2` + `pulseMs`

> 燈亮錯顆一律是**設定或接線**問題，不需要改韌體。

08

8. 相關文件

- 工作流引擎規格 / API：`../specs/spec-mqtt-chain-workflow.md`
- MQTTX 連線設定：`../guides/guide-web-login.md`、`mqttx-tls` SOP

01 使用者

Web 登入與密碼

> 從韌體 **5.9.47** 起，MES Gateway 的 Web 介面（`http://<DUT-IP>/`）強制需要登入。出廠時內建兩組預設帳號：**管理員** 與 **唯讀**，使用者首次登入後**請立刻更改 Token**。

01

預設帳號

韌體首次啟動（或執行恢復原廠後第一次開機）會自動建立兩組 Token：

角色	預設 Token	權限	說明
🔴 **管理員** (admin)	`change-me-admin`	`PERM_ADMIN` (2)	全部功能：編輯設定、規則、I/O 控制、OTA 更新等
👁️ **唯讀** (viewer)	`change-me-viewer`	`PERM_READONLY` (0)	只能查看 Dashboard / Config / Rules / Devices / Settings；所有寫入按鈕被 disable

> ⚠️ **必改**：上線前第一件事就是改掉這兩組預設值，否則任何同網段的人都能用 `change-me-admin` 完全控制設備。

02

登入步驟

1. 打開瀏覽器，輸入設備 IP（如 `http://192.168.51.77/`）
2. 第一次開或之前登出後，畫面中央會跳出 🔐 **登入 MES Gateway** Modal
3. 從下拉選擇 **權限級別**：
   - 「🔴 管理員（admin，全部功能）」
   - 「👁️ 唯讀（viewer，只能查看）」
4. 在「Token 值」欄位輸入對應 token：
   - 出廠預設：`change-me-admin` 或 `change-me-viewer`
   - 自訂後請輸入新值
5. 按「登入」

> 登入成功後 Token 會存於瀏覽器的 `localStorage`（key 為 `opta_token`），下次開同一個瀏覽器**不會再跳 modal**。

03

登出 + 帳戶 Badge

右上管理區（Header 最右側，5.9.48+）有三件式：

元素	說明
🌐 語系切換器	自動 / 🇹🇼 繁中 / 🇺🇸 EN
🔴 admin / 👁️ viewer 角色 badge	顯示目前登入身份；admin 紅點、viewer 眼睛圖示
🚪 登出	點擊清除 localStorage + 頁面 reload + login modal 重新跳出

> 5.9.47 把登出放在側邊欄底部；**5.9.48 起改放右上 header**（更顯眼、與 user badge 一起）

04

修改預設 Token（必須做）

透過 API 改

```bash
# 改 admin token (index 0)
curl -X POST http://192.168.51.77/api/tokens \
  -H "Authorization: Bearer change-me-admin" \
  -H "Content-Type: application/json" \
  -d '{"index":0,"value":"<your-new-strong-admin-token>","permission":2,"name":"a

# 改 viewer token (index 1)
curl -X POST http://192.168.51.77/api/tokens \
  -H "Authorization: Bearer change-me-admin" \
  -H "Content-Type: application/json" \
  -d '{"index":1,"value":"<your-new-strong-viewer-token>","permission":0,"name":"

# 確認
curl -H "Authorization: Bearer <your-new-strong-admin-token>" \
  http://192.168.51.77/api/tokens
```

> Token 字元規則：4 ~ 31 字元，建議 16 字元以上隨機字串（如 `openssl rand -hex 16`）

透過 Web UI 改（規劃中）

`Settings → 使用者管理` 區塊（admin only）— 待 sprint-28 補上 UI（FR-A5-25 follow-up）。

05

權限級別說明

韌體層 `TokenManager.h` 定義三級：

級別	數值	限制
`PERM_READONLY`	0	只能 GET，所有 POST/DELETE/PUT 回 401
`PERM_CONTROL`	1	可 I/O 控制（POST `/api/io/do`）但不能改 config/rules（保留中，目前 boot 不會 seed）
`PERM_ADMIN`	2	全部功能

實作層：`tokenManager.hasPermission(actual, required)` 用 `actual >= required` 比較，所以 admin 可以做 readonly 的事，但 viewer 不能做 admin 的事。

> 目前並非所有 endpoint 都有 `checkAuth()` 檢查（部分留作後續強化）— 但已上 auth 的高敏感路徑（license reset、tcpio clear、rules clear、tokens 管理）都正確擋住。

06

新增其他 Token

最多可有 **4 組** token（`MAX_TOKENS=4` in TokenManager.h）。例如新增一組 control 等級 token 給機器人 / 自動化測試：

```bash
curl -X POST http://192.168.51.77/api/tokens \
  -H "Authorization: Bearer <your-admin-token>" \
  -H "Content-Type: application/json" \
  -d '{"value":"automation-bot-token","permission":1,"name":"playwright"}'
```

不帶 `index` 就是「新增到末尾」；slot 滿（4 組）時回 409。

07

刪除 Token

```bash
curl -X DELETE -H "Authorization: Bearer <your-admin-token>" \
  "http://192.168.51.77/api/tokens?index=2"
```

防呆：**不能刪除最後一個 ADMIN token**（避免設備永遠無法管理）。回 400 `Cannot delete (out of range or last admin)`。

08

API endpoint 一覽

Method	Path	權限	說明
GET	`/api/tokens`	ADMIN	列所有 token（value 顯示前 4 + `****` + 後 4）
POST	`/api/tokens`	ADMIN	新增 / 更新 token，body: `{value, permission, name, enabled, index?}`
DELETE	`/api/tokens?index=N`	ADMIN	刪除 token（防最後一個 admin）

09

常見問題

Q1: 我忘記 admin token 了怎麼辦？

走「恢復原廠」流程（Settings → 維護操作 → 恢復原廠）— 設備會回到出廠 baseline，重啟後 token 自動 reseed 為 `change-me-admin` / `change-me-viewer`。

Q2: Token 是明文傳輸嗎？

是。韌體 HTTP server 不支援 TLS，所以 `Authorization: Bearer xxx` 在區網中是明文。建議：

- 設備放在受控網段
- 不要把 admin token 寫進公開腳本
- 高敏感場景考慮在前面加 reverse proxy（如 nginx）走 HTTPS（5.9.47 韌體可外連 cloud HTTPS，但本地 server 仍是 HTTP）

Q3: Playwright 測試怎麼帶 token？

```JS
const ADMIN_AUTH = { headers: { Authorization: 'Bearer <your-admin-token>' } };
const res = await request.get('/api/system', ADMIN_AUTH);
```

或用 `BASE_URL=http://...` + `localStorage.setItem('opta_token', '<token>')` 在 page.goto 前注入。

10

出貨清理

出貨清理流程**不會 reset token**（避免客戶重啟後找不到帳號）。出貨前 SOP 必須：

1. 確認 admin / viewer token 已被改為客戶指定值（或讓客戶第一次登入後自己改）
2. 把預設 token `change-me-admin` / `change-me-viewer` 列入 SOP「上線後立即更換」清單
3. 文件交付時包含本檔（`guide-web-login.md`）

> 若想出貨時 reset token 為預設，可呼叫 `tokenManager.seedDefaults()`（內部 API，目前無對外 endpoint，可在 sprint-28 補 `POST /api/admin/tokens/reset`）

11

相關文件

- `src/TokenManager.h` / `src/TokenManager.cpp` — 韌體實作
- `src/ApiHandlers_Logic.cpp` `handleGetTokens` / `handlePostToken` / `handleDeleteToken` — API 實作
- `web/src/index.html` `#loginModal` — Login UI
- `web/src/app.js` `_apiToken` / `submitLogin()` / `logout()` — 認證邏輯

02 安裝工程師

硬體安裝與接線

📅 最後更新：2026-03-14 | 📌 負責人：KC

本指南涵蓋 Arduino Opta MES Gateway 的實體特性、分割區規劃、安裝以及周邊整合。

01

1. 硬體版本與辨識

Arduino Opta（常以 **Finder Opta** 品牌出貨）共有三種主要版本，全部皆由 v5.0+ Unified Firmware 堆疊支援。

版本	主要硬體	FCC ID (WiFi lookup)	Part Number (Finder)
Opta Lite	Ethernet	N/A	8A.04.9.024.8300
Opta RS485	Ethernet + RS485	N/A	8A.04.9.024.8310
Opta WiFi	Ethernet + WiFi + BT	`2A97G-3A001`	8A.04.9.024.8320

> [!TIP] **WiFi 辨識**：查看 USB-C 連接埠附近的 FCC ID，以確認工業機型是否具備無線功能。

1.1 韌體型號感知（OTP，v5.9.244+ / WI-151）

韌體開機會讀 **OTP** `OptaBoardInfo._board_functionalities.wifi` **bit** 自動分辨機型（讀 OTP，不碰 WiFi 硬體，可靠）：

偵測結果	機型	對應差異
`wifi=1`	**Opta WiFi (8320)**	完整 WiFi/BLE/Ethernet，AP 設定模式可用，心跳用藍燈
`wifi=0`	**Opta RS485 (8310)**	僅 Ethernet + RS485，**無 WiFi/BLE**；網頁過濾掉 WiFi 設定、心跳改用琥珀燈、進不了 AP 模式

- 偵測結果曝光於 `GET /api/system` 與 `GET /api/license` 的 `hasWifi` (true/false) 與 `boardModel` (`Opta WiFi (8320)` / `Opta RS485 (8310)`) 欄位，供前端做型號感知過濾。
- 開機 serial log 印 `[BOARD] OTP wifi=.. rs485=.. eth=..`。讀不到 OTP 時保守假設 `hasWifi=true` (行為同舊版)。
- ⚠️ 不可用 `WiFi.status()` 判型號：8310 實測回 `0` (WL_IDLE) 而非 `255` (WL_NO_MODULE)，與閒置的 8320 無法區分。

02

2. 輸入／輸出特性

2.1 共用輸入腳位 (I1-I8 / A0-A7)

這 8 個端子 (I1-I8) 可同時作為數位與類比輸入使用。

- **數位邏輯**：0-24V DC 邏輯準位。`HIGH` 為 **>6.6V**，`LOW` 為 **<4.46V**。
- **類比範圍**：0-10V DC 量測 (ADC)。
- **輸入阻抗**：約 9kΩ (工業 sinking)。
- ⚠️ **重要實作注意事項**：由於 STM32H747XI 的線路佈局，在這些腳位上呼叫標準的 `digitalRead()` **永遠回傳 LOW**。工業 DI 訊號必須透過 ADC 讀取：`analogRead(pin) > 512` (假設為 10-bit 解析度)。

2.2 數位輸出 (DO1-DO4)

- 4 路繼電器或固態輸出 (依機型而定)。
- 透過 `digitalWrite()` 或閘道器的 rule engine 控制。

03

3. QSPI Flash 與 LittleFS 儲存

Arduino Opta 配備 **16MB 外部 QSPI Flash**，用於存放設定、網頁資產 (SPA) 與生產統計資料。

3.1 分割區配置（標準對應）

Address	Partition	Size	用途
0x000000	P1	1 MB	WiFi Radio Firmware (Murata 1DX)
0x100000	P2	5 MB	OTA 更新緩衝區
0x600000	P3	1 MB	Mbed KVStore（系統設定）
0x700000	P4	7 MB	**使用者資料 (LittleFS)**：`/fs/config.json`、`/fs/stats.bin`、`/web/index.html`

3.2 QSPI 空白 MBR 自我修復（v5.9.241+ / 8310 安裝根治）

> [!IMPORTANT] **全新 8310「安裝一直有問題」的真兇已根治（v5.9.241）**。全新出廠 Opta 8310（無 WiFi）的 QSPI 是**空白 MBR** → `MBRBlockDevice::init()` 回 **`-3101`（BD_ERROR_INVALID_MBR）**。舊韌體的自動建分割區救援**只判** `-3102`（MBR 有效但缺 partition 4），沒涵蓋 `-3101` → 檔案系統不掛載 → config/token/網路設定永遠存不了（serial 狂洗 `[LocalConfig] Cannot save: filesystem not mounted`）。
>
> **修法（`src/LocalConfig.h`）**：救援條件改為 `err == -3102 || err == -3101`。**任何空白 QSPI 設備第一次燒入本韌體就自我建分割區 + 格式化**（P1 WiFi 1MB / P2 OTA 6MB / P3 KVStore 1MB / P4 user 7MB，與官方 QSPIFormat 一致）。
>
> **相容 8320**：救援只在 `userPartition->init()` 失敗時才進入；已 provision 的設備 init() 回 0、整段跳過、既有分割區/資料不動。
>
> 因此 **v5.9.241+ 起，全新 8310/8320 都可直接燒韌體，不再強制依賴下方 QSPIFormat 手續**（8320 仍需經官方 WiFiFirmwareUpdater 寫入 WiFi 韌體與憑證至 P1 才能用 WiFi）。

3.3 必要的初始化 (QSPIFormat)

部署前，必須使用 **`QSPIFormat`** sketch（取自 `STM32H747_System` 範例）為 flash 建立分割區。

1. 透過 Arduino IDE（Core v4.5.0+）上傳 `QSPIFormat`。
2. 以 115200 baud 開啟 Serial Monitor。
3. **確認建立分割區 (Y)**：建立 7MB 的使用者資料分割區。
4. **確認完整抹除 (Y)**：讓硬體準備好建立索引。
5. **還原 WiFi Firmware (Y)**：填入 Partition 1 的關鍵步驟。
6. **選擇 LittleFS (Y)**：將 Partition 4 格式化供閘道器使用。

04

4. 安裝與設定流程

4.1 初次部署

1. 完成 **QSPIFormat**（第 3.2 節）。

2. 若缺少 Radio firmware，請燒入 **WiFiFirmwareUpdater**（IDE WiFi 函式庫）。
3. 透過 PlatformIO 部署專案 firmware：`pio run -t upload`。
4. 透過 Python 部署 Web Dashboard：`python3 scripts/upload_web.py --ip <IP>`。

4.2 拓荒包（Pioneer）USB 首燒流程（WI-149，新設備推薦）

新設備（或救磚）建議 USB 首燒精簡的**拓荒包（Bootstrap / Pioneer）** `.bin`，它自包單一網頁（不依賴 QSPI 資產），開機即可設定並一鍵升級成完整版：

1. USB 燒 **`mes-gateway-bootstrap.bin`**（DFU；指令見 `../guides/guide-cli-opta.md`）。
2. 開機 → 瀏覽器開設備 IP → **拓荒設定頁**：顯示 UID（可複製）、網路設定、大大的「上傳 .mesb」鈕。
3. 設好網路（持久化生效）→ 上傳官方 `.mesb`（韌體 + 6 個網頁資產一包）。
4. 設備自動燒錄並重開 → 變成**完整版**（韌體 + 完整 UI）。閉環 bootstrap → .mesb → 完整版已實機驗證。

> **型號感知過濾（WI-151）**：8310（無 WiFi）開拓荒頁時自動隱藏 WiFi SSID/密碼欄，並隱藏整個介面模式選單（強制乙太網路，只留 Ethernet IP 的 DHCP/靜態設定）。8320 則顯示完整 WiFi 設定。

4.2.1 拓荒包是「無版號、固定產物」（為什麼、怎麼編）

拓荒包**刻意不帶版號**，檔名永遠是 `mes-gateway-bootstrap.bin`、雲端下載連結永不變：

- 它只負責 onboarding（顯 UID／設網路／上傳 .mesb），**完整版的版本由 .mesb 自己攜帶**，拓荒包本身不顯版本。
- 因此**極少更新** —— 可能一兩百個完整版迭代都不用碰它。**只有「bootstrap 關鍵環節」出 bug 才需要重出**，例如：QSPI 空白 MBR provisioning（§3.2）、OTP 型號偵測、onboarding 流程或網路設定頁。一般功能／規則／UI 改版，走 `.mesb` OTA 即可，與拓荒包無關。
- **編譯方式（需要時才呼叫）**：

```bash
bash scripts/build_bootstrap.sh        # → 產出 release/mes-gateway-bootstrap.bin
```

腳本內部編 `env:opta_pioneer`，事後自動 `git checkout src/WebPage.h` 還原完整版（拓荒首頁不入庫）。韌體內仍有一個診斷用內部版本號（不對外顯示）。

4.3 配置設定 (AP Mode，8320 only)

若不存在任何設定，或在開機時透過前面板的 **USER button** 強制進入：

1. 連線至 SSID：`MES-Gateway-Setup`（密碼：`12345678`）。
2. 瀏覽至 `http://192.168.3.1`。
3. 設定 WiFi/Ethernet 後重新開機。

> [!WARNING] **User button 開機長按分段是「型號感知」的**（`g_hasWifi`，v5.9.249+）：
>
> | 機型 | 0–5s | 5–10s | 10s+ |
> |---|---|---|---|
> | **8320（有 WiFi）三段** | 正常開機 | **AP 設定模式** | 原廠重置（清設定） |
> | **8310（無 WiFi）兩段** | 正常開機 | **原廠重置（清設定）** | 同左（5s+ 即重置，無 AP 段） |
>
> **AP 模式需要 WiFi、只有 8320 有。**8310 無 WiFi 模組（`WiFi.beginAP()` 不可用），故跳過 AP 段、**5s+ 直接原廠重置**；8310 要重設網路請用拓荒頁或 Ethernet 設定。⚠️ button 原廠重置**只清設定、不清授權憑證**（屬下節 §4.4 的 L1；交機請改用 L2）。

4.4 把 OPTA 恢復到全新出廠狀態（交機 / 轉售 / 重新部署）

> 客戶最常問：「我想把這台 OPTA 變回全新的，怎麼做？」依「要清得多乾淨」分三級。**關鍵觀念**：USB 燒韌體（DFU）只覆蓋**韌體**，**不會清 QSPI**（設定、授權憑證都在 QSPI）。要真的全新，必須**另外清 QSPI**。

級別	方法	是否 CLI	清除範圍	適用情境
L1 設定重置（現場、免電腦）	開機長按 USER button（**8320：10s+ ／ 8310：5s+**，燈快閃後放開）	❌ 實體鈕	只清 `config.json`（網路/規則/IO 設定）。⚠️ **不清授權憑證**	現場改設定重來、忘記 IP
L2 完整出廠（網頁一鍵，**推薦交機用**）	設定頁按「恢復原廠」，或 `POST /api/system/factory-reset`（Bearer `smmsadmin`）	❌ 網頁/HTTP	`localConfig.format()` **格式化整個 LittleFS 設定分割區（P4）** → 設定 + **授權憑證** 全清	交機、轉售、徹底乾淨
L3 連韌體一起重來（救磚 / 換乾淨 onboarding）	DFU 燒 `mes-gateway-bootstrap.bin`（§ 4.2）**＋ 接著做 L2**	⚠️ USB DFU	韌體換成拓荒包 ＋ QSPI 設定全清	韌體疑損壞、要回到「剛出廠＋onboarding」
L4 全 QSPI 重格（最徹底）	官方 `QSPIFormat` sketch（§ 3.3）：Arduino IDE 上傳 → Serial 115200 → 依序 `Y` 確認 partition / erase / WiFi firmware / LittleFS	✅ **CLI/IDE**	**整顆 QSPI 重格**（P1 WiFi 韌體 + P2/P3/P4 全部）→ 出廠空白	換代、QSPI 疑損壞、要連 WiFi 韌體一起重建

> **業主問「有沒有 CLI 清 QSPI 的方法」→ 就是 L4（QSPIFormat sketch，§ 3.3）**。但自 v5.9.241+ 的空白 MBR 自我修復（§ 3.2）後，連整顆空白 QSPI 都會開機自我 provision，**L4 幾乎只在 QSPI 實體疑損壞時才需要**；日常「恢復全新」用 L2（網頁、免 CLI）即可。

為什麼交機一定要用 L2（不是按鈕 L1）：按鈕原廠重置（8320 10s+ ／ 8310 5s+）只刪設定檔，**授權簽章憑證（`/cfg/license.tok`）會殘留** → 重開機後韌體會「從有效憑證還原 ACTIVE」，等於授權沒清掉就交出去。L2 的 `factory-reset` 會 `format()` 整個分割區、連憑證一起抹掉，才是真正的全新。

L2 操作（指令版）：

```bash
# 對設備本機 IP 直接打（需設備管理 token，預設 smmsadmin）
curl -X POST http://<device-ip>/api/system/factory-reset -H "Authorization: Bearer
# 設備回 {"success":true,...} 後自動重開 → 開機進 onboarding（無設定、授權 UNKNOWN）
```

驗證已回到全新：重開後 `GET /api/license` 應為 `UNKNOWN`、`GET /api/config` 無自訂設定、首頁進設定/拓荒流程即代表乾淨。

05

5. 部署疑難排解

5.1 LED 狀態指示燈

USER LED 心跳（型號感知，WI-152）：

- **8320（有 WiFi）**：藍燈（LED_USER = LEDB = PE_5）慢閃 = 系統心跳。
- **8310（無 WiFi）**：藍燈是 **BLE 模組燈**，官方明文「僅 WiFi 版能亮」，8310 點不亮。故 8310 心跳改用**琥珀色**（紅 LEDR + 綠 LEDG 同亮），與純紅（故障）、純綠（DFU/reset）明確區隔。
- 心跳閃速：MQTT 已啟用但未連上 broker → 100ms 快閃；正常 → 500ms 慢閃。

其他：

- **恆亮綠燈**：Bootloader/DFU 模式啟用中（連按兩下 Reset 進入）。
- **快速紅燈閃爍**：致命當機 (Mbed OS Panic)。
- **緩慢紅燈閃爍**：Radio 模組初始化失敗（缺少 Firmware 或匯流排競用）。

5.2 常見修正

- **「Failed to mount filesystem for WiFi firmware」**：重跑第 3.2 節 (QSPIFormat)，並確認 WiFi firmware 還原達到 100%。
- **系統在 WiFi 啟動時卡住**：確認在 `WiFi.beginAP()` 之前有呼叫 `localConfig.unmount()`（已於 v5.4.24+ 修正）。
- **網頁上傳期間出現 Broken Pipe**：上傳程式現已改用 `curl`。請確認部署用的機器已安裝 `curl`。

5.3 USB CDC 阻塞陷阱 (Mbed OS)

Arduino Opta (Mbed OS) 使用原生的 USB CDC 實作來進行序列除錯。

- **問題**：若主機（PC）開著作用中的 Serial Monitor，而 USB 線被實體拔除，Mbed OS 的序列緩衝區可能會被阻塞。主迴圈內的 `Serial.print` 呼叫可能會停滯，直到逾時為止。
- **症狀**：I/O 規則無反應，或主迴圈看似「凍結」。
- **解法**：在拔除 USB 線之前，**務必先關閉 PC 上的 Serial Monitor**，以確保 firmware 在外部供電下能以確定性的時序持續運作。

06

6. 周邊整合範例

6.1 PATLITE NE-M1ATB-M（七色積層信號燈）

- **Opta 接線**：DO1 (Red)、DO2 (Green)、DO3 (Blue)、DO4 (Buzzer)。
- **運作**：24V DC 控制。
- **Setup**: 在 Rule Engine 手動建立 Action Groups（紅燈/綠燈/藍燈各控制 DO1-DO3、蜂鳴器用 DO4 pulse 模式 500ms）。v5.9.112 之後不再提供一鍵套用模版。

6.2 校正引導燈號（三色燈 / 任意兩路 DO）

- 號誌燈與磅秤是**各自獨立的設備**，由 Rule Engine 的 CH_DO 動作串起來；沒有燈也能完成校正。
- **燈號沒有固定顏色** —— 韌體只控制「第幾顆燈亮/滅」，顏色由現場接線決定。
- **DO 編碼**：`index = (expIdx << 4) | ch`，`expIdx` 0=本機、1=第一台擴充；**`ch` 從 0 起算**（面板標示 DO1 → `ch=0` → index `0`）。第一台擴充的 ch2/ch3 → index `18` ／ `19`。
- **燈號對照**：state=0 全滅、state=1 第一顆燈亮（放空籃）、state=2 第二顆燈亮（完成）。
- Action 用 CH_DO output（`outputMode` 0=ON / 4=OFF）依步驟切換；**該滅的要明寫 OFF**，否則上一顆不會熄。
- 🔧 接線後請在 I/O 頁逐路實測，記錄哪一路點亮哪顆燈，再據以填 Action。

- 📖 完整設定與操作：`../guides/guide-calibration-wizard.md` **§ 7 號誌燈搭配**（接線、索引、Action、驗收）。

02 安裝工程師

新裝置配置

📅 最後更新：2026-06-16 | 📌 負責人：KC

本文件描述當拿到一台全新的 Arduino Opta (或 QSPI 分區毀損、需要徹底重置的設備) 時，如何從零開始進行完整的格式化與韌體部署，以確保設備能夠正常切分 User Data Partition (Partition 4) 給 LittleFS 使用，並避免 `-3101` 掛載失敗的錯誤。

[!TIP] **新做法（v5.9.241+）：QSPI `-3101` 已自我修復、首燒可走拓荒包。** v5.9.241 起韌體開機會自動修復空白 QSPI（`-3101` / `-3102` 都救），全新設備第一次燒入即自我建分割區 + 格式化，**不再強制依賴下方手動 QSPIFormat 手續**（8320 仍需官方 WiFiFirmwareUpdater 寫入 P1 WiFi 韌體才能用 WiFi）。新設備最簡單的開荒路徑是 **「拓荒包（Pioneer）USB 首燒 → 拓荒頁上傳 .mesb」**（見下方「拓荒包流程」與 `guide-cli-opta.md`）。本文第一/三階段為傳統手動流程，仍可用。

01

拓荒包（Pioneer）流程（WI-149，新設備最推薦）

1. **USB 首燒** `mes-gateway-bootstrap.bin`（**無版號、固定一顆**；DFU；指令見 `guide-cli-opta.md`）。拓荒包是同一份完整韌體，只把首頁換成自包的開荒頁（inline CSS/JS，不依賴 QSPI 資產）。雲端下載連結永不變，發佈程序見 `guide-ota-embedded-web.md`。
2. **開機設網路**：瀏覽器開設備 IP → 拓荒設定頁。頁面只露三件事：
   - 🆔 **裝置 UID**（唯讀，可一鍵複製，供雲端綁定/授權）。
   - 🌐 **網路設定**：介面模式（自動/僅有線/僅 WiFi）+ Ethernet（DHCP/靜態 IP）+ WiFi（SSID/密碼）。存檔走 `POST /api/config` merge，**持久化生效**。
   - 🚀 **上傳 .mesb**：大按鈕，選官方 `.mesb` → 上傳後設備自動燒錄重開成完整版。

> 💡 **MQTT 出廠已預設官方 broker（v5.9.261）**：完整版上線後已內建官方 TLS broker + 帳密（port 8883），**一般不需手動設定** MQTT；只有要改用自家 broker 時才到「設定 → MQTT」修改。

3. **型號感知過濾（WI-151）**：拓荒頁 fetch `/api/license` 取 `hasWifi`；若為 8310（`hasWifi:false`），自動**隱藏 WiFi SSID/密碼欄 + 整個介面模式選單**（強制 ETH_ONLY，只留 Ethernet IP 設定）。8320 顯示完整選項。
4. **一鍵完整安裝**：上傳官方 `.mesb`（韌體 + 6 個網頁資產 app.js/workflow/styles/i18n×3 + favicon.ico 打包成單一檔）→ 設備寫 QSPI 並重開 → 完整版上線。閉環已實機驗證。

從無到有：把整台機器清乾淨再重建（簡易）

依「要清多乾淨」分三級，由輕到重：

級別	清除範圍	怎麼做
L1 原廠重置（最常用）	只清**設定**（`config.json`），**留授權憑證、留分割區**	開機長按 USER 按鈕到原廠重置（8320 ≥10s / 8310 ≥5s，見附錄）
L2 徹底清空（交機推薦）	**連 QSPI 分割區一起格式化**（真正「從無到有」）	跑原廠 QSPIFormat 全清 → 見下方「第一階段」
L3 連韌體一起換	L2 + 重燒底層韌體	雙擊 RESET 進 DFU 重燒，再做 L2

清乾淨後的**重建**一律走拓荒包三步（不論哪一級）：

1. **USB 燒拓荒包** `mes-gateway-bootstrap.bin`（無版號固定一顆）。
2. **開機 → 開荒頁**設網路（8310 自動隱藏 WiFi 欄）。
3. **上傳官方** `.mesb` → 自動重開成完整版。

> 💡 多數情況用 **L1** 就夠（換現場、清掉舊設定）；只有 QSPI 真的壞了 / 要當全新機交付才需要 **L2**。

> 以下第一～四階段為**傳統手動 / 進階備援流程**，需要 L2 徹底清空或不走拓荒頁時才用。

02

第一階段：QSPI 徹底清空與格式化 (含 WiFi Firmware 重建)

在寫入自己的專案韌體之前，必須透過原廠提供的 `QSPIFormat.ino` 工具完整切分 QSPI Flash 磁區。

> ⚠️ **警告：嚴禁在此階段使用 CLI 或全自動腳本 (如 `pio`)。**由於 Opta 的硬體設計限制，從 DFU 模式燒錄底層程式後，設備會陷入 `dfuMANIFEST` 狀態**不會自動重啟**。這會導致自動連線腳本丟失 USB Port 並且卡死。為了保證 100% 成功，請務必手動透過 Arduino IDE 執行本階段。

1. **拔掉 Opta 的 USB 線並重新插上**（確保硬體徹底斷電，脫離可能的 DFU 錯亂狀態）。
2. 打開 **Arduino IDE**。
3. 點擊頂部選單：**檔案 (File) > 範例 (Examples) > STM32H747_System > QSPIFormat**。
4. 在上方選單選擇版子 **Arduino Opta** 以及對應的 Port (應為 `/dev/cu.usbmodem...`，非 DFU)。
5. 點擊 **上傳 (Upload)**。
6. 上傳成功後，打開 Arduino IDE 右上角的 **Serial Monitor (設定為 115200)**。
7. **對著 Opta 設備按一下實體 RESET 鍵**（USB 孔旁邊）。
8. 等待約一分鐘的紅燈閃爍 (Full Erase)，紅燈熄滅後，Serial Monitor 會出現 `Do you want to proceed? Y/[n]` 提示字元。
9. **依序輸入三次 `Y` 並送出**。這會依序建立分區、寫入 WiFi 韌體並以 LittleFS 掛載 Partition 4。

當出現 `QSPI Flash formatted! It's now safe to reboot or disconnect your board.` 即表示底層格式化大功告成。

03

第二階段：專案韌體部署

完成環境淨化後，即可開始寫入我們的 MES Gateway 韌體。

1. **利用 PlatformIO 編譯並上傳:** 切換到專案根目錄 (`PillarArduno`)，執行：

```bash
pio run -t upload
```

註：若無法自動偵測 port，請加上 `--upload-port /dev/cu.usbmodemXXXX`

2. **驗證初次啟動日誌:** 開啟 Serial Monitor。確認系統是否有成功掛載 Partition 4 並通過 DHCP 取得 IP：

```log
[LocalConfig] QSPI size (KB): 16384
[LocalConfig] User partition size (KB): 7168
[LocalConfig] LittleFS mounted successfully!
[LocalConfig] Existing config found: NO
...
Ethernet: DHCP... OK, IP = 192.168.0.x
```

04

第三階段：部署 Web UI (前端畫面)

> ⚠️ **`upload_web.py` 已廢除**(ADR-004/005)。`index.html` + 核心 `app.js` 已編進韌體 PROGMEM，CSS／語系／`workflow.html` 走 QSPI，全部由 `.mesb` 一檔帶齊 —— **不再有單獨上傳 web 的步驟**。

韌體啟動後內建 Web Server 已就緒；把 QSPI 網頁資產補齊有兩條路：

1. **灌官方 `.mesb` (推薦，fw＋6 資產一次到位)：**

```bash
curl -sS -X POST "http://<DEVICE_IP>/api/ota/bundle" \
  -H "Authorization: Bearer smmsadmin" \
  -H "Content-Type: application/octet-stream" \
  --data-binary @release/mes-gateway-v<版本>.mesb
# 設備重燒重開，約 90 秒
```

2. **純補資產不重燒**：逐一 POST `web/build/*.gz`（見 `guide-cli-opta.md` §5 方法 B）。
3. **確認畫面：** 打開瀏覽器訪問 `http://<DEVICE_IP>/`，應看到完整的 MES Gateway 儀表板。

> 💡 若是全新設備走拓荒包流程（上方），上傳 `.mesb` 這步已自動完成，無需本階段。

05

第四階段：與 Config Server 對接還原 (選擇性)

若這是一台汰換的重製備機，您可以將舊有的設定一鍵還原回去。若為全新出廠設備，可略過此步直接在 Web 介面開始您的設定。

1. **尋找最新的備份 ID：**

```bash
curl -s http://<CONFIG_SERVER_IP>:8888/api/devices/<DEVICE_ID>/backups
```

從回傳的 JSON 陣列中挑選要還原的 `filename`（例如：`config.bak.1773145690841.json`）。

2. **觸發遠端還原：** 呼叫 Config Server 的 `restore` API 將該備份直接推向目標新 IP：

```bash
curl -X POST http://<CONFIG_SERVER_IP>:8888/api/devices/<DEVICE_ID>/restore \
  -H "Content-Type: application/json" \
  -d '{"filename":"<BACKUP_FILENAME>","targetIp":"<DEVICE_IP>"}'
```

注意：被推播的設備將會自動覆寫 `/fs/config.json` 並在一秒後自動重啟。重啟後 MQTT 與 I/O 設定即全數恢復運作。

恭喜，您已完成 Opta MES Gateway 的全新建置流程！

06

附錄：實體 USER 按鈕（開機長按，型號感知 WI-152）

設備正面有一顆實體 **USER 按鈕**。**開機當下按住**可觸發以下動作（運行中按無效）；門檻依型號自動切換（8320 有 WiFi、三段；8310 無 WiFi、兩段），BOOT_LED 進入各 zone 時會閃爍提示。

設備	0–5 秒放開	5–10 秒放開	10 秒+ 放開
8320（有 WiFi）三段	正常開機	**AP 模式**（`MES-Gateway-Setup` / `12345678` / `192.168.3.1`）	**原廠重置**
8310（無 WiFi）兩段	正常開機	**原廠重置**（5 秒+，無 AP 段）	原廠重置

> ⚠️ **「原廠重置」只刪設定、不清分割區。**它僅 `remove("/cfg/config.json")` + `remove("/cfg/config.bak")` 後重開機，**授權憑證與 QSPI 分割區都保留**。這是「恢復出廠」三級裡最輕的 L1。真正的「清分割區 / 格式化 QSPI」是上面第一階段的 QSPIFormat，或 `-3101` 自我修復，跟按鈕是兩回事。

03 管理者

備份與還原

> 版本：v1.2 | 更新日期：2026-06-16

本指南說明如何備份、還原和遷移 Config Server 的設備資料，適用於本地端（`SERVER_ROLE=local`）和雲端（`SERVER_ROLE=cloud`）兩種角色。

01

1. 資料結構總覽

Config Server 的資料儲存在 `config-server/data/` 目錄下，採用**檔案系統**作為持久層（無外部資料庫）。

```
config-server/data/
├── licenses.json          ← 授權記錄（Cloud-only）
├── <device-id>/           ← 每台裝置一個子目錄
│   ├── config.json        ← 設備配置快照（心跳上傳）
│   ├── web/               ← Web UI 檔案（index.html.gz）
│   └── firmware/           ← OTA 韌體暫存
```

關鍵檔案說明

檔案	用途	來源
`config.json`	設備的完整配置（網路、IO、MQTT、規則等）	裝置心跳自動上傳
`licenses.json`	所有裝置的授權啟用記錄	Cloud 授權 API
`web/`	（舊）設備 Web UI 上傳快取	**已停用** —— `upload_web.py` 已廢除，Web 資產改隨 `.mesb` 一檔投遞（見 `guide-ota-embedded-web.md`）
`firmware/`	設備 OTA 韌體暫存	Dashboard 上傳

02

2. 備份

2.1 完整備份（推薦）

備份整個 `data/` 目錄即可涵蓋所有設備資料：

```bash
# 建立帶時間戳的備份
cd config-server
tar -czf backup-$(date +%Y%m%d-%H%M%S).tar.gz data/
```

2.2 單一設備備份

若只需備份特定設備：

```bash
# 替換 <DEVICE_ID> 為實際設備 ID
tar -czf backup-<DEVICE_ID>.tar.gz data/<DEVICE_ID>/
```

2.3 透過 API 匯出設備配置

```bash
# 從 Config Server 匯出設備當前配置
curl -s http://localhost:8888/api/devices/<DEVICE_ID>/config | jq . > device_bacl
```

2.4 建議的備份頻率

環境	頻率	方式
開發/測試	手動	功能開發前後各一次
生產(本地)	每日	cron + tar
生產(雲端)	每小時	cron + rsync 或雲端快照

03

3. 還原

3.1 完整還原

```bash
cd config-server

# 停止伺服器
# pm2 stop config-server 或 Ctrl+C

# 還原
tar -xzf backup-20260324-120000.tar.gz

# 重新啟動
node server.js
# 或 pm2 restart config-server
```

3.2 單一設備還原

```bash
tar -xzf backup-<DEVICE_ID>.tar.gz -C config-server/
```

3.3 透過 API 匯入設備配置

```bash
curl -s -X POST http://localhost:8888/api/devices/<DEVICE_ID>/config \
  -H "Content-Type: application/json" \
  -d @device_backup.json
```

04

4. 遷移

4.1 本地端 → 本地端（換機器）

1. **停止**舊機器上的 Config Server。
2. **複製**整個 `config-server/` 目錄到新機器：

```bash
rsync -avz config-server/ new-host:/path/to/config-server/
```

3. 在新機器上安裝依賴並啟動：

```bash
cd config-server
npm install
SERVER_ROLE=local PORT=8888 node server.js
```

4. **更新裝置**的 Config Server URL（透過 Web UI「設定 → 伺服器」卡片）。

4.2 本地端 → 雲端

1. **備份** `data/` 目錄。
2. 將 `data/` 複製到雲端伺服器。
3. 設定環境變數：

```bash
export SERVER_ROLE=cloud
export PORT=3001
export FIREBASE_SERVICE_ACCOUNT=/path/to/firebase-key.json
```

4. 啟動伺服器。雲端角色會額外載入 `license` 模組。
5. 裝置端需更新 Config Server URL 為雲端地址。

4.3 雲端 → 本地端

1. **匯出** `data/` 和 `licenses.json`。
2. 複製到本地機器。
3. 使用 `SERVER_ROLE=local` 啟動。
    - **注意**：`licenses.json` 在本地模式下不會被使用，但保留以備未來遷回雲端。

05

5. 注意事項

5.1 裝置端配置同步

> [!IMPORTANT] Config Server 儲存的是裝置**上傳的快照**，而非「唯一真實來源」。裝置的 QSPI Flash（`/fs/config.json`）才是運行中的主配置。若兩者不一致，以裝置端為準。

5.2 授權資料（Cloud-only）

`licenses.json` 包含所有裝置的授權狀態記錄。遷移時**必須**一併複製此檔案，否則已啟用的裝置會在雲端顯示為未授權。

5.3 韌體檔案

`config-server/firmware/` 目錄儲存透過 Dashboard 上傳的韌體版本。這些檔案較大（約 500KB-1MB），備份時請確認磁碟空間充足。

5.4 敏感資訊

`.env` 檔案包含 Firebase 金鑰等敏感資訊，**不應**納入備份壓縮檔中。建議單獨管理：

```bash
# 備份時排除 .env
tar -czf backup.tar.gz --exclude='.env' data/
```

06

6. 自動備份腳本範例

```bash
#!/bin/bash
# config-server-backup.sh
# 每日執行：crontab -e → 0 2 * * * /path/to/config-server-backup.sh

BACKUP_DIR="/backups/config-server"
DATA_DIR="/path/to/config-server/data"
TIMESTAMP=$(date +%Y%m%d-%H%M%S)

mkdir -p "$BACKUP_DIR"
tar -czf "$BACKUP_DIR/backup-$TIMESTAMP.tar.gz" -C "$(dirname $DATA_DIR)" data/

# 保留最近 30 天的備份
find "$BACKUP_DIR" -name "backup-*.tar.gz" -mtime +30 -delete

echo "[$(date)] 備份完成: backup-$TIMESTAMP.tar.gz"
```

07

7. 雙路徑備援：device push 到雲端＋地端兩個 cs（v5.9.114+）

每台 device 同時支援 push config 到：

- **Cloud cs**（內建 `https://opta.smms.com.tw`，總是 active，無法關閉）
- **Local cs**（`config.configServer.url` 有設才 active）

兩條路徑彼此獨立，任何一條成功都會更新 `lastPushAt`。Cloud 斷線時 local 仍累積 snapshot 歷史，反之亦然。

設定 device 指向 local cs

device 端 admin token 起 curl：

```bash
DEV=192.168.0.31; AUTH="Authorization: Bearer <device-admin-token>"
curl -X POST -H "$AUTH" -H "Content-Type: application/json" \
  -d '{"url":"http://192.168.0.120:8888","token":"admin-token","enabled":true,"au
  http://$DEV/api/configserver
```

之後 device 改 config / 手動 `POST /api/configserver/push` / 心跳 autoPush 都會同時打兩邊。

觀察 push 結果（fw v5.9.119+）

```bash
curl http://$DEV/api/configserver | jq .push
```

回傳 `push.cloud` 跟 `push.local` 各自的 `{result, httpCode, ageS, failCount, cooldownMsLeft}`。連續失敗 N 次（PUSH_FAIL_THRESHOLD）會進 5 min cooldown，cooldown 期間 skip 該路徑，另一條仍正常推。

注意

- Local cs 不在線時 device 仍會嘗試 push、進 cooldown — 這是「fail-fast」設計，不會 hang 主迴圈。
- Local cs 啟動後不會自動被 device 發現，必須由人工 `POST /api/configserver` 設 URL。
- 把 `enabled=false` 或 `url=""` 即可關閉 local push，cloud push 不受影響。

08

8. PG 備份監控（v2.2.6 cs+）

`opta-backup` 容器每日 `pg_dump`，預設 `BACKUP_INTERVAL_SEC=86400` (24h)、`BACKUP_KEEP_DAYS=14`。

不必 SSH 就能查健康狀態：

```bash
curl -H "Authorization: Bearer <admin-jwt>" \
  https://opta.smms.com.tw/api/admin/backup-status
```

回傳：

```json
{
  "mounted": true,
  "dumpDir": "/var/backups/pg",
  "dumpCount": 11,
  "lastDumpFile": "optacs-20260519T065122Z.dump",
  "lastDumpAt": "2026-05-19T06:51:23.139Z",
  "hoursSinceLastDump": 22.7,
  "oldestDumpAt": "2026-05-09T06:51:20.218Z",
  "totalSize": 228239,
  "healthy": true
}
```

`healthy: false` 表示最後一次 dump 超過 30 小時 — 該檢查 backup container 是否還活著（`docker ps opta-backup`）。

03 管理者

地端設定伺服器部署

> 對象:在自家工廠內部署設定伺服器的客戶 / 現場 IT。你會學到:這套東西是什麼、怎麼裝、**怎麼登入**、怎麼日常使用、怎麼更新與排錯。

01

1. 這是什麼?為什麼要裝它?

地端設定伺服器(Local Config Server) 是一套安裝在**你工廠內部網路**的小型服務,負責:

- 📩 **設定備份**:每台 MES Gateway 設備會把自己的設定快照定期送到這裡,作為**第二個備份點**(第一個是 MetaSource 官方雲端)。
- 🕒 **設定還原**:設備設定誤改 / 故障換機時,可從這裡的歷史快照一鍵還原。
- 🖥️ **本地總覽**:在一個網頁畫面看到廠內所有設備的在線狀態、IP、韌體版本。

它**不需要連到外網、不需要另外裝資料庫**,用 Docker 一鍵啟動。資料只留在你自己的機器上。

> 📌 **與官方雲端的分工**:官方雲端(`opta.smms.com.tw`)永遠是主要管理與授權中心;地端伺服器是**廠內的第二備份與本地檢視**。兩者可並存,互不影響。

02

2. 開始之前要準備

項目	說明
一台內網機器	Windows / macOS / Linux 都可以。與設備在**同一個區域網路**。建議常開機。
Docker	需安裝 **Docker**(含 `docker compose`)。Windows / macOS 裝 Docker Desktop;Linux 裝 Docker Engine + Compose plugin。
下載包	從官方網站下載的 `opta-config-server-local-vX.Y.Z.zip`。

03

3. 安裝(兩步驟)

1. **解壓縮**下載的 zip,進入解壓後的資料夾。裡面應該有:
   - `opta-config-server-local-*.tar.gz`(服務本體,Docker image)
   - `install.sh`、`docker-compose.yml`、`.env.example`、`README.md`
2. **執行安裝腳本**:

   macOS / Linux

   ```bash
   cd 解壓後的資料夾
   ./install.sh
   ```

   Windows(用 Git Bash / WSL 跑 `./install.sh`,或在 PowerShell 手動執行下面三行):

   ```powershell
   docker load -i (Get-Item opta-config-server-local-*.tar.gz)
   Copy-Item .env.example .env
   docker compose up -d
   ```

腳本會自動:**載入 image → 建立 `.env` → 啟動服務**。完成後畫面會印出登入網址。

04

4. 登入 ⭐(最關鍵的一步)

> 很多人卡在這一步,所以請務必照做。

啟動後,在**同一台機器**用瀏覽器打開:

```
http://localhost:8888
```

若從**廠內其他電腦**連,把 `localhost` 換成這台伺服器機器的 IP,例如 `http://192.168.1.50:8888`。

登入畫面會跳出來。輸入:

欄位	值
帳號 (Email)	`local@localhost`
密碼 (Password)	`.env` 檔裡 `DEFAULT_ADMIN_PASSWORD` 的值(預設為 `mesgateway-local`)

> 還有一個備用帳號 `admin@localhost`,密碼相同。一般用 `local@localhost` 即可。

4.1 ⚠️ 關於密碼,務必知道的兩件事

1. **第一次登入後請馬上改密碼** —— 登入後點畫面**右上角「🔑 改密碼」**,改成只有你知道的密碼。
2. `DEFAULT_ADMIN_PASSWORD` **只在「第一次啟動、資料庫還空著」時生效**。之後再去改 `.env` 的那一行**不會**改到已經建立好的帳號密碼。要改密碼,一律從**網頁右上「🔑 改密碼」**操

作。

> 💡 為什麼會有「登入卡關」？如果安裝時把 `.env` 裡的 `DEFAULT_ADMIN_PASSWORD` **刪掉或留空**，系統會**隨機產生一組密碼，而且只印在容器日誌裡一次**。這時就會發生「不知道密碼、進不去」。所以**安裝前請確認 `.env` 裡這行有填一個你記得住的密碼**。萬一已經發生，見 §7 重設密碼。

05

5. 日常使用

登入後是一個分頁式的儀表板。地端模式會看到：

- 🖥️ **設備**：廠內所有設備的卡片（在線/離線、IP、韌體版本、設定版本、運行時間）。點某台可以開「歷史」看設定快照、做「比較」與「還原」。
- 🏭 **客戶**：把設備分組歸戶（若你需要分線別/分區管理）。

> 🔒 **授權** 與 **韌體部署** 這兩個分頁是官方雲端才有的功能，地端模式不會顯示，屬正常現象。

常用 Docker 指令

```bash
docker compose ps          # 看服務有沒有在跑
docker compose logs -f     # 看即時日誌（排錯用）
docker compose down        # 停止服務（資料保留，不會遺失）
docker compose up -d       # 重新啟動
```

06

6. 更新到新版

收到新的下載包時:

1. 把新 zip **解壓到同一個資料夾覆蓋**(你的 `.env` 會被保留,腳本不會覆蓋已存在的 `.env`)。
2. 再執行一次 `./install.sh`。

你的歷史資料(設定快照、登入帳號密碼)都存在名為 `opta-local-data` 的 Docker volume,更新**不會**遺失。

07

7. 重設密碼(忘記密碼 / 進不去)

地端伺服器是單機部署,沒有「忘記密碼寄信」機制。若進不去,用下列方式重來一次乾淨的初始化:

> ⚠️ 下列做法會**清掉伺服器上的設備設定快照歷史**(不影響設備本身運作,設備下次心跳會重新上報)。若你的快照很重要,請先聯絡 MetaSource 協助。

```bash
# 1. 停止並移除資料 volume
docker compose down
docker volume rm opta-local-data

# 2. 確認 .env 裡 DEFAULT_ADMIN_PASSWORD 設成你要的新密碼
#    (用文字編輯器打開 .env 改這一行)

# 3. 重新啟動 → 會用新密碼重新建立帳號
docker compose up -d
```

之後用 `local@localhost` + 你新設的密碼登入。

08

8. 常見問題排錯

症狀	可能原因 / 解法
網頁打不開	① 服務沒起來 → `docker compose ps` 看狀態、`docker compose logs -f` 看錯誤。② 從別台連要用**伺服器 IP** 不是 `localhost`。③ 防火牆要放行 **8888** 埠。
登入一直失敗	帳號是 `local@localhost`(含 `@localhost`);密碼是 `.env` 的 `DEFAULT_ADMIN_PASSWORD`。若已改過密碼,用改過的。真的進不去走 §7。
設備沒出現在清單	設備要設定把**這台伺服器的位址**當成備份目標,且和伺服器在**同一網段**。確認設備端設定。
8888 埠被占用	編輯 `.env` 的 `HOST_PORT`(例如改 `9000`),`docker compose up -d` 重啟;之後用新埠號連。
`Exec format error`	下載包的 CPU 架構和你的機器不符。請向 MetaSource 索取對應架構的版本。

09

9. 安全建議

- 第一次登入**立刻改密碼**。
- 把 `.env` 的 `JWT_SECRET` 改成一串隨機字串(`openssl rand -hex 32`)。
- 這台機器只在**內網**提供服務,不要對公網開放 8888 埠。

需要協助請聯絡 **MetaSource**。

03 管理者

備援與韌性

📅 最後更新：2026-05-26 ｜ 🛠️ 對應韌體：v5.9.130+ ｜ 📌 負責人：KC

本手冊整理 Opta（MES Gateway）的四項備援機制、各自的驗證結果，以及 MQTT subscribe 失效的外部 watchdog 解法與部署方式。

01

1. 備援機制總覽

#	機制	規則	狀態（2026-05-26 實測）
A	NVS 設定持久化	LocalConfig save/load，跨重開不丟	✅ PASS
B	Config Server 雙層 fallback	單一心跳，地端優先 → 失敗 fallback 官方 cloud	✅ PASS
C	MQTT 斷線重連	broker 斷線無 auto-disable，持續 retry	⚠️ 連線復原但 **subscribe 失效** → 外部 watchdog 解
D	MQTT broker 空 fallback	broker 設定空 → 連官方 `mosquitto.smms.com.tw`	✅ 韌體已實作

02

2. A — NVS 設定持久化

所有設定（signals / actions / rules / tcpio / mqtt / configserver）存於 QSPI Flash，由 `LocalConfig` 序列化。

驗證：軟重開（`POST /api/system/reboot`）後 uptime 657s → 35s，所有設定逐項比對完全一致。重開機、斷電重插皆保留。

03

3. B — Config Server 雙層 Fallback

層級	URL	職責
官方 Server	韌體寫死 `https://opta.smms.com.tw`	授權 + OTA + 備份還原
地端 Server	使用者設定（可空）	備份還原

規則：維護**單一心跳**，目的地優先地端，地端不存在/失敗時 fallback 官方。授權與 OTA **只走官方**。

驗證：地端 URL 設假（`192.168.99.99:9999`）+ 官方留 `opta.smms.com.tw` → 心跳 60s 後 `cloud result=OK`，local 不嘗試 → fallback 正確。

設定 API：`GET/POST /api/configserver`（見 `../api/api-config-server.md`）。

04

4. C — MQTT Subscribe 失效（Known Issue + Watchdog 解法）

4.1 問題

broker 重啟 / 網路抖動後：

- `mqttClient.connected()` 回 `true`，gateway **publish 正常**（遙測還在發）
- 但 **subscribe 失效**：收不到任何 inbound（連自己 publish 給自己訂的 topic 的 echo 也收不到，**對稱性全死**）
- 結果：MQTT 命令 / wizard chain 全失靈，**直到設備 reboot 才復原**

4.2 已嘗試的 in-device 修法（皆失敗）

嘗試	結果
connected() false→true 邊緣偵測 → resubscribe	❌
每 30s 無條件 refresh subscribe	❌
healthcheck → disconnect 強制重連	❌
被動 lastInboundMs > 120s → reboot	❌ 沒觸發
主動 self-ping > 2 次無 echo → reboot	❌ 沒觸發

> 根因疑為 MQTT lib（mbed/pubsubclient）連線狀態機在 broker 重啟後的內部狀態，無 serial debug 無法定位。in-device 自我診斷已證實不可靠。**結論：設備無法可靠自我偵測此狀態，需外部探測。**

4.3 解法：外部 Watchdog（✅ 已驗證）

從設備**外部**定期測 round-trip，連續失敗就打 reboot。腳本：`scripts/opta_watchdog.py`（部署位置見 §4.5）。

原理：

1. publish nonce 到 `…/cmd/signal/5`（未使用的 MQTT 輸入 index）
2. 輪詢 `GET /api/signals`，看 `wdprobe` signal 的 `currentValue` 是否反映
3. 連續 N 次（預設 3）沒反映 = subscribe 死 → `POST /api/system/reboot`
4. 等設備 + MQTT 復原，重置計數

> 為什麼外部有效：它測的是**真正壞掉的 external→device 路徑**，不依賴設備自我診斷。HTTP 不通時不計入失敗（避開設備重開中誤判）。

前置：設備上需有 `wdprobe` signal（隨 NVS 持久化）：

```JSON
POST /api/signals
{"index":4,"enabled":true,"name":"wdprobe","trigger":0,"debounceMs":0,
 "sources":[{"type":6,"index":5,"op":4,"threshold":0.5,"and":false,"expIndex":0,
```

（slot4、CH_MQTT idx5、TRIG_ALWAYS、op≥0.5；不綁任何 rule，純供探測讀取。）

4.4 驗證紀錄（2026-05-26）

```
18:28:44  探測失敗 1/3   ← 停 broker 25s + 重啟後 subscribe 死
18:28:57  探測失敗 2/3
18:29:10  探測失敗 3/3
18:29:10  → POST /api/system/reboot
18:30:07  設備已復原 (~50s)
之後：adv1 → rules/2/trigger 校正標零 → wizard chain 自動回來 ✅
```

4.5 部署

腳本已收錄於 `scripts/opta_watchdog.py`。

```BASH
# 1. 確認 wdprobe signal 已建在設備上 (隨 NVS 持久化,一次即可,見 §4.3)

# 2. 常駐 (擇一):
#   a) 前景測試:
python3 scripts/opta_watchdog.py --interval 30 --fails 3
#   b) 部署到 always-on 主機 + macOS launchd / Linux systemd 常駐 (建議)
#      例: cp scripts/opta_watchdog.py /opt/opta/ 後設 systemd unit
```

參數建議(prod)：`--interval 30 --fails 3` → 約 90s+ 持續失敗才 reboot，濾掉瞬斷。

腳本內需設定：`IP`、`PRE`(UID 前綴)、`BROKER`(host/port/user/pass)。

> 注意：watchdog 應跑在**獨立 always-on 主機**(非設備本身)，且能同時連到 broker 與設備 HTTP。

05

5. D — MQTT Broker 空 Fallback

韌體已實作(`main.cpp` MQTT init + `ApiHandlers_Core.cpp` config POST)：

```
broker 設定非空 → 連地端 broker
broker 設定空   → 連官方 mosquitto.smms.com.tw (port 依 useTls 8883/1883)
```

官方 broker reachable(DNS 210.241.233.133，1883/8883 皆通)。實測 broker 清空後連不上，原因為**官方 broker 需要對應 credentials(user/pass)**——這是部署端設定問題，非韌體 bug。若要真正啟用官方 fallback，需設備帶官方 broker 的帳密(可由 license 服務派發)。

06

6. 故障排除速查

症狀	可能原因	處理
MQTT 命令沒反應、wizard chain 不動，但設備 web 正常	broker 重啟後 subscribe 失效（C）	`POST /api/system/reboot`；長期靠 watchdog 自動處理
設定重開後消失	（不應發生，A 已驗證）	檢查 QSPI 掛載 `qspiMounted`
心跳一直失敗	地端 URL 不通且官方需 credentials	檢查 configserver URL / token
「授權檢查失敗」modal 偶現	設備忙時 license API 逾時	v5.9.130+ 已放寬 timeout + auto-retry，通常自動消失

07

7. 相關文件

- 工作流引擎 / API：`../specs/spec-mqtt-chain-workflow.md`
- Config Server：`../api/api-config-server.md`
- MQTTS / broker：`spec-connectivity.md`、`mqtt-broker` SOP
- 單執行緒 HTTP 競爭（C 根因背景）：Opta 韌體 HTTP 處理為單執行緒，當輪詢與 inbound 請求並發時會互相競爭，這是情境 C（MQTT 重連失效）的根因背景。

04 開發者 / 整合

裝置 REST API

> 📅 最後更新：2026-06-16 ｜ 🛠️ 對應韌體：v5.9.249 ｜ 📌 負責人：KC

Opta（MES Gateway）韌體在 TCP 埠 `80` 上以 HTTP/1.1 提供約 **50+ 個 REST endpoint**。所有路由都是在 `src/main.cpp` 的網路 loop 內以 `firstLine.indexOf(...)` 字串比對方式分派；handler 實作分佈於 `ApiHandlers_Core.cpp`、`ApiHandlers_Connectivity.cpp`、`ApiHandlers_Modbus.cpp`、`ApiHandlers_Logic.cpp`、`OtaHandler.cpp`。

v5.1 起為 **Unified Network Architecture**——同一份韌體同時支援乙太網路與 WiFi。v5.6 整合 TCP IO Channel 與非同步 RS485 掃描。v5.9.125+ 新增 MQTTS（`useTls` / `caCert`）支援。

01

認證機制

韌體採用 Token-based RBAC。Token 由 `POST /api/tokens` 建立，可透過下列任一方式攜帶：

- **HTTP Header**：`Authorization: Bearer <token>`
- **Query 參數**：`?token=<token>`

僅 `/api/system` 在 dispatcher 層強制檢查（v5.9 之後因 login modal 用此 endpoint 驗證 token），其餘 endpoint 由各自 handler 內呼叫 `checkAuth(req, PERM_*)` 把關，未通過則回 `401 Unauthorized` JSON：

```json
{"error": "Unauthorized"}
```

權限等級

等級	數值	可做的事
`PERM_READONLY`	0	只讀的狀態查詢（IO、tokens 列表、config server）。
`PERM_CONTROL`	1	現場控制：DO 切換、Modbus write、converter 預覽。
`PERM_ADMIN`	2	設定變更：config、rules、tokens、license、OTA、factory reset。

`-` 表示 dispatcher 直接放行（如 `/api/io`、`/api/config` GET、`/`、`/favicon.ico` 等）。

02

Endpoint 總覽

系統 / 診斷

Endpoint	Method	Permission	說明
`/`	`GET`	-	主控台 HTML（QSPI 或內嵌 fallback）
`/favicon.ico`	`GET`	-	favicon
`/api/system`	`GET`	`READONLY`	系統健康狀態（RAM/Flash/uptime/MAC/license）
`/api/system/reboot`	`POST`	`ADMIN`	延遲 reboot（先回 200 再 reset）
`/api/system/factory-reset`	`POST`	`ADMIN`	Factory reset（清除設定）
`/api/poll`	`GET`	-	合併 system / io / virtual / stats 的 bulk JSON
`/api/config/integrity`	`GET`	`READONLY`	Config checksum / 完整性驗證

設定

Endpoint	Method	Permission	說明
`/api/config`	`GET`	-	取得完整節點設定
`/api/config`	`POST`	`ADMIN`	更新節點設定（partial update）
`/api/io-name`	`POST`	`ADMIN`	命名單一 IO 通道
`/api/time/sync`	`POST`	`ADMIN`	從瀏覽器同步 RTC（WI-079）

I/O

Endpoint	Method	Permission	說明
`/api/io`	`GET`	-	DI / DO / AI / 擴充模組即時狀態
`/api/io/do`	`POST`	`CONTROL`	切換 DO（單顆或多顆）
`/api/expansions`	`GET`	-	擴充模組列表（型號、IO 數）
`/api/expansions/name`	`POST`	`ADMIN`	命名擴充模組
`/api/analytics`	`GET`	-	Sparkline 用 ring buffer 樣本

規則 / 訊號 / 動作 / 虛擬通道

Endpoint	Method	Permission	說明
`/api/rules`	`GET`	-	列出所有 trigger rule
`/api/rules`	`POST`	`ADMIN`	新增 / 修改 / 刪除 rule（依 body）
`/api/rules/clear`	`POST`	`ADMIN`	清空所有 rule（WI-103 FR-H1-02）
`/api/signals`	`GET / POST`	`ADMIN` *	Sensor Group（📡）定義
`/api/actions`	`GET / POST`	`ADMIN` *	Actuator Group（⚙️）序列
`/api/virtual`	`GET`	-	虛擬通道狀態
`/api/virtual`	`POST`	`ADMIN`	更新虛擬通道
`/api/history`	`GET`	-	最近 100 筆事件

> *GET 不檢查 auth，POST 需 `ADMIN`。

Modbus / RS485

Endpoint	Method	Permission	說明
`/api/modbus`	`GET`	-	列出 Modbus 裝置與暫存器
`/api/modbus`	`POST`	`ADMIN`	批次更新 Modbus 設定
`/api/modbus-devices`	`POST`	`ADMIN`	新增 / 更新單顆 RS485 device
`/api/modbus-devices`	`DELETE`	`ADMIN`	刪除 RS485 device
`/api/modbus/scan`	`GET` / `POST` / `DELETE`	`ADMIN`	RS485 非同步掃描（POST 啟動、GET 輪詢、DELETE 取消）
`/api/modbus/write`	`POST`	`CONTROL`	手動寫 Modbus 暫存器（FC06 / FC16）
`/api/modbus/lock`	`POST`	`ADMIN`	鎖匯流排，阻止 polling（用於手動 IO）
`/api/modbus/probe`	`POST`	`CONTROL`	Probe 單一裝置
`/api/modbus/sync-baud`	`POST`	`ADMIN`	跟 slave 同步鮑率
`/api/rs485/{busId}/.../write`	`POST`	`CONTROL`	對指定 RS485 device 寫入（WI-118）
`/api/modbus-tcp/targets`	`GET`	-	Modbus TCP 遠端讀取設定
`/api/modbus-tcp/targets`	`POST`	`ADMIN`	更新 Modbus TCP targets
`/api/modbus-tcp/write`	`POST`	`CONTROL`	寫入遠端 Modbus TCP

TCP IO / Converter（轉換器）

Endpoint	Method	Permission	說明
`/api/tcpio`	`GET`	-	列出 TCP IO 通道
`/api/tcpio`	`POST`	`ADMIN`	新增 / 更新
`/api/tcpio`	`DELETE`	`ADMIN`	刪除
`/api/tcpio/clear`	`POST`	`ADMIN`	清空所有（WI-103 FR-H1-02）
`/api/converter/template`	`GET / POST`	`ADMIN`*	Outbound assembly template
`/api/converter/preview`	`POST`	`CONTROL`	Template 變數展開預覽

連線 / MQTT / WiFi

Endpoint	Method	Permission	說明
`/api/mqtt/test`	`POST`	`ADMIN`	測試 MQTT broker 連線並發測試訊息
`/api/mqtt/publish`	`POST`	`CONTROL`	通用 MQTT 發佈（工作流程「觸發」用，WI-128）。body `{"topic","payload","qos"?}`；topic 限本機 prefix 命名空間、禁萬用字元、payload ≤127B
`/api/wifi/status`	`GET`	-	目前 WiFi 連線狀態
`/api/wifi/scan`	`GET`	-	掃描可用 SSID
`/api/wifi/connect`	`POST`	`ADMIN`	連線指定 SSID
`/api/wifi/disconnect`	`POST`	`ADMIN`	中斷 WiFi
`/api/wifi/save`	`POST`	`ADMIN`	儲存憑證至 Flash（provisioning）
`/api/network/apmode`	`POST`	`ADMIN`	切換到 AP（soft AP）模式重開機

> WiFi 與 `/api/network/apmode` 僅在編譯時定義 `USE_WIFI` 才會啟用。

雲端 / Config Server

Endpoint	Method	Permission	說明
`/api/configserver`	`GET`	`READONLY`	取得 cloud 同步設定（URL / token / enabled）
`/api/configserver`	`POST`	`ADMIN`	更新 cloud 設定
`/api/configserver/test`	`POST`	`ADMIN`	觸發非同步連線測試
`/api/configserver/test`	`GET`	`READONLY`	輪詢測試結果
`/api/configserver/push`	`POST`	`ADMIN`	強制立刻推送 config 到雲
`/api/debug/configserver`	`GET`	`READONLY`	上述 GET 的 legacy alias
`/api/telemetry-token`	`GET`	`READONLY`	回快取的 scoped telemetry token；無 / 過舊則觸發主 loop 向雲端換取並回 `202 fetching`（ADR-017 高頻遙測）

Token 與認證

Endpoint	Method	Permission	說明
`/api/tokens`	`GET`	`READONLY`	列出 token（已遮罩 secret）
`/api/tokens`	`POST`	`ADMIN`	建立新 token（WI-112，僅此回傳明文）
`/api/tokens`	`DELETE`	`ADMIN`	撤銷 token

統計與電力

Endpoint	Method	Permission	說明
`/api/stats`	`GET`	-	生產計數器與 uptime（/fs/stats.bin 持久化）
`/api/stats/reset`	`POST`	`ADMIN`	重置 IO 統計並刪除 /fs/stats.bin
`/api/power`	`GET`	-	PowerMonitor 電力資料

License

Endpoint	Method	Permission	說明
`/api/license`	`GET`	-	License 狀態與授權功能
`/api/license/activate`	`POST`	`ADMIN`	申請授權（async，由雲端 heartbeat 套用 verdict）
`/api/license/renew`	`POST`	`ADMIN`	續約
`/api/license/reset`	`POST`	`ADMIN`	清空所有 license 狀態（WI-103 FR-H1-01）
`/api/license/token`	`POST`	`ADMIN`	灌入 base64 簽章 token → 驗簽+比 UID+比到期 → 存 `/cfg/license.tok`（WI-155 瀏覽器 inbound 授權，救 outbound/MQTT 壞的設備如 8310）
`/api/license/token`	`GET`	-	讀回 `/cfg/license.tok` 驗證回報（`{present:false}` 或 token 資訊）
`/api/license/enforce`	`POST`	`ADMIN`	切換 enforcement 旗標 `{"on":true|false}`（WI-145 rollout/測試）

OTA（空中更新）

Endpoint	Method	Permission	說明
`/api/ota/versions`	`POST`	`ADMIN`	觸發向雲端拉取版本列表（async）
`/api/ota/versions`	`GET`	-	輪詢列表結果
`/api/ota/status`	`GET`	-	目前版本 + 是否有可用更新
`/api/ota/start`	`GET`	-	直接啟動下載最近 heartbeat 廣播的更新
`/api/ota/trigger`	`POST`	`ADMIN`	用 body 指定 URL / version 觸發 OTA
`/api/ota/upload`	`POST`	`ADMIN`	直接上傳 raw firmware `.bin`（串流；偵測到 `.mesb` 會拒絕，要走 `/api/ota/bundle`）
`/api/ota/bundle`	`POST`	`ADMIN`	上傳 `.mesb` **單一檔**（firmware + 全部 web 資產，串流）→ `handleMesbUpload`：fw→QSPI P2、資產→P4，重開。拓荒/發佈核心(WI-125)
`/api/upload`	`POST`	`ADMIN`	上傳 web UI 至 QSPI（multipart 串流）
`/api/web?name=<檔>`	`POST`	`ADMIN`	串流上傳單一 web 資產到 QSPI `/cfg/web/<檔>`（CSS/語系/app.js，免重燒迭代 UI，WI-124）

Debug

Endpoint	Method	Permission	說明
`/api/debug/tcptest`	`GET`	-	測試 EthernetClient.connect() 兩種寫法
`/api/debug/exp`	`GET`	-	Dump `expDiCache` 與 signal sources

03

詳細範例

系統健康狀態 — `GET /api/system`

> 此 endpoint 是前端 heartbeat / login 驗證用。**從 v5.9（WI-115k）起 dispatcher 強制 `PERM_READONLY`**——login modal 直接拿這個端點驗 token 是否合法，否則先前任意密碼都回 200 會被誤判成功。

回應主要欄位：

```json
{
  "apMode": false,
  "version": "5.9.130",
  "board": "Arduino Opta",
  "uptime": 123456,
  "memory": {
    "totalRAM": 523624,
    "freeRAM": 453880,
    "usedRAM": 69744,
    "usagePercent": 13.4
  },
  "flash": {
    "totalFlash": 786432,
    "usedFlash": 781000,
    "qspiFlash": 16777216
  },
  "mac": "AA:BB:CC:DD:EE:FF",
  "licensed": true,
  "licensedFeatures": ["rule_engine", "mqtt", "tcp_io"],
  "bootTries": 1,
  "systemStatus": "OK"
}
```

> ⚠️ `flash.usedFlash` 接近 `786432`（partition 上限）時要警覺——v5.9.113 之後 build 已穩定 ≥ 99.3% 使用率，新功能加進來前要先評估 binary size，否則 OTA 會被 `/api/ota/trigger` 的 `413 Payload Too Large` 擋下（見 WI-115g）。

`apMode = true` 是前端切換到 **Provisioning Wizard** 的主要 trigger。

設定 — `GET /api/config`

完整節點設定 dump，主要 block：

```json
{
  "firmwareVersion": "5.9.130",
  "deviceName": "opta-line-3",
  "mac": "AA:BB:CC:DD:EE:FF",
  "network": {
    "ethDhcp": true,
    "ethIp": "192.168.1.100",
    "ethGateway": "192.168.1.1",
    "ethSubnet": "255.255.255.0",
    "wifiDhcp": true,
    "wifiIp": "...",
    "wifiGateway": "...",
    "wifiSubnet": "...",
    "interfaceMode": 0
  },
  "wifi": { "ssid": "MyAP", "passwordLen": 16 },
  "tcp": { "port": 80 },
  "mqtt": {
    "enabled": true,
    "broker": "mqtt.example.com",
    "port": 8883,
    "clientId": "opta-abc",
    "username": "device",
    "password": "********",
    "topicPrefix": "factory/lineA/",
    "publishPeriodicIo": true,
    "useTls": true,
    "caCert": "-----BEGIN CERTIFICATE-----\nMIIDxz...==\n-----END CERTIFICATE----
  },
  "rs485": { "baudRate": 9600, "masterMode": true, "slaveId": 1 },
  "inputMode": [0, 0, 0, 0, 0, 0, 0, 0],
  "inputOp":   [0, 0, 0, 0, 0, 0, 0, 0],
  "inputThreshold": [3.3, 3.3, 3.3, 3.3, 3.3, 3.3, 3.3, 3.3],
```

```
  "inputNames": ["DI1", "DI2", "..."],
  "doNames":    ["DO1", "DO2", "DO3", "DO4"],
  "expansions": [...],
  "qspiMounted": true,
  "modbusDevices":   [...],
  "modbusRegisters": [...],
  "rs485Buses":      [...]
}
```

> **v5.9.125+**：MQTT block 新增 `useTls` 與 `caCert`。`caCert` 在 wire format 上把實際換行字元轉成字面 `\n`，前端送回時必須維持 JSON-escape 形式。

設定 — `POST /api/config`

需 `PERM_ADMIN`。**只送要改的欄位**，handler 內以 `indexOf("\"key\":value")` 字串比對；JSON 必須是 **compact 格式（key/value 之間沒有空格）**，否則會靜默失敗。

```JSON
{
  "deviceName": "opta-line-3",
  "mqtt": {
    "enabled": true,
    "broker": "mqtt.example.com",
    "port": 8883,
    "useTls": true,
    "caCert": "-----BEGIN CERTIFICATE-----\\n...\\n-----END CERTIFICATE-----\\n"
  }
}
```

回應：

- `200 OK`：`{"success": true, "message": "Config updated and saved"}`

- `400 Bad Request`：`{"error": "No valid fields to update"}`
- `401 Unauthorized`：`{"error": "Unauthorized"}`

> 寫入時透過 `NetworkStateManager::requestSafeSave()` 延後到 WiFi state machine 安全空檔，避免跟 supplicant transition 衝突。MQTT 相關欄位改動會自動重連 broker。

切換數位輸出 — `POST /api/io/do`

需 `PERM_CONTROL`。

```JSON
{ "index": 0, "value": true }
```

或同時設定多顆（舊版）：

```JSON
{ "values": [true, false, false, true] }
```

> Pulse 模式（`pulse: true, duration: <ms>`）目前 handler 預留但尚未實作。

回應：

- `200 OK`：`{"success": true, "index": 0, "value": true}`
- `400 Bad Request`：index 越界 `{"error": "invalid index"}`

Modbus 掃描 — `POST /api/modbus/scan`

需 `PERM_ADMIN`。RS485 掃描是 **非同步** 的：POST 啟動 → 持續 GET 輪詢結果 → 必要時 DELETE 取消。

```bash
# 啟動
curl -X POST -H "Authorization: Bearer $TOKEN" \
     http://opta.local/api/modbus/scan -d '{"busId":0}'
# → 202 {"status":"scanning"}

# 輪詢
curl -H "Authorization: Bearer $TOKEN" http://opta.local/api/modbus/scan
# 完成：{"devices":[{"slaveId":1,"name":"TDA-08B","registers":[...]}]}

# 取消
curl -X DELETE -H "Authorization: Bearer $TOKEN" http://opta.local/api/modbus/sca
```

掃描期間 RS485 bus 會被自動 lock 阻止 polling，掃完自動釋放。也可手動 `POST /api/modbus/lock` 在做手動寫入時暫鎖匯流排。

MQTT 測試 — `POST /api/mqtt/test`

需 `PERM_ADMIN`。可帶 body 用指定設定測試；省略則用目前設定。

```json
{
  "broker": "mqtt.example.com",
  "port": 8883,
  "username": "test",
  "password": "test",
  "useTls": true,
  "caCert": "-----BEGIN CERTIFICATE-----\\n...\\n"
}
```

回應：

- `200 OK`：`{"success": true, "message": "Test message published"}`
- `400 / 500`：`{"success": false, "error": "..."}`

> **MQTTS 自簽 broker 注意**（v5.9.125~130）：mbedTLS 不認 IP-type SAN，broker cert 必須同時放 `IP:x.x.x.x` 與 `DNS:x.x.x.x`，否則 handshake 會在 verify 階段失敗。

Token 管理

列表（`GET /api/tokens`） — `PERM_READONLY`：

```JSON
[
  { "index": 0, "name": "frontend",  "permission": "ADMIN",    "createdAt": 1737(
  { "index": 1, "name": "factory-rd", "permission": "CONTROL", "createdAt": 1737:
]
```

建立（`POST /api/tokens`） — `PERM_ADMIN`：

```JSON
{ "name": "ops-readonly", "permission": "READONLY" }
```

回應 **僅此一次** 包含明文 token：

```JSON
{ "success": true, "token": "tok_...", "name": "ops-readonly", "permission": "RE/
```

撤銷（`DELETE /api/tokens`） — `PERM_ADMIN`：

```JSON
{ "index": 1 }
```

License 啟用流程

License 採 **device → cloud → device** 三段式：

1. `POST /api/license/activate` → 標記 pending，回 `{"status":"pending"}`
2. 雲端（opta.smms.com.tw）約 30 秒一次 heartbeat 收 verdict
3. `GET /api/license` 取得當下狀態：

```JSON
{
  "status": "active",
  "licenseId": "LIC-2026-xxx",
  "features": ["rule_engine", "mqtt", "tcp_io"],
  "expiryDate": 1764547200,
  "lastKnownTime": 1745000000
}
```

> **重要**：`/api/license` GET handler 不會主動 POST activate，避免跟 server「rejected → pending 重申」邏輯衝突（前端 polling 會把 rejected 推回 pending 永遠跑不出 PENDING 迴圈）。Verdict pull 是 cloud→device，使用者顯式 activate 才是 device→cloud。

OTA 觸發 — `POST /api/ota/trigger`

需 `PERM_ADMIN`。Body 帶 url（必要）與 version（建議，作為 guard）：

```JSON
{ "url": "https://opta.smms.com.tw/firmware/opta_v5.9.131.bin", "version": "5.9.1
```

回應：

HTTP	條件	Body
`200 OK`	URL 有效 + version ≠ current	`{"ok":true,"message":"OTA download started","version":"..."}`
`409 Conflict`	version == FW_VERSION（避免 same-version brick）	`{"error":"Same version","message":"...","current":"...","target":"..."}`
`413 Payload Too Large`	binary > 780,000 bytes	`{"error":"Firmware too large for safe OTA","size":...,"limit":780000}`
`400 Bad Request`	缺 URL	`{"error":"No OTA URL specified"}`

> WI-115c：**拒絕 same-version OTA**——重複 swap 同一份 firmware 會踩到 bootloader fallback 邏輯，被丟去 DFU。
> WI-115g：partition 上限 786432 bytes，留 6KB 緩衝設 **780,000 bytes** 為硬上限。超過請改用 USB DFU。

時間同步 — `POST /api/time/sync`

需 `PERM_ADMIN`，從瀏覽器同步 RTC（WI-079）。

```JSON
{ "timestamp": 1748044800 }
```

回應：

- `200 OK`：`{"status":"success","timestamp":1748044800}`
- `400 Bad Request`：timestamp 比 `lastKnownTime` 還舊會被拒。

非同步 Config Server 測試

`POST /api/configserver/test` 觸發後立刻回 `{"status":"testing"}`；client socket 關閉後韌體才在 `loop()` 跑實際連線（避免佔用 outbound socket）。輪詢 `GET /api/configserver/test`：

```JSON
{
  "status": "done",
  "success": true,
  "latency": 142,
  "version": "v2.2.10",
  "error": ""
}
```

04

批次輪詢 — `GET /api/poll`

合併 `system + io + virtual + stats` 為單一 response，減少前端輪詢成本。內部依序呼叫各 handler 並省略 HTTP header（`printHeader = false`），最後組成單一 JSON 物件，欄位即四個子 handler 的合併。

05

HTTP 狀態碼

Code	意義
`200`	成功
`202`	已接受（async 任務排隊；輪詢另一 endpoint 取結果）
`400`	Bad Request（JSON 解析失敗、必填缺、值越界）
`401`	Unauthorized（token 缺或權限不足）
`404`	Not Found（unknown API path 或 QSPI 檔不存在）
`409`	Conflict（OTA same version）
`413`	Payload Too Large（OTA binary > 780KB）
`500`	Internal Server Error（config 未載入等）

錯誤一律走 JSON：

```JSON
{ "error": "register_not_found", "message": "registerIdx out of range for this d
```

Modbus handler 另有 `sendErr(httpCode, statusText, errorCode, message)` 統一包裝，常見 `errorCode`：

- `register_not_found`
- `register_not_writable`（Register `mode != 1`）
- `device_not_found`
- `bus_locked`

06

近期變更（v5.9.x 重點）

Version	變更
v5.9.130	MQTTS handshake 修正——chain 父類別 `setRootCA` + `threshold=1`，子類 `MqttsTlsDebug` 印 cert source/len 與 mbedTLS debug。
v5.9.125	`/api/config` GET/POST 新增 `mqtt.useTls` 與 `mqtt.caCert`（PEM，wire 上 `\n` escape）。
v5.9.x WI-118	新增 `/api/rs485/{busId}/.../write`——以 bus path 對指定 RS485 device 寫入。
v5.9.x WI-115k	`/api/system` dispatcher 強制 `PERM_READONLY`，修補登入時假成功問題。
v5.9.x WI-115g	OTA 加入 780KB size guard；超過直接 `413` 擋下。
v5.9.x WI-115c	OTA 拒絕 same-version trigger，避免 bootloader fallback。
v5.9.x WI-115f	主控台 HTML 加 ETag（值＝`FW_VERSION`），減少重複下載。
v5.9.x WI-115d	移除 legacy `GET /app.js`，bundle 直接 inline 進 `WebPage.h`。
v5.9.x WI-112	Token 管理：`POST` / `DELETE /api/tokens`。
v5.9.x WI-103	FR-H1-01 `/api/license/reset`、FR-H1-02 `/api/rules/clear` + `/api/tcpio/clear`。
v5.9.x WI-079	`/api/time/sync`：瀏覽器同步 RTC。
v5.9.x WI-085	DO 索引由 0-based → 1-based（DO1–DO4），影響 `/api/io` 回傳與 `/api/io/do` index。

07

實作備註

1. **無 router framework**：所有 endpoint 在 `main.cpp:1657-2071` 以 `else if (firstLine.indexOf("METHOD /path") >= 0)` 串接，**順序決定優先權**——較長的路徑（例如 `/api/configserver/test`）必須排在較短前綴（`/api/configserver`）**之前**。新增 endpoint 時注意這個順序陷阱。
2. **Auth 分佈**：除 `/api/system` 在 dispatcher 即驗，其餘 endpoint 在各 handler 一進入就呼叫 `checkAuth(req, PERM_*)`，失敗回 `401`。
3. **Streaming uploads**：`/api/upload` 與 `/api/ota/upload` 不把 body 載進記憶體，邊收邊寫 QSPI / partition；OTA upload 期間 `localConfig.unmount()` → 寫入 → `remount()`。
4. **Async pattern**：Config server test、OTA version list、Modbus scan、License activate 都用「先回 202/pending → 在 `loop()` 跑工作 → 後續 GET 輪詢結果」模式，避免阻塞單一 client socket。
5. **JSON 寫回**：所有 JSON response 都標 `Content-Type: application/json`、`Connection: close`，多數不帶 `Content-Length`（chunked-style，由 FIN 標 EOF），client 必須讀到 socket 關閉才算完整。
6. **Config POST 字串比對**：handler 用 `indexOf("\"key\":value")`，**JSON 必須 compact 無空格**——Python `json.dumps` 預設帶空格會靜默失敗，務必加 `separators=(",", ":")`。
7. **`USE_WIFI` 編譯切換**：WiFi 與 AP mode 相關 endpoint 用 `#ifdef USE_WIFI` 包住；Ethernet-only build 不會註冊這些路由，request 會回 404。
8. **Content-Length 用 byte 數**：handler 依 `Content-Length` 讀 body，中文 UTF-8 每字 3 bytes；client 若用字元數會截斷 body（POST 回 success 但欄位沒套用）。
9. **保留 MQTT topic `{prefix}/cmd/_ping`**（v5.9.130+）：韌體 self-ping 健康檢查專用，外部請勿佔用。
10. **MQTT-Chain 工作流**：`/api/signals`、`/api/actions`、`/api/rules`、`/api/tcpio`、`/api/converter/*` 可組成有狀態工作流（state 經 MQTT loopback）。完整 schema、欄位、雷區見 `../specs/spec-mqtt-chain-workflow.md`。

08

延伸閱讀

主題	文件
MQTT-Chain 工作流引擎 + API schema	`../specs/spec-mqtt-chain-workflow.md`
TDA08B 校正引導精靈操作手冊	`../guides/guide-calibration-wizard.md`
備援機制 + MQTT Watchdog 部署	`../guides/guide-failover-resilience.md`
Config Server API	`api-config-server.md`

04 開發者 / 整合

Config Server API

> 📅 最後更新：2026-05-24 | 🛠️ 對應版本：v2.2.10 | 📌 負責人：KC
> 🔗 雲端：`https://opta.smms.com.tw`（`SERVER_ROLE=cloud`）| 地端：`http://<SERVER_IP>:8888`（`SERVER_ROLE=local`）

本文件涵蓋 MES I/O Gateway Config Server 的全部 REST API。Server 採 **單一 codebase + 角色模式**，以 `SERVER_ROLE` 環境變數切換：

- `local`：地端備份 / 還原伺服器（檔案系統 + SQLite，無 PG、無 license、無 Cloud Bridge）
- `cloud`：雲端授權 + OTA 派送（**PostgreSQL 必要**，啟用 license 模組與 Cloud Bridge）

標 ☁️ 的端點僅 `SERVER_ROLE=cloud` 時可用。

01

認證機制

`middleware/auth.js` 採三軌制：

順序	Token 型態	來源	注入 `req.user`
1	**JWT**	`POST /api/auth/login` 簽發（12h 有效）	`{ id, email, role, displayName }`
2	**Legacy DEVICE_TOKEN**	環境變數 `DEVICE_TOKEN`（預設 `admin-token`）	`{ role: 'device', id: 0 }`
3	**License Token**	雲端 license 模組簽發給已啟用 device	`{ role: 'device', uid }`

呼叫方式：

```
Authorization: Bearer <token>
```

未帶 Bearer header → `401 Unauthorized {"error": "Unauthorized", "message": "Missing Authorization header"}`。失敗事件會打 `auth.reject` log，使用者回報 401/403 時用 `grep auth.reject` 立刻看到原因。

角色 (`role`)

Role	來源	權限
`admin`	DB user role	所有 `/api/admin/*`、user/license/firmware 管理
`local_admin`	DB user role	等同 admin 但限地端
`customer`	DB user role	Tenant-scoped：`/api/devices` 只看自己擁有的
`device`	Legacy / License Token	Device 端 heartbeat、config sync、OTA fetch

`adminOnly` middleware 額外把關到 `role ∈ {admin, local_admin}`。

02

端點總覽

共用 API（local + cloud）

#	Method	Path	認證	呼叫方	用途
0	`GET`	`/api/health`	Public	All	健康檢查（含 role / version / device 數）
1	`POST`	`/api/auth/login`	Public	User	登入取得 JWT
2	`GET`	`/api/auth/me`	JWT	User	查目前登入者
3	`POST`	`/api/auth/password`	JWT (非 device)	User	自助改密碼
4	`POST`	`/api/devices/claim/generate`	JWT	User Dashboard	產生 6 位數 OTP 綁定 device
5	`POST`	`/api/devices/claim`	Public	Device	Device 用 OTP 完成綁定
6	`POST`	`/api/devices/:id/heartbeat`	Optional	Device	Device 心跳（含 firmware/license/OTA 回路）
7	`GET`	`/api/devices`	Required	Dashboard	列出所有設備（tenant filter）
8	`DELETE`	`/api/devices/:id`	Required	Dashboard	刪除設備（記憶體）
9	`POST`	`/api/admin/devices/cleanup-offline`	Admin	Dashboard	批次刪除離線設備
10	`GET`	`/api/devices/:id/config`	Required	Device	下載設備設定
11	`POST`	`/api/devices/:id/config`	Required	Device	上傳設備設定（自動備份 + snapshot）
12	`GET`	`/api/devices/:id/backups`	Required	Dashboard	Backup 清單（含 diff summary）
13	`GET`	`/api/devices/:id/backups/:filename`	Required	Dashboard	取單一 backup 全文
14	`GET`	`/api/devices/:id/diff`	Required	Dashboard	Backup vs current diff
15	`POST`	`/api/devices/:id/restore`	Required	Dashboard	還原至 backup
16	`GET`	`/api/devices/:id/snapshots`	Required	Dashboard	自動 snapshot 列表（最近 50 / 30 天）
17	`GET`	`/api/devices/:id/snapshots/:filename`	Required	Dashboard	取單一 snapshot

#	Method	Path	認證	呼叫方	用途
18	`POST`	`/api/devices/:id/snapshots/:filename/rollback`	Required	Dashboard	Rollback 到 snapshot
19	`POST`	`/api/devices/:id/web`	Required	Device	上傳 web bundle（CRC 比對省流量）
20	`GET`	`/api/devices/:id/web/crc`	Required	Device	查 web bundle CRC
21	`GET`	`/api/firmware/latest`	Public	Device	依 channel 查最新韌體（auto-update）
22	`GET`	`/api/firmware/download/:filename`	Public	Device	下載韌體 `.bin`
23	`POST`	`/api/firmware/upload`	Admin	Dashboard	上傳韌體
24	`GET`	`/api/firmware`	Admin	Dashboard	列出韌體（含每版本在線 device 數）
25	`GET`	`/api/firmware/:filename/devices`	Admin	Dashboard	看哪些 device 在跑這版
26	`GET`	`/api/firmware/:filename/channel-history`	Admin	Dashboard	Promote/demote 歷史
27	`POST`	`/api/admin/firmware/:filename/channel`	Admin	Dashboard	改 channel（dev/beta/stable）— nginx 不擋
28	`PATCH`	`/api/admin/firmware/:filename/channel`	Admin	Dashboard	同上
29	`DELETE`	`/api/admin/firmware/:filename`	Admin	Dashboard	刪韌體（擋 stable，可 `?force=1`）
30	`POST`	`/api/firmware/deploy/:deviceId`	Admin	Dashboard	Queue OTA 給單一 device
31	`POST`	`/api/devices/:id/ota` *(deprecated)*	Optional	-	舊 `.mesb` 上傳 → 一律 `410 Gone`
32	`GET`	`/api/devices/:id/ota/latest` *(deprecated)*	Optional	-	舊 `.mesb` 查詢 → 一律 `410 Gone`
33	`GET`	`/api/customers`	Required	Dashboard	客戶列表（含 device/online 計數）
34	`GET`	`/api/customers/:id`	Required	Dashboard	客戶詳細
35	`GET`	`/api/customers/:id/devices`	Required	Dashboard	該客戶旗下設備
36	`POST`	`/api/customers`	Required	Dashboard	建立客戶
37	`PATCH`	`/api/customers/:id`	Required	Dashboard	修改客戶

#	Method	Path	認證	呼叫方	用途
38	`DELETE`	`/api/customers/:id`	Required	Dashboard	刪除客戶（device 改未指派而非刪除）
39	`POST`	`/api/devices/:uid/assign-customer`	Required	Dashboard	指派 / 解除 device 對客戶歸屬
40	`GET`	`/api/admin/users`	Admin	Dashboard	使用者列表
41	`POST`	`/api/admin/users`	Admin	Dashboard	新增使用者
42	`PUT`	`/api/admin/users/:id`	Admin	Dashboard	修改使用者
43	`PUT`	`/api/admin/users/:id/password`	Admin	Dashboard	Admin 重置別人密碼
44	`GET`	`/api/audit-logs`	Admin	Dashboard	稽核日誌（分頁）
45	`GET`	`/api/admin/backup-status`	Admin	Dashboard	pg_dump 健康度（v2.2.6+）
46	`GET`	`/api/cloud/status`	Public	Dashboard	Cloud Bridge 狀態（local 模式回 disabled）

☁️ Cloud-only

僅當 `SERVER_ROLE=cloud` 時掛載（`server.js:302-309`）：

#	Method	Path	認證	說明
C1	`POST`	`/api/license/activate`	Public	Device 申請啟用（whitelist 直接 active）
C2	`GET`	`/api/license/status/:uid`	Public	Device 查狀態
C3	`GET`	`/api/admin/license/list`	Admin	全部 license（含統計）
C4	`POST`	`/api/admin/license/import`	Admin	批次匯入 UID 白名單
C5	`POST`	`/api/admin/license/approve/:uid`	Admin	審核通過（pending → active）
C6	`POST`	`/api/admin/license/reject/:uid`	Admin	拒絕（device 可重新申請回 pending）
C7	`POST`	`/api/admin/license/deactivate/:uid`	Admin	強制停權（active → rejected，無 grace period）
C8	`DELETE`	`/api/admin/license/:uid`	Admin	硬刪除（寫 tombstone，90 天不可重新 activate）
C9	`DELETE`	`/api/admin/license-revoked/:uid`	Admin	清掉 tombstone（讓被刪 UID 可重新 activate）

03

0. 健康檢查

```
GET /api/health
```

`200 OK`：

```JSON
{
  "status": "ok",
  "role": "cloud",
  "version": "2.2.10",
  "uptime": 3600,
  "timestamp": "2026-05-24T06:00:00.000Z",
  "deviceCount": 12
}
```

```BASH
curl https://opta.smms.com.tw/api/health
```

04

1. Auth

`POST /api/auth/login`

```JSON
{ "email": "kowei.chen@gmail.com", "password": "..." }
```

`200 OK`：

```JSON
{
  "ok": true,
  "token": "eyJhbGciOi...",
  "user": { "id": 1, "email": "kowei.chen@gmail.com", "role": "admin", "displayName": "KC" }
}
```

`401`：`{"error": "Unauthorized", "message": "invalid credentials"}`

JWT TTL **12 小時**，逾期後須重新 login。

`GET /api/auth/me`

回目前 JWT 對應的 user 物件。供前端 reload 後重新水合 user state。

`POST /api/auth/password`

```json
{ "currentPassword": "old", "newPassword": "new12345" }
```

`newPassword` 至少 8 字元。Device token 一律 `403`（不准改 `device` user）。

05

2. Device Claim

OTP-based 雙向綁定（Sprint 21 WI-070）：

`POST /api/devices/claim/generate`（user 端）

需 JWT。產生 6 位數 OTP，10 分鐘有效。

`200 OK`：

```json
{
  "ok": true,
  "claimToken": "493217",
  "expiresAt": "2026-05-24T06:10:00.000Z",
  "instruction": "在設備頁面輸入此 6 位數驗證碼以完成綁定"
}
```

`POST /api/devices/claim`（device 端）

```json
{ "deviceId": "opta-AA-BB-CC", "claimToken": "493217" }
```

`200 OK`：`{"ok": true, "message": "Device claimed", "deviceId": "...", "userId": 1}`
`410`：OTP 失效

06

3. Heartbeat

`POST /api/devices/:id/heartbeat`

最熱門 endpoint。Device 約 30 秒一次回報；server 回 OTA / license verdict / 待派送 web upload。

掛載於 auth middleware **之前** → token 為 optional（若帶會解，缺也不擋）。

Request body：

```JSON
{
  "deviceName": "opta-line-3",
  "ip": "192.168.0.100",
  "uptime": 123456,
  "configVersion": 17,
  "uid": "AA:BB:CC:DD:EE:FF",
  "firmwareVersion": "5.9.130",
  "firmwareVariant": "standard",
  "otaProgress": null
}
```

Response `200 OK`：

```JSON
{
  "status": "ok",
  "license": { "status": "active", "licenseId": "LIC-2026-..." },
  "ota": {
    "url": "https://opta.smms.com.tw/api/firmware/download/opta_v5.9.131.bin",
    "version": "5.9.131",
    "force": false
  }
}
```

可能的 `status`：

- `ok` — 一切正常
- `ota_available` — 有更新可拉
- `download_update` — 後端要求立刻拉 config + web（例如 restore / rollback 完成）

Side effects：

1. **Persist 至 PG `devices` 表（uid PK）**（cloud 模式；v2.2.10 dff2c2d）
2. 若 `configVersion` 變化 → 觸發 auto-snapshot（source=`auto-heartbeat`）

3. 派送 pending OTA / web upload 給 device
4. 呼叫 `cloud-bridge.onHeartbeat`（Arduino Cloud 同步）
5. License upsert：未見過的 UID 會建 `unseen` 紀錄

07

4. Devices

`GET /api/devices`

Auth required。Cloud 模式回傳 **PG `devices` 表 + in-memory map 的 union（uid 去重）**；Local 模式 fallback 只看 in-memory。

`customer` role 會被 tenant filter（只看自己擁有的 device IDs）。

JSON

```
[
  {
    "id": "opta-AA-BB-CC",
    "uid": "AA:BB:CC:DD:EE:FF",
    "deviceName": "opta-line-3",
    "ip": "192.168.0.100",
    "firmwareVersion": "5.9.130",
    "firmwareVariant": "standard",
    "lastSeen": "2026-05-24T05:59:30.000Z",
    "uptime": 123456,
    "configVersion": 17,
    "otaProgress": null,
    "pendingCommand": null,
    "registered": true,
    "status": "online",
    "customerId": 5,
    "customerName": "Foxconn Tier-1"
  }
]
```

> **v2.2.10（commit dff2c2d）**：原本 `/api/devices` 用 in-memory `deviceId` 為 key，導致同樣 `deviceName` 但不同 UID 的 device 互蓋。改用 PG 的 `uid` PK 後，每顆 Opta 都有唯一 row；舊資料沒 uid 者標 `registered: false`。

`DELETE /api/devices/:id`

刪除 in-memory map 上的 device（不會動 PG row）。

`POST /api/admin/devices/cleanup-offline`

```JSON
{ "olderThanSec": 86400 }
```

或 `?olderThanSec=86400` query，預設 86400（24h）。

`200 OK`：

```JSON
{ "success": true, "count": 3, "devices": [...], "olderThanSec": 86400 }
```

08

5. Device Config

`GET /api/devices/:id/config`

回 device 的最新 config JSON（剝掉 `crc32` 欄位）。

`POST /api/devices/:id/config`

```JSON
{ "deviceName": "opta-line-3", "mqtt": { ... }, ... }
```

Side effects：

1. 寫 `backups/config.bak.{ts}.json` 並 prune 7 天以上舊檔
2. 觸發 `saveSnapshotIfChanged`（source=`auto-push`；v2.2.3+）
3. 取消任何 pending `download_update` command

`200 OK`：`{"success": true, "message": "Config updated"}`

09

6. Backups & Restore

`GET /api/devices/:id/backups`

```JSON
[
  {
    "filename": "config.bak.1748044800.json",
    "timestamp": 1748044800,
    "date": "2026-05-24T00:00:00.000Z",
    "sizeBytes": 8421,
    "summary": { "added": 1, "removed": 0, "changed": 3 }
  }
]
```

`GET /api/devices/:id/backups/:filename`

`filename` 需符合 `config.bak.\d+.json`，否則 `400 Bad Request`。回完整 backup JSON。

`GET /api/devices/:id/diff?file=config.bak.{ts}.json`

```JSON
{
  "added":   { "mqtt.useTls": true },
  "removed": { "mqtt.legacyField": "..." },
  "changed": { "mqtt.broker": { "from": "...", "to": "..." } }
}
```

`POST /api/devices/:id/restore`

```JSON
{ "filename": "config.bak.1748044800.json" }
```

設置 `pendingCommand=download_update` + `pendingWebUpload=true`，下次 heartbeat device 會收到指令並主動拉新 config。

10

7. Snapshots（自動觸發）

不同於 manual backup，snapshot 由系統在 `configVersion` 變化時自動產生。`backups/snapshot.{ts}.v{version}.json`。**最多保留 50 個 / 30 天**。

`GET /api/devices/:id/snapshots?limit=20`

```JSON
[
  {
    "filename": "snapshot.1748044800.v17.json",
    "timestamp": 1748044800,
    "date": "2026-05-24T00:00:00.000Z",
    "configVersion": 17,
    "sizeBytes": 8421,
    "source": "auto-heartbeat",
    "deviceName": "opta-line-3",
    "summary": { "added": 1, "removed": 0, "changed": 3 }
  }
]
```

`source` 可能值：

- `auto-push` — 由 `POST /api/devices/:id/config` 觸發
- `auto-heartbeat` — heartbeat 偵測 `configVersion` 變化觸發

`GET /api/devices/:id/snapshots/:filename`

回完整 snapshot：

```JSON
{
  "_snapshot": { "timestamp": 1748044800, "configVersion": 17, "deviceName": "...", "source": "auto-pu
  "deviceName": "...",
  "mqtt": { ... }
}
```

`POST /api/devices/:id/snapshots/:filename/rollback`

設 `pendingCommand=download_update` 並把 snapshot 內容寫回當前 config。

```JSON
{ "success": true, "message": "Rolled back to snapshot", "newConfigVersion": 18 }
```

11

8. Web Upload

Device firmware 把 inline 的 `WebPage.h` 算 CRC32 上傳;server 比對 CRC,若無變更則跳過寫檔省流量。

`POST /api/devices/:id/web`

Raw binary body(`Content-Type: application/octet-stream`,最大 2 MB)。

`200 OK`:

```JSON
{ "success": true, "bytes": 23456, "crc": 3735928559 }
```

未變更:

```JSON
{ "success": true, "unchanged": true, "crc": 3735928559 }
```

`GET /api/devices/:id/web/crc`

```JSON
{ "crc": 3735928559 }
```

無紀錄回 `{"crc": 0}`。

---

9. Firmware Management

Public — Device 用

`GET /api/firmware/latest?channel=stable`

`channel` 可選 `stable` / `beta` / `dev` / `all`,預設 `stable`。

```JSON
{
  "filename": "opta_v5.9.131.bin",
  "version": "5.9.131",
  "variant": "standard",
  "releaseChannel": "stable",
  "size": 781234,
  "sha256": "ab12...",
  "changelog": "- Fix MQTTS handshake\n- Add ...",
  "downloadUrl": "/api/firmware/download/opta_v5.9.131.bin"
}
```

無符合：`{"error": "No firmware in channel", "hint": "Try ?channel=all"}`

`GET /api/firmware/download/:filename`

回 `application/octet-stream`。`filename` 必須是 `.bin`，否則 `400`。

Admin — Dashboard 用

`POST /api/firmware/upload`

Raw `.bin` body（最大 4 MB）。Headers：

Header	必要	說明
`X-Firmware-Version`	✓	`M.m.p` 格式（如 `5.9.131`）
`X-Firmware-Changelog-B64` 或 `X-Firmware-Changelog`	✓	≥ 10 字元（純文字者直接放，URL-safe 者放 base64）
`X-Firmware-Channel`		`dev` / `beta` / `stable`（預設 `dev`）
`X-Firmware-No-License`		`1` = no-license variant（變體欄位）

`200 OK`：

```JSON
{
  "ok": true,
  "filename": "opta_v5.9.131.bin",
  "size": 781234,
  "sha256": "ab12...",
  "releaseChannel": "dev",
  "changelogLength": 142
}
```

`GET /api/firmware?channel=stable&variant=all`

回 channel 下韌體列表，附「目前在線跑這版的 device 數」統計：

```JSON
[
  {
    "filename": "opta_v5.9.131.bin",
    "version": "5.9.131",
    "variant": "standard",
    "releaseChannel": "stable",
    "size": 781234,
    "sha256": "ab12...",
    "devices": { "total": 8, "online": 5, "byVersion": { "5.9.131": 3, "5.9.130": 2 } }
  }
]
```

`GET /api/firmware/:filename/devices`

`GET /api/firmware/:filename/channel-history`

回 promote/demote 歷史，每筆含 `{ timestamp, fromChannel, toChannel, changedBy, reason }`。

`POST /api/admin/firmware/:filename/channel`（或 `PATCH`）

```JSON
{ "channel": "stable", "reason": "Passed regression suite 2026-05-24" }
```

> 雙寫 POST + PATCH 是因為前端 nginx CDN 預設擋 PATCH。後端兩個 verb 都接到同一 handler。

`200 OK`：

```JSON
{ "ok": true, "filename": "opta_v5.9.131.bin", "releaseChannel": "stable", "previousChannel": "beta" }
```

`DELETE /api/admin/firmware/:filename?force=1`

- 若該 firmware 在 `stable` channel → `409 Conflict`（必須先 demote）
- 若有在線 device 跑這版 → `409 Conflict`，加 `?force=1` 強制
- 若有 in-flight OTA → `409 Conflict`

`POST /api/firmware/deploy/:deviceId`

```JSON
{ "filename": "opta_v5.9.131.bin" }
```

Query `?force=1`：device 收到 heartbeat 回應後立刻自動執行（不等使用者按 update）。

```JSON
{ "ok": true, "message": "OTA queued", "targetVersion": "5.9.131", "targetDevice": "opta-line-3", "for
```

13

10. OTA（已廢棄 `.mesb` 格式）

```
POST /api/devices/:id/ota
GET  /api/devices/:id/ota/latest
```

兩個一律回 `410 Gone`，body 內附遷移指引：

```JSON
{
  "error": "Gone",
  "message": "The .mesb OTA API has been removed. Use /api/firmware/upload and /api/firmware/deploy/:d
  "migration": "https://docs.smms.com.tw/migration/mesb-to-firmware-v2"
}
```

14

11. Customers（多租戶）

`GET /api/customers?tier=...&active=true`

```JSON
[
  {
    "id": 5,
    "name": "Foxconn Tier-1",
    "short_code": "FXN",
    "tier": "enterprise",
    "email": "...",
    "contactPerson": "...",
    "status": "active",
    "createdAt": "2026-04-01T00:00:00.000Z",
    "deviceCount": 12,
    "onlineCount": 8
  }
]
```

`POST /api/customers`

```JSON
{ "name": "...", "short_code": "...", "tier": "smb", "email": "...", "contactPerson": "...", "status":
```

`201 Created`。

`PATCH /api/customers/:id`

部分更新。

`DELETE /api/customers/:id`

不會刪 device，只把該 customer 旗下 device 改為「未指派」。

`POST /api/devices/:uid/assign-customer`

指派：

```JSON
{ "customerId": 5 }
```

解除指派：

```json
{ "customerId": null }
```

Find-or-create（沒有對應 customer 自動建一個）：

```json
{ "customerName": "新客戶 X" }
```

15

12. Admin Users（Sprint 21）

`GET /api/admin/users`

```json
[
  { "id": 1, "email": "kc@...", "role": "admin", "displayName": "KC", "is_active": true }
]
```

`POST /api/admin/users`

```json
{ "email": "ops@example.com", "password": "min8chars", "role": "customer", "displayName": "Ops" }
```

`PUT /api/admin/users/:id/password`

```json
{ "newPassword": "min8chars" }
```

`PUT /api/admin/users/:id`

```json
{ "role": "customer", "displayName": "...", "isActive": false }
```

16

13. Audit Logs

`GET /api/audit-logs?limit=50&offset=0`

`limit` 最大 200。

```JSON
{
  "logs": [
    {
      "timestamp": "2026-05-24T06:00:00.000Z",
      "action": "license.approve",
      "userId": 1,
      "objectId": "AA:BB:CC:DD:EE:FF",
      "details": { "previousStatus": "pending", "newStatus": "active" },
      "ip": "1.2.3.4"
    }
  ],
  "total": 4123,
  "limit": 50,
  "offset": 0
}
```

17

14. PG Backup Status（v2.2.6+）

`GET /api/admin/backup-status`

讀 `/var/backups/pg` mount 點，回傳每日 `pg_dump` 健康度：

```JSON
{
  "mounted": true,
  "dumpDir": "/var/backups/pg",
  "dumpCount": 14,
  "lastDumpFile": "pg_dump_2026-05-23.sql.gz",
  "lastDumpAt": "2026-05-23T02:00:00.000Z",
  "hoursSinceLastDump": 28,
  "oldestDumpAt": "2026-05-10T02:00:00.000Z",
  "totalSize": 1583492000,
  "healthy": false
}
```

`healthy: false` 表示最後一份 dump 距今 > 26 小時——需要排查 cronjob / mount。

18

15. Cloud Bridge

`GET /api/cloud/status`

Local 模式：

```JSON
{ "enabled": false, "reason": "SERVER_ROLE=local" }
```

Cloud 模式：

```JSON
{
  "enabled": true,
  "started": "2026-05-20T14:00:00.000Z",
  "mqtt": { "connected": true, "host": "...", "port": 1883 },
  "arduinoCloud": { "connected": true, "thingsActive": 12 },
  "devices": 12
}
```

19

☁️ License API（Cloud-only）

License 採七段狀態機，由 device-side activate / heartbeat 與 admin-side 審核共同推進：

State	進入條件	退出條件
`unseen`	Heartbeat 看到新 UID 自動建立	Device 顯式 `activate` → `pending`
`pending`	Device `activate` 或 admin 把 unseen 提升	Admin `approve` → `active`；`reject` 維持 pending
`whitelisted`	Admin `import` 批次預核	Device `activate` 直接到 `active`
`active`	`approve` 或 whitelist + activate	Admin `deactivate` → `rejected`
`rejected`	Admin `reject` / `deactivate`	Device 重新 `activate` → `pending`
`revoked`	Admin `DELETE`（90 天 tombstone）	Admin `DELETE /api/admin/license-revoked` 清除

`POST /api/license/activate`

```JSON
{ "uid": "AA:BB:CC:DD:EE:FF", "deviceName": "opta-line-3", "firmwareVersion": "5.9.130", "agreed": tru
```

`200 OK`（whitelisted）：

```JSON
{ "status": "active", "message": "License activated", "licenseId": "LIC-2026-001", "activatedAt": "202
```

`200 OK`（未預核）：

```JSON
{ "status": "pending", "message": "License pending admin approval" }
```

`403`（revoked tombstone 仍有效）：

```JSON
{ "status": "rejected", "message": "UID revoked. Contact admin." }
```

`GET /api/license/status/:uid`

```JSON
{
  "uid": "AA:BB:CC:DD:EE:FF",
  "status": "active",
  "activatedAt": "2026-05-24T06:00:00.000Z",
  "deviceName": "opta-line-3",
  "expiresAt": null
}
```

Admin

`GET /api/admin/license/list?status=pending`

```JSON
{
  "stats": { "total": 142, "active": 98, "pending": 12, "whitelisted": 20, "rejected": 8, "unseen": 4
  "licenses": [ { "uid": "...", "status": "pending", "deviceName": "...", "firstSeenAt": "..." } ],
  "revoked": [ { "uid": "...", "revokedAt": "..." } ]
}
```

`POST /api/admin/license/import`

```JSON
{ "uids": ["AA:BB:CC:DD:EE:01", "AA:BB:CC:DD:EE:02"] }
```

`{ "success": true, "imported": 1, "skipped": 1, "total": 2 }` — pending 的會升級 whitelisted；已 active 的 skip。

`POST /api/admin/license/approve/:uid`

Pending → active，發出 `licenseId`。

`POST /api/admin/license/reject/:uid`

任意狀態 → rejected。**Device 可再次 activate 推回 pending**——這跟 deactivate 不同。

`POST /api/admin/license/deactivate/:uid`

Active → rejected，**無 grace period**——下次 heartbeat 立刻通知 device 失權。

`DELETE /api/admin/license/:uid`

硬刪 license row + 寫 90 天 tombstone。Device 之後 activate 會直接拒絕。

```JSON
{
  "success": true,
  "deleted": { "uid": "...", "status": "active" },
  "tombstoneTtlDays": 90,
  "hint": "To allow re-activation, call DELETE /api/admin/license-revoked/:uid"
}
```

`DELETE /api/admin/license-revoked/:uid`

清掉 tombstone（讓被刪 UID 可再 activate）。

20

HTTP 狀態碼

Code	意義
`200`	成功
`201`	Created（POST `/api/customers`、`/api/admin/users`）
`400`	Bad Request（validation 失敗、filename 格式錯）
`401`	Unauthorized（缺 Bearer / token 失效）
`403`	Forbidden（role 不足；rejected/revoked license）
`404`	Not Found
`409`	Conflict（韌體在用、stable channel 不可刪）
`410`	Gone（`.mesb` API 已廢；claim OTP 過期）
`500`	Server error

統一錯誤 shape：

```JSON
{ "error": "Unauthorized", "message": "Missing Authorization header" }
```

21

Body Size Limits

Endpoint	Limit	設定處
`express.json()` 預設	1 MB	`server.js:37`
`/api/devices/:id/web`	2 MB	`server.js:39`
`/api/devices/:id/ota`	2 MB	`server.js:41` (廢)
`/api/firmware/upload`	4 MB	`server.js:43`

22

CORS

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS
Access-Control-Allow-Headers: Content-Type, Authorization
```

OPTIONS 一律回 `200`。

23

Legacy / Migration

Method	Path	行為
`GET`	`/config`	`400 — Migration: Use /api/devices/:id/config instead.`
`POST`	`/config`	同上
`PUT`	`/config`	同上

24

近期變更（v2.2.x highlights）

Version	變更
v2.2.10 (`dff2c2d`)	`/api/devices` 改用 PG (`uid` PK) 列設備，修同 `deviceId` 互蓋 bug。
v2.2.6	新增 `/api/admin/backup-status`，監控 daily `pg_dump`。
v2.2.4	Self-serve `POST /api/auth/password` + admin `PUT /api/admin/users/:id/password`。
v2.2.3	`POST /api/devices/:id/config` 觸發 `auto-push` snapshot，補上 heartbeat-only 走過的縫。
Sprint 21	WI-065 JWT auth、WI-070 device claim OTP、WI-071 user CRUD。
ADR-008	單一 codebase + `SERVER_ROLE` 切換 local/cloud；license 模組僅 cloud 載入。
WI-077	MQTT credentials 在 env 改用 base64-encoded default，避免明文密碼進 git。

25

實作備註

1. **Auth 順序**：JWT → DEVICE_TOKEN → License Token。任一吻合就 `next()`，全失敗才回 401。失敗事件打 `auth.reject` log。
2. **Mount 順序很關鍵**：`server.js:103-117` 先掛 public router（auth/firmware-public/deviceClaim/heartbeat），再 `app.use('/api/devices', authMiddleware)` 等於把 `/api/devices/*` 全段保護起來——但 heartbeat 已先掛載，所以它仍是 optional auth。
3. **Tenant filter**：`customer` role 在 `GET /api/devices` handler 內查 user → device 對應關係後過濾結果。Admin role 看全部。
4. **Heartbeat 是 cloud sync 的主軸**：device → server 推狀態、server → device 回授權與 OTA 指令；任何 admin 動作（approve / deploy / restore）都是設 pending flag，等下一次 heartbeat 才生效。
5. **Snapshot vs Backup**：Backup 是手動觸發 `POST /api/devices/:id/config` 寫的 timestamped 副本（7 天輪替）；Snapshot 是系統在 configVersion 改變時自動寫的（50 個 / 30 天）。Rollback 走 snapshot。
6. **Firmware 刪除安全網**：Stable channel 必須先 demote 才能刪；有 in-flight OTA 或在線 device 跑該版時要 `?force=1` 才放行。
7. **License revoked tombstone**：硬刪後 90 天內 device activate 會被拒（避免使用者隨意拔 PG row 後 device 自動復活）；要復活先 `DELETE /api/admin/license-revoked/:uid`。
8. **PATCH 雙寫 POST**：`/api/admin/firmware/:filename/channel` 同時掛 POST 跟 PATCH——前端 nginx CDN 預設擋 PATCH，POST 是 escape hatch。

04 開發者 / 整合

MQTT 整合

狀態:✅ **IMPLEMENTED(v6.0.83,2026-08-28 更新 WI-211 QoS／節流／週期重發／payload 格式)** | 原草擬:2026-06-02

01

⓿ WI-155(v5.9.272–283)— 對外命名空間 & 身分更新(整合商先讀)

對外 topic 前綴 `<P>` 從「**強制 UID 前綴**」升級為「**可設定命名空間 + MAC**」,並保持**完全向後相容**:

模式	topic 前綴 `<P>`	何時
新(命名空間)	`{projectname}-{env}/gateway/{mac12hex}`	設定頁填了「專案名稱」+「環境」(兩者皆全小寫;mac=12 hex 無冒號)
舊(legacy,預設)	`mes/gateway/{uid}`	未設專案名稱/環境時(既有設備不變)

範例:`topwell-dev/gateway/a8610a508b61`、`kspicnc-staging/gateway/aabbcc445566`。

整合商重點:

1. **訂閱用萬用字元涵蓋兩種**:`+/gateway/+/info`、`+/gateway/+/status` …(`+` 第一段同時吃 `mes` 與 `{proj}-{env}`)。
2. **MAC = 對外身分**:`whoami` / `info` 回應**已加** `mac` **欄位**(12 hex 小寫,= 新格式 topic 的 mac 段、= MQTT clientId `OPTA-<mac>` 的 mac)。內部路由/授權仍以 **UID** 為鍵(UID↔MAC 由雲端心跳自動建表,WI-147 不破)。

3. **MAC 撞號自動解決(AC4)**:同 MAC 多台時,雲端令「後到者」自動降級回 legacy `mes/gateway/{uid}`(永遠唯一、不誤路由);撞號排除後自癒。管理端可查 `GET /api/devices/mac-collisions`。
4. `cmd/signal/<N>` **payload**:同時接受 `{"value":N}` **(新)** 與**裸數字(舊)**,平滑遷移。
5. **MAC override(進階)**:設定頁可指定軟體 MAC(不碰硬體 OTP);設了會失聯的 MAC → 試用期 90s 內無 IP 自動退回硬體 MAC(防救不回)。⚠ 改 MAC 後設備可能取得**新 IP**(到路由器查或看雲端 ip);失聯排查見 `../guides/guide-troubleshooting.md` 〈MAC 覆寫後找不到設備〉。一般無需使用。

> 下文 §0 起的 `<P>` = `mes/gateway/<UID>` 為 **legacy 表示法**;啟用命名空間時整段前綴改為 `{proj}-{env}/gateway/{mac}`,**其後的子主題(`/cmd/...`、`/info`、`/status`、CONVERTER 輸出…)結構完全不變**。

> 原始狀態:草擬:2026-06-02
>
> ⚠ **實作與本草案的差異**:因設備 MQTT buffer 限 512B,**通道清單改「分頁」**—— `info` 只回摘要(身分/net/counts),每個通道另發 `{prefix}/info/io/<id>`(見 §3 實際格式)。草案 §3 的「單一 info 內含 channels[]/commands[]」未採用。`commands[]`(cmd/signal 角色標註)**v1 未做**,夥伴端可由 `info/io/<id>` 的 in 通道 topic 推得:規則引擎以 `cmd/signal/<N>` 數值訊號為來源,訊號可為 latched(持續閘控)或 pulse(上升緣觸發),再對應到 TCP IO 通道。對象:**現場管理端整合方**(第三方軟體團隊)+ 本控制器韌體開發
>
> 本合約定義第三方現場管理軟體如何**只透過 MQTT**(不需逐台 IP、不需走雲端):①探索現場有哪些設備　②取得設備管理資訊　③知道能對哪些 topic 做遠端控制。
>
> 已實作的底層語意:`cmd/signal/<N>` 為規則引擎的數值訊號來源,可作 latched(持續閘控)或 pulse(上升緣觸發),並對應到 TCP IO 邏輯通道。

02

❶-2 WI-211（v6.0.83）— QoS、發送節流與 payload 格式（整合商必讀）

一、QoS 預設值改了

方向	預設	為什麼
設備發出（狀態、回報）	**1**（至少一次）	送出去的絕大多數是「**最後狀態**」——重複送達會被下一筆覆蓋，無害。需要的是「一定會到」，不是「恰好一次」。
設備發出（`<P>/rules/*/trigger` 純通知）	**0**（最多一次）	它本來就是**設計上可丟**的通知（設備已刻意合併中間值）。QoS 1 每則都要等 `PUBACK`，函式庫逾時 1000ms——broker 在網際網路上只要塞車超過一秒就判定失敗並斷線重連。遺失風險由**週期重發**兜底（見下）。
設備接收（`<P>/cmd/#` 命令）	**2**（恰好一次）	命令是控制，**不可重複執行**（例如「開燈」被送兩次）。

v6.0.82 之前**收發都寫死 QoS 2**，且無法設定。實測每則發送要 265~400ms 且期間設備主迴圈完全凍結—— 密集狀態變化時會把連線打掉。現在兩者都可在設備網頁（設定 → MQTT）調整。

> **對整合商的影響**：若你的訂閱端假設 QoS 2 的「恰好一次」語意，請改為**冪等處理**（同一個狀態值收到兩次，結果必須相同）。狀態類 topic 本來就該這樣處理。

二、⚠️ 不要假設「一個事件 = 一則訊息」

v6.0.82 起，設備對**狀態類 topic** 做兩件事：

1. **同 topic 合併**：同一個 topic 還在發送佇列裡尚未送出時，新的值**直接取代**舊的，不新增一則。
2. **每 topic 最小發送間隔 1 秒**：同一個 topic 在 1 秒內的連續變化，只會送出**最後一個值**。

單次變化不受影響（該 topic 若 1 秒內沒送過就立刻送）。只有**連續快速變化**會被壓。

> **這是刻意的。**設備是單執行緒工控閘道，每則 MQTT 發送都會阻塞現場控制。密集發送曾造成連線反覆斷開、甚至設備自我重開（WI-210）。**控制的即時性優先於通知的完整性** —— 中間被覆蓋的值本來就已經不是現況了。

若你需要完整的變化歷程，看下一節。

二-B、週期重發：過時上限 60 秒

設備每 **60 秒**會把每個狀態類 topic 的**現況原樣重送一次**（該 topic 若期間本來就有更新，則不重送）。

> **這是給你的保證**：即使某一則訊息在傳輸中遺失，**你的狀態最多過時 60 秒**，不會永遠停在錯的值。這也是 `rules/*/trigger` 敢用 QoS 0 的前提。
>
> 為什麼不是「同一則發兩次」：兩份相隔幾毫秒的複本會**一起死在同一條斷掉的連線上** —— 而連線斷掉正是 QoS 0 唯一會發生的遺失情境。重複發送擋不到它，週期重發可以。

對整合商的影響：你會週期性收到「內容沒變」的訊息。這是正常的，請以冪等方式處理（收到相同狀態值不應觸發重複動作）。已被週期推送的資料（如 RS485 量測值）不受影響 —— 它們本來就在更新，不會額外重發。

三、payload 夾帶歷史（選用，預設關閉）

設備網頁勾選「payload 夾帶歷史紀錄」後，**狀態類 topic 的 payload 會從裸值改成 JSON**：

```jsonc
// 關閉時（預設）── 格式與 v6.0.82 之前完全相同
12.345

// 開啟後
{"v":12.345,"t":1787839126,"h":[[354,12.1],[712,12.28]]}
```

欄位	意義
`v`	**最後狀態**（即時值）。只要即時狀態的介面**只讀這個欄位就夠了**。
`t`	本批的起點時間，**Unix epoch 秒**。`t=0` 代表設備時鐘尚未同步 → 只有相對順序可信。
`h`	上一次發送到這一次之間，被合併掉的每一筆：`[相對 t 的毫秒數, 當時的狀態]`。
`trunc`	只在歷史超過 8 筆被截斷時出現（`true`）。**代表這批不完整**，不要當成只發生了 8 次。

實際樣本（本站實測）：

```json
{"v":"A2-MQTT滅","t":1787839126,"h":[[354,"A2-MQTT滅"]]}
```

`v` 的型別跟隨原本的值：數字不加引號、字串加引號。

> ⚠️ **開啟這個選項會破壞既有的解析程式。**這是知情的選擇，不是升級後被動改變 —— 韌體升級到 v6.0.82 **不會**自動開啟它。要開之前，先確認所有訂閱端都改好了。
>
> 為什麼上限是 8 筆：單則 payload 的硬上限是 128 bytes。超過就標 `trunc`。

03

0. 完整系統 MQTT Topic 清單（權威）

`<P>` = topic 前綴：legacy `mes/gateway/<UID>`（24 字 UID）**或** WI-155 命名空間 `{projectname}-{env}/gateway/{mac12hex}`（見 § ❶）。以下子主題結構兩種模式通用。為**韌體實際 sub/pub 的系統 topic**（v5.9.283 核對 `src/MesMQTT.h` / `main.cpp`）。

探索 / 身分 / 狀態

Topic	方向	Retained	說明
`mes/gateway/whoami`	管理端 → 全設備（**免前綴廣播**）	–	探索全廠；各設備回 `<P>/info`
`<P>/cmd/info`	管理端 → 設備	–	指定查詢單台 → 回 `<P>/info`
`<P>/info`	設備 → 管理端	–	身分摘要（uid/name/fw/ip/online/uptime/counts）
`<P>/info/io/<id>`	設備 → 管理端	–	每通道控制/資料介面（分頁，避 512B 上限）
`<P>/status`	設備 → broker	✅	在線狀態 birth/LWT（`online`）

命令（管理端 → 設備，`<P>/cmd/#`）

Topic	Retained	說明
`<P>/cmd/signal/<N>`	❌ **切勿 retain**	MQTT 訊號值（規則引擎來源；觸發/閘控）。payload: `{"value":N}`（WI-155 新）**或**裸數字（舊）皆可
`<P>/cmd/do/<N>`	–	數位輸出 DO<N> 控制
`<P>/cmd/scale/active`	–	磅秤發重量門控（`1` 開 / `0` 停）
`<P>/cmd/scale/calibrate`	–	磅秤校正命令
`<P>/cmd/telemetry`	–	啟動聚合遙測 + keepalive（payload=間隔秒數，`0` 停；超時自停）
`<P>/cmd/_ping`	–	**內部** self-ping：設備每 45s 發給自己訂的此 topic，確認 MQTT subscribe 仍活著

回報 / 輸出（設備 → 外）

Topic	Retained	說明
`<P>/telemetry`	–	聚合遙測（`cmd/telemetry` keepalive 門控才發）
`<P>/cmd/scale/calibrate/ack`	–	磅秤校正命令 ack
`<P>/<自訂>`	–	**CONVERTER 通道輸出**，topic 由設定決定。慣例：`<P>/scale/weight`（磅秤）、`<P>/power`（用電量）、`<P>/count`（SF965 計米）；見 `../guides/guide-triggered-periodic-report.md`

授權（雲端 → 設備）

Topic	Retained	說明
`<P>/license/state`	✅	簽章 token 下發（base64；WI-145；唯一 publisher = 雲端；retained 重連即重投最新 token）

雲端中繼（cloud-bridge）

- 雲端 `cloud-bridge` 訂 **`+/gateway/#`**（WI-155:同時涵蓋 legacy `mes/gateway/#` 與命名空間 `{proj}-{env}/gateway/#`）做 telemetry fan-out（→ `wss://opta.smms.com.tw/ws/telemetry`）；遙測拆解子主題如 `<P>/rs485/<idx>/<field>`、`<P>/power/<idx>/<field>`。
- WI-155:命名空間 topic（`{proj}-{env}/gateway/{mac}/...`）由 cloud-bridge 經 **MAC↔UID 反查**改寫成 canonical `mes/gateway/{uid}/...` 後再處理 → 下游（PG/Things、WS fan-out）與儲存皆以 UID 為鍵，整合不受命名空間影響。

> **小結**：固定系統 topic 約 **14 條**（探索 5 + 命令 6 + 回報 2 + 授權 1），外加數量不定的 **CONVERTER 自訂輸出** topic。⚠️ `cmd/signal/*` 與所有 `cmd/*`：**MQTT 無 per-message 授權，控制權 = broker 連線權**（見 §6 安全）。`cmd/*` 一律別 retain（retained 重連會反覆灌回誤觸發）。

04

1. 為什麼需要這份合約

- 雲端 config server 有自己的設備管理；但**現場管理端是另一個團隊的獨立軟體**，不走雲端、不便逐台用 HTTP（要先知道每台 IP）。
- 控制器與現場管理端**共用現場 MQTT broker**。最自然的整合面就是 MQTT：pub/sub、免逐台 IP、可廣播。
- **前綴 `mes/gateway/<UID>` 是強制的、綁設備 UID**（韌體以設備 24 字 UID 組 topicPrefix，確保每台 topic 命名空間唯一、不撞台）。→ 管理端「要先知道 UID 才能跟設備對話」，因此需要**不需先知 UID 的探索機制**。

05

2. 介面總覽（三個能力）

能力	方向	Topic	機制
探索 (whoami)	管理端→全設備	`mes/gateway/whoami`（**免前綴廣播**）	廣播請求
資訊回報 (info)	設備→管理端	`<P>/info`（`<P>` = `mes/gateway/<UID>`）	回應 / 也可指定查 `<P>/cmd/info`
在線狀態 (status)	設備→管理端	`<P>/status`（retained）	既有，birth/LWT

> 探索流程：管理端訂 `mes/gateway/+/info`（萬用）→ 發一筆 `mes/gateway/whoami`（payload 空或 `1`）→ **每台設備各自回 `<P>/info`**（含自己的 UID）→ 管理端即取得全廠清單與每台的控制合約。指定查詢：已知 UID 時發 `<P>/cmd/info`（payload 空）→ 該台回 `<P>/info`。

06

3-IMPL. 實際回應格式（v5.9.197，分頁）★以此為準

收到 whoami / cmd/info 後，設備發**多筆**訊息（QoS0、非 retained）：

① **摘要 →** `<P>/info`（單筆，小）：

```JSON
{"schema":1,"uid":"00310044303351103432393 2","mac":"a8610a508b61","name":"OPTA-K(
 "ip":"192.168.72.77","online":true,"uptimeSec":46,
 "counts":{"signals":7,"rules":7,"actions":6,"tcpio":5}}
```

> WI-155(v5.9.279+):新增 `mac` 欄(12 hex 小寫)= 對外身分。整合端用它對應 MAC-based topic;內部仍以 `uid` 為鍵。② **每個啟用通道 →** `<P>/info/io/<channelId>`（控制/資料介面，分頁）：

```JSON
{"id":5,"name":"啟動磅秤","dir":0,"mode":0,"proto":0,"topic":"mes/gateway/<UID>/cm
{"id":22,"name":"發出重量","dir":1,"mode":2,"proto":0,"topic":"mes/gateway/<UID>/s
```

- `dir`：0=IN（管理端可發此 topic 控制）／1=OUT（設備發、管理端訂閱取資料）
- `mode`：0=Legacy／1=Parser／2=Converter　`proto`：0=MQTT／1=Modbus TCP
- 管理端流程：訂 `mes/gateway/+/info` + `mes/gateway/+/info/io/+` → 發 `mes/gateway/whoami` → 收齊全廠摘要＋各通道。

> 以下 §3 為**原始草案**（單一大 JSON），保留作設計脈絡；實際以上方 §3-IMPL 為準。

07

3. `info` 回應 payload（JSON）★合約核心〔草案，未採用〕

> 重點：`channels` **與** `commands` **就是遠端控制合約** —— 管理端據此知道「能對哪些 topic 發什麼、設備從哪些 topic 出資料」。

```jsonc
{
  "schema": 1,                          // 合約版本，破壞性變更才 +1
  "uid": "003100443033511034323932",
  "name": "OPTA-KC-A1",                 // 設備名稱（設定頁可改）
  "model": "MES-Gateway-Opta",
  "fw": "5.9.196",
  "net": { "ip": "192.168.72.77", "mac": "AA:BB:..", "online": true, "uptimeSec"
  "rs485": [                            // 掛載的 Modbus 設備
    { "name": "TDA08B-1", "slave": 3, "baud": 9600, "online": true }
  ],
  "counts": { "signals": 7, "rules": 7, "actions": 6, "tcpio": 5 },

  // ★ 遠端控制 / 資料介面（TCP IO 通道）— 管理端最需要的
  "channels": [
    { "name": "發出重量", "dir": "out", "proto": "mqtt",
      "topic": "mes/gateway/<UID>/scale/weight", "desc": "每2秒，門控開時。payload
    { "name": "啟動磅秤", "dir": "in",  "proto": "mqtt",
      "topic": "mes/gateway/<UID>/cmd/signal/7", "desc": "發重量開關" }
  ],

  // ★ 內建命令暫存器對照（cmd/signal/N，遠端控制用）
  "commands": [
    { "topic": "mes/gateway/<UID>/cmd/signal/5", "role": "啟動",       "kind": "p
    { "topic": "mes/gateway/<UID>/cmd/signal/6", "role": "校正參數",   "kind": "la
    { "topic": "mes/gateway/<UID>/cmd/signal/7", "role": "發重量開關", "kind": "la
  ]
}
```

欄位語意

- `kind`：`pulse`＝要送 `1` 再 `0`（邊緣觸發）；`latched`＝送一次保留（狀態）；`level`＝持續電平門控。管理端據此知道**怎麼驅動**每個控制 topic（pulse 須送 `1` 再送 `0` 才不會卡在觸發態，latched/level 則送一次即保留，故只要遵守 kind 語意就不會前後競態）。

- `dir`：`in`＝管理端可發佈來控制；`out`＝設備發佈、管理端訂閱取資料。
- `topic`：合約回**完整 topic**（已含前綴），管理端直接用，不必自己拼。

> **大小考量**：`channels` 多時 payload 會變大。若超過單筆 MQTT/RAM 上限，v1 可：① `info` 只回身分+net+counts，②另設 `<P>/cmd/io` → `<P>/io` 專回完整 channels 清單（分頁）。實作時定。

08

4. 安全性（實作前必須拍板）

- MQTT 沒有 per-message 授權 —— **任何能連上 broker 的人都能發 whoami / cmd/signal**。
  - `info` 是**唯讀管理資訊** → 一般可接受（不含密碼/token）。
  - 但 `cmd/signal/*`（遠端控制）同樣無授權 → **控制權＝broker 連線權**。現場 broker 的帳密/網段隔離就是安全邊界。
  - ⚠️ 若管理資訊含敏感欄位（MAC、內網拓樸）或要限制控制，需在 broker ACL（依 topic/帳號授權）層處理，韌體不做 per-client 授權。
- 廣播 whoami 要評估**多設備同時回報的訊量**（N 台 → N 筆 info）；可加隨機抖動避免同時湧出。

09

5. 與現有介面的關係

現有	提供什麼	為何仍需本合約
雲端 config server	雲端設備清單/部署	現場第三方不走雲端
HTTP `/api/system`、`/api/poll`、`/api/config`、`/api/tcpio`	同等資訊（含 deviceUid、IP、通道）	要先知道每台 IP；MQTT 免逐台 IP
`<P>/status`（retained `online`）	在線與否	只有 online 字串、無設備資訊、無 UID 以外內容

> 本合約等於把 HTTP 的 `/api/system` + `/api/tcpio` 精華，用 **MQTT 廣播探索 + 回應** 包成第三方好整合的形狀。

10

6. 實作備註（韌體，WI-131 待排）

- 新增 handler：訂 `mes/gateway/whoami`（免前綴，需在既有 `cmd/#` 之外多訂一條）+ `<P>/cmd/info`。
- 組 `info` JSON：重用既有 `/api/system`、`/api/tcpio` 的序列化邏輯（ArduinoJson）。
- **Flash 預算**：目前 91.8%（餘 ~64KB）。新 handler + JSON 組裝要量 size；channels 清單可能要分頁/精簡。
- birth 是否一併豐富化（`status` 改 JSON，於上線時即帶身分摘要）→ 可選。
- 合約版本 `schema` 欄位：破壞性變更才 +1，讓管理端能相容多版設備。

11

7. 待夥伴團隊確認

- [ ] `info` 要哪些欄位（本草案：身分/net/rs485/counts/channels/commands）是否齊全？
- [ ] channels 是否需要更細（每通道的 payload schema、QoS、retain 慣例）？
- [ ] 探索：廣播 whoami 足夠，還是也要定期主動 publish（管理端被動清點）？
- [ ] 安全邊界：broker ACL 由誰管？是否需要限制 `cmd/*` 控制權？

12

8. 相關文件

- 通用引擎 / API：`spec-mqtt-chain-workflow.md`
- HTTP API：`../api/api-firmware.md`

04 開發者 / 整合

MQTT 工作流程鏈

> 📅 最後更新：2026-05-26 ｜ 🛠️ 對應韌體：v5.9.130+（含 chain/timer 批次變更）｜ 📌 負責人：KC

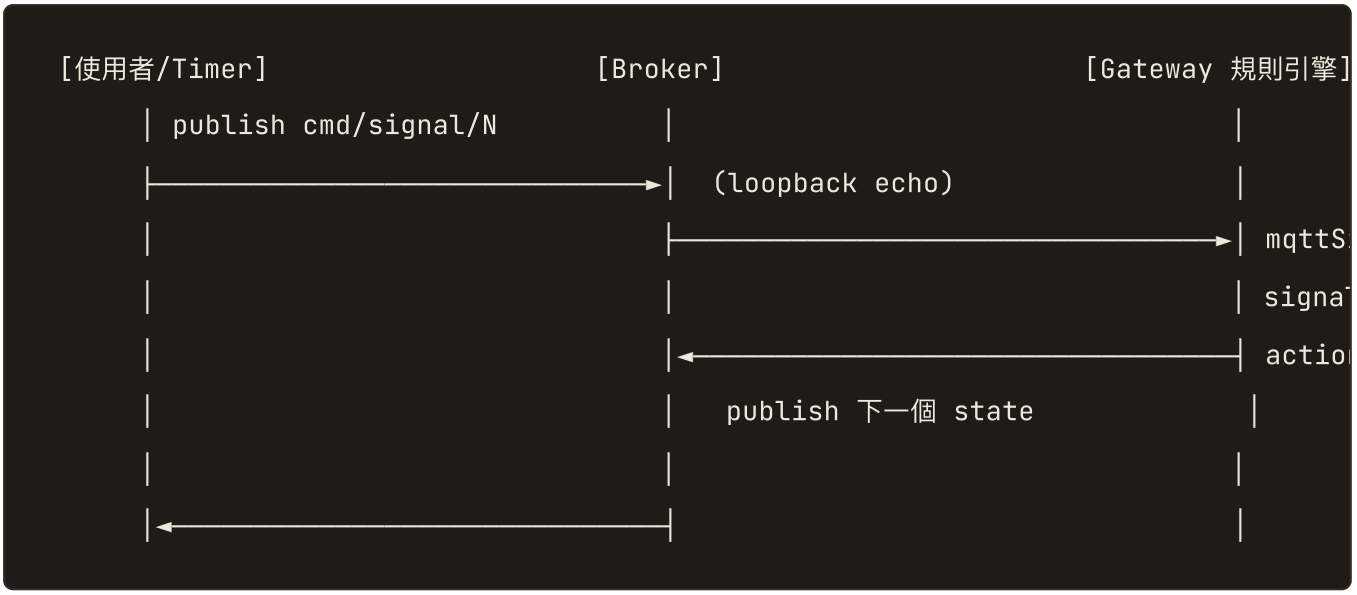
本文件說明 Opta（MES Gateway）以**純設定**（不改韌體）建構**有狀態、可組合工作流**的機制：以 MQTT topic 當「狀態匯流排」，透過規則引擎的 signal → rule → action 串接，再以 Converter timer 做定時驅動。TDA08B 秤重校正引導精靈即以此實作（見 `guide-calibration-wizard.md`）。

01

1. 核心概念

規則引擎本身是**無狀態、邊緣觸發**的：`signal`（輸入條件）為真 → `rule` 觸發 → 執行 `action`（一個或多個輸出）。要做「有狀態的多步驟流程」（例如校正精靈的 步驟1→2→3），單一規則表達不了。

解法：用 MQTT topic 當外部狀態。每個流程把自己的「目前狀態」publish 到一個 topic；該 topic 同時被設備自己訂閱回來（broker loopback），變成下一條規則的輸入條件。流程之間因此 **decouple、可組合**——一個流程發狀態，自己或別的流程訂閱去驅動下一段。



> **關鍵特性**：狀態經 broker round-trip 在**下一個 loop** 才被處理，加上引擎是「先快照所有 signal 再執行規則」，因此**天然序列化、無 cascade**（一次輸入不會連鎖跳好幾步）。

02

2. 元件對照

元件	API endpoint	角色
Signal（輸入群組）	`GET/POST /api/signals`	條件判斷。來源可為 DI/AI/MQTT/Modbus/TCP IO/虛擬通道，多來源以 AND/OR 聚合
Action（輸出群組）	`GET/POST /api/actions`	一個動作 → 多個輸出（DO / Modbus 寫 / MQTT / TCP IO publish）
Rule（規則）	`GET/POST /api/rules`	連結 signal → action
TCP IO Channel	`GET/POST/DELETE /api/tcpio`	MQTT 輸入/輸出通道；Converter 模式可定時推送模板
Converter 模板	`GET/POST /api/converter/template`	TCP IO Converter 通道的訊息模板（含 `${...}` 變數）

2.1 ChannelType（type 欄位）

值	名稱	signal 來源	action 輸出
1	CH_DI	數位輸入	—
2	CH_DO	—	數位輸出（繼電器/SSR）
3	CH_AI	類比輸入	—
5	CH_MODBUS	RS485 讀值（依 index 映射）	**RS485 暫存器寫入**（`writeValue`）
6	CH_MQTT	`mqttSignalValues[index]`（由 `cmd/signal/{index}` 更新）	—
7	CH_TCP	TCP IO 通道 currentState	**TCP IO 通道 publish**（陣列索引）
8	CH_VAR	虛擬計數器	計數器 +=

> ⚠️ **重要慣例**：規則引擎實作 inline 在 `src/main.cpp` 的網路 loop（不是 `RuleEngine.cpp` 類別——後者未被使用）。TCP IO 輸出在 main.cpp 走 **`CH_TCP` (7) + 陣列索引 + 固定 `mqttPayload`**，不是 RuleEngine 的 CH_TCPIO(9)+channelId。

03

3. API 規格

3.0 通用規則

- 認證：`Authorization: Bearer <admin-token>`（POST 需 `PERM_ADMIN`）。
- **JSON 必須 compact（冒號後無空格）**。韌體以 `body.indexOf("\"key\":...")` 嚴格字串比對解析；`json.dumps()` 預設 `", "` / `": "` 會 mismatch → 欄位被忽略、走 legacy fallback（例如 `sourceCount` 被硬塞成 1）。

  - Python：`json.dumps(obj, separators=(',',':'))`
  - Content-Length 必須是 **byte 數**（中文 UTF-8 每字 3 bytes），不是字元數。
- 設備 HTTP server 為**單執行緒**，請求須**序列化**（前一個完成再送下一個）；並發會互相 reset。POST 後建議等 ~3-6s 讓 NVS 寫入沉澱再 GET 驗證。

3.1 Signal — `POST /api/signals`

```JSON
{"index":1,"enabled":true,"name":"校正標零","trigger":3,"debounceMs":0,
 "sources":[{"type":6,"index":0,"op":0,"threshold":0,"and":true,"expIndex":0,"dea
            {"type":6,"index":1,"op":4,"threshold":0.5,"and":true,"expIndex":0,"
```

欄位	說明
`index`	signal 槽位（0–7，`MAX_SIGNAL_GROUPS=8`）。POST 同 index = 更新
`trigger`	0=ALWAYS（每 loop 評估，cur 直接反映）、1=RISING、2=FALLING、3=CHANGE
`sources[]`	1–4 個來源
`sources[].type`	ChannelType（見 2.1）
`sources[].index`	通道索引（CH_MQTT 即 `cmd/signal/{index}` 的 N）
`sources[].op`	0:== 1:!= 2:> 3:< 4:>= 5:<=
`sources[].threshold`	閾值
`sources[].and`	**與「自己」聚合到前面結果的邏輯**：true=AND、false=OR。要兩個來源 AND，**第 2 個來源** `and:true`（main.cpp 用 `source[i].andWithNext` 慣例）
`sources[].deadband`	遲滯死區（防震盪）

> 回應 `{"success":true,"index":N}`。**POST 後務必 GET 驗 `sourceCount`**：若顯示比預期少（例如 2-source 卻 sc=1），多半是 JSON 有空格或請求被截斷。

刪除：body 帶 `{"index":N,"action":"delete"}`（會級聯刪除引用此 signalId 的 rule）。

3.2 Action — `POST /api/actions`

```JSON
{"index":2,"enabled":true,"name":"標零","deviceTag":"磅秤","valueType":1,"actuatic
 "outputs":[{"type":5,"index":5,"outputMode":1,"writeValue":1},
            {"type":5,"index":7,"outputMode":1,"writeValue":1},
            {"type":7,"index":2,"outputMode":0},
            {"type":2,"index":1,"outputMode":0},
            {"type":2,"index":18,"outputMode":4}]}
```

欄位	說明
`index`	action 槽位（0–7，`MAX_ACTIONS=8`）
`outputs[].type`	ChannelType
`outputs[].index`	CH_MODBUS=modbusRegisters 陣列索引；CH_TCP=tcpIo 陣列索引；CH_DO=DO 編碼（見下）
`outputs[].outputMode`	0=直接 ON、1=映射輸出（用 `writeValue`）、2=脈衝、3=不動作、4=強制 OFF、5=toggle
`outputs[].writeValue`	（WI-118）outputMode=1 時的輸出值；shorthand 自動建 mapping{trigger=1→value}
`outputs[].pulseMs`	脈衝時長（outputMode=2）
`actuationMask`	位元遮罩，0xFF=全部啟用

CH_DO index 編碼： `(expIdx<<4)|ch`。本機 DO = expIdx 0（ch 0–3）；擴充模組 0 = index 16+ch（如 idx18=exp0 ch2、idx19=exp0 ch3）。

> ⚠️ 留意 `"value"` 欄位是 legacy（會被當 outputMode 解析），請用顯式 `outputMode` + `writeValue`。

3.3 Rule — `POST /api/rules`

```JSON
{"index":2,"name":"校正標零","enabled":true,"signalIndex":14,"triggerOnTrue":true,
 "actionIndices":[35],"actionCount":1,"priority":0}
```

欄位	說明
`index`	rule 槽位（0–15，`MAX_RULES=16`）
`signalIndex`	觸發此規則的 **signalId**（穩定 ID，非陣列索引）
`triggerOnTrue`	true=signal 為真時觸發；false=為假時觸發
`actionIndices[]`	觸發的 **actionId**（穩定 ID，最多 4 個）
`priority`	0=Normal 1=High 2=Emergency

> signal/action 用穩定 ID 互相引用（`findSignalById` / `findActionById`），所以陣列重建只要 ID 不變就安全。

3.4 TCP IO Channel — `POST /api/tcpio`

```JSON
{"name":"wz-state","protocol":0,"direction":1,"mode":0,"mqttQos":0,
 "mqttTopic":"mes/gateway/<UID>/cmd/signal/0","mqttPayload":"1"}
```

欄位	值
`protocol`	0=MQTT、1=Modbus TCP
`direction`	0=INPUT（訂閱→signal）、1=OUTPUT（發佈→action）
`mode`	0=LEGACY、1=PARSER（JSON 取值）、2=CONVERTER（模板定時推送）
`mqttTopic`	訂閱/發佈 topic（**buffer 64 bytes**；含 UID 前綴 `mes/gateway/{24}/cmd/signal/N` = 49 字）
`mqttPayload`	OUTPUT 通道發佈的固定字串（CH_TCP action 用）
`converterIntervalSec`	CONVERTER 模式定時推送間隔（秒；0=僅規則觸發）

> 回應含 `channelId`（穩定 ID）+ `idx`（陣列索引）。CH_TCP action 的 `index` 用**陣列索引**。刪除：`POST /api/tcpio` body `{"channelId":N,"action":"delete"}`。

3.5 Converter 模板 — `POST /api/converter/template`

```JSON
{"idx":7,"template":"{\"weight\":${rs485.0.weight},\"stable\":${rs485.0.stable},
```

`idx` = tcpIo 陣列索引。支援的 `${...}` 變數：

變數	值
`${io.DI0~7}` / `${io.DO0~3}` / `${io.AI0~7}`	本機 I/O 狀態
`${rs485.<SlaveID>.<field>}`	RS485 設備（**v5.9.262+：N = Modbus Slave ID,非陣列索引**;故換卡位/重排設備不會改變變數路徑）；`<field>` = voltage/current/power/reactive/apparent/pf/freq/thd/kwh，**或自訂暫存器名稱**（如 weight/stable，依 modbusRegisters 名稱查 `getCustomValue`）。例 slave 1 電壓 = `${rs485.1.voltage}`。
`${tcpio.N}`	TCP IO Parser 通道值
`${counter.N.count}` / `${timer.N.elapsed}`	虛擬通道
`${sys.uptime}` / `${sys.name}` / `${sys.ip}`	系統

> ⚠ **newlib-nano `snprintf` 不支援 `%f`**。所有浮點變數（rs485）內部以整數格式化（`%ld.%02ld`）輸出 2 位小數。

04

4. Timer（定時驅動）

Converter 通道的 `converterIntervalSec` 即一個獨立計時器：每 N 秒填充模板並 publish。**多個通道可各自設不同間隔並行**（實測 5 秒 + 10 秒並行各自準確）。

兩種用途：

1. **遙測心跳**：模板帶即時值，定時送出（例如每 5 秒發 weight）。

2. **Timer → Chain 觸發**：topic 指向 `cmd/signal/N`、payload 固定值 → 每 N 秒觸發一個 signal → 啟動下一段工作流。

> 時間基礎是 `millis()`（相對時間差），**不需 NTP**、不漂移。NTP 僅在訊息需要絕對牆鐘時間時才需要（Opta RTC 無電池備援，開機預設 2024-01-01 供 TLS）。

05

5. 已知雷區（實作必讀）

雷	說明
JSON 空格	`json.dumps` 預設有空格 → 嚴格 indexOf 比對失敗 → 欄位忽略。必用 `separators=(',',':')`
Content-Length byte 數	中文 UTF-8 每字 3 bytes，用字元數會截斷 body
單執行緒 HTTP	請求須序列化，並發互相 reset
2-source 解析	POST 後務必 GET 驗 `sourceCount`；不符就重送
andWithNext 慣例	main.cpp 用「自己」的 `and` 旗標聚合，要 AND 把第 2 來源設 `and:true`
CH_TCP vs CH_TCPIO	main.cpp 只認 CH_TCP(7)+陣列索引；RuleEngine 的 CH_TCPIO(9) 未被使用
nano-float	模板/輸出浮點用整數格式化，勿用 `%f`

06

6. 相關文件

- 操作手冊：`guide-calibration-wizard.md`
- 備援機制：`guide-failover-resilience.md`
- REST API 全集：`../api/api-firmware.md`
- TCP IO 通道規格：`spec-tcp-io-channels.md`

07

7. 補充說明

- **磅秤接線語意**：`cmd/signal/<N>` 為訊號來源,可作 latched(持續閘控,值非 0 即啟用)或 pulse(上升緣觸發單次動作);磅秤校正以狀態機推進,門控決定 Converter 是否發重量。
- **RS485 寫入**：Converter 輸出可回寫 RS485 暫存器;32 位元值跨兩暫存器,讀寫不對需切換 Endianness(byte/word swap)。
- **單執行緒 HTTP 競爭**：韌體為單執行緒,RS485 輪詢與 inbound HTTP 共用同一迴圈,長掃描會延後 HTTP 回應;故輪詢逐 slave 讓出並餵看門狗,避免 HTTP 逾時與重啟。

04 開發者 / 整合

觸發式定期回報

📅 最後更新:2026-06-16 | 📌 負責人:KC

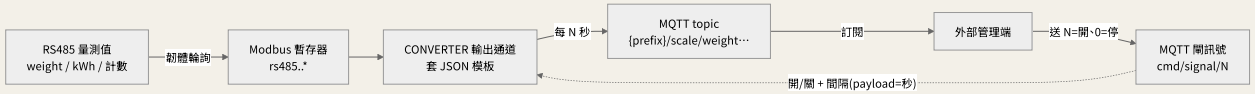
本文是「磅秤發重量 / 用電量發佈 / SF965 計米回報」這類需求背後**共用的通用機制**的權威說明。各裝置的可重現 recipe 見 §6 的實例連結;本文講「**精神、最小骨架、怎麼在 Web 介面做、怎麼變成工作流程頁一鍵卡片**」,看完任何 RS485 量測值都能自己兜出「外部下命令 → 設備每 N 秒回報」。

01

1. 精神:不為每種設備寫特製韌體

過去「磅秤發重量」一度想做成磅秤專屬韌體(WI-127),後來**全面廢除特製化**(WI-130),改成 **重用現有的通用機制**。核心只有三塊既有積木:

1. **RS485 來源** — 設備已在輪詢的 Modbus 暫存器(重量 / 用電量 / 計數值…)。
2. **CONVERTER 輸出通道** — 把暫存器值套進 JSON 模板,週期發到 MQTT topic。
3. **MQTT 閘訊號** — 外部發 `{prefix}/cmd/signal/{N}` 控制「發 / 停」與「多久發一次」。



✅ **沒有任何設備專屬韌體**。換一台新設備 = 換 Modbus 暫存器 map + 改 JSON 模板,流程骨架不變。

02

2. 最小骨架（純 level 閘）：1 個訊號 + 1 個通道

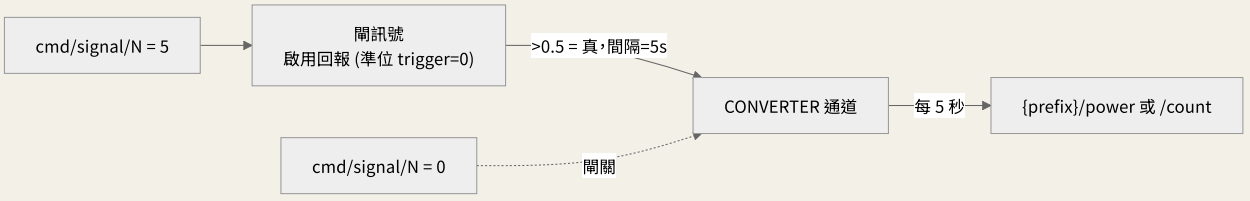
最單純的「按命令週期上報」只需兩個物件(用電量 / 計米器都走這個):

層	物件	設定
訊號	啟用回報 (閘)	來源 `type=6(CH_MQTT) index=N`, `op=4 threshold=0.5`, **`trigger=0`** (**`TRIG_ALWAYS`** ／**準位**)。讀 `{prefix}/cmd/signal/N`,>0.5 為真。
通道	週期回報 (CONVERTER)	`mode=2 direction=1`, `converterGateSignalId=<上面的訊號>`, `converterIntervalFromGate=true` (WI-143),JSON 模板取暫存器值。

> ⚠️ **閘訊號務必用準位 `trigger=0` (`TRIG_ALWAYS`)**,不要用 `CHANGE` (3)/ `RISING` (1)。準位才代表「持續開關」語意,也才會在工作流程頁自動呈現成 on/off 卡片(見 §5;實測: `CHANGE` 觸發的閘不會合成卡片)。

操作(payload = 間隔秒數,WI-143):

```
{prefix}/cmd/signal/N = 5   → 閘開，每 5 秒發一次
{prefix}/cmd/signal/N = 2   → 即時改成每 2 秒
{prefix}/cmd/signal/N = 0   → 閘關，停止
```



`converterIntervalFromGate=true` 時,**發佈間隔直接取閘訊號的 MQTT payload**(秒);設 `false` 則用固定的 `converterIntervalSec`。**送 `0` 一律停**。⚠️ 這些 `cmd/signal/*` **千萬別用 retain(`-r`)** —— retained 會在設備重連時反覆灌回、誤觸發。

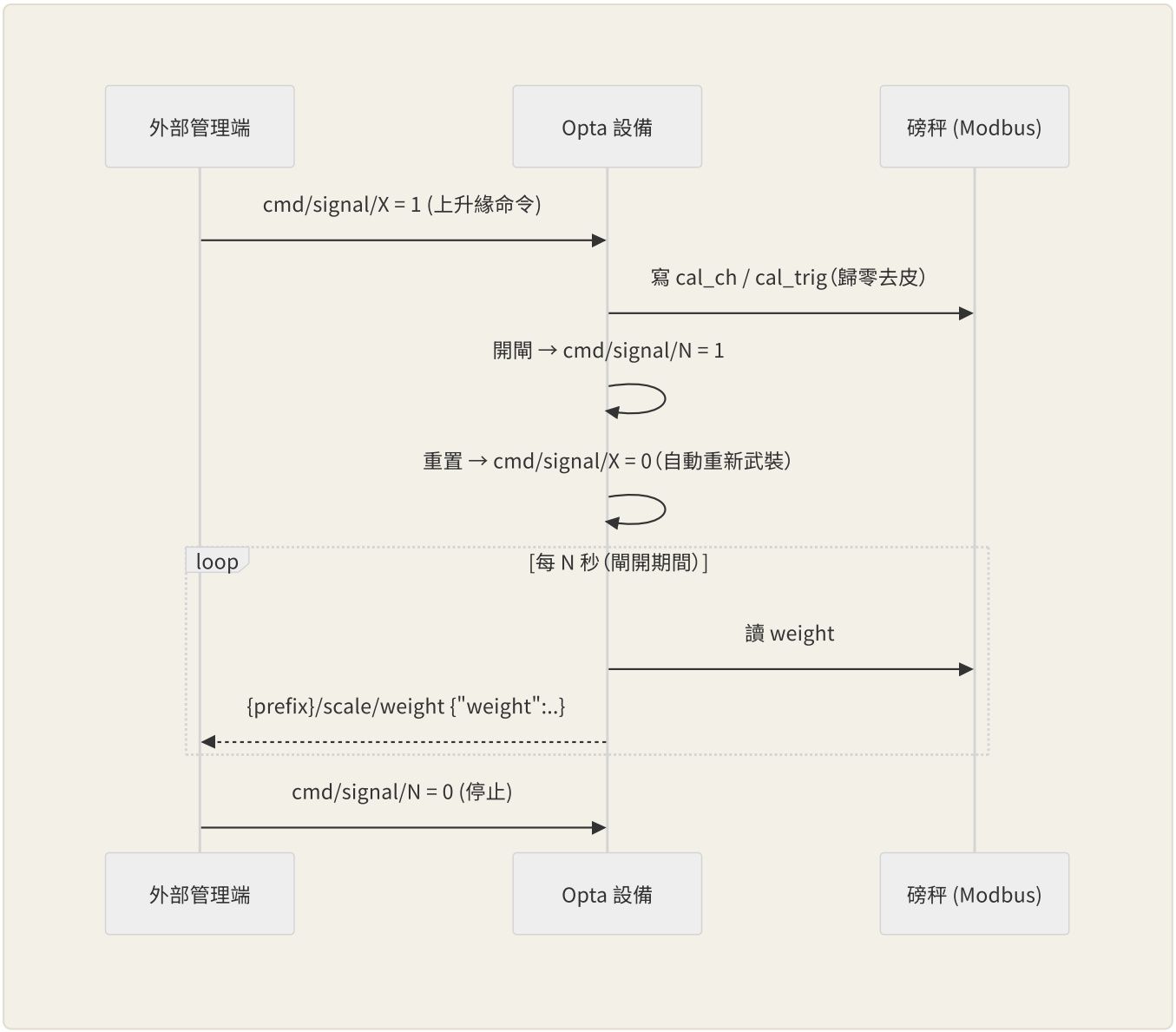
📌 **發佈「時機」三種模式**(並存,擇一):

模式	設定	行為
固定間隔	`converterIntervalSec`(網頁對話框預設)	每 N 秒固定發,例每 5 秒
payload=秒數(本文示範)	`converterIntervalFromGate=true`(WI-143)	送到閘的數字=幾秒發一次、送 `0` 停
變更時才發	`converterOnChange=true`(v5.9.263)	值有變才發、沒變不重複;間隔填 0=純變更

03

3. 進階骨架（命令 + 校正）：磅秤那種

當「上報前要先做一件事」(例如磅秤要先歸零去皮),就在最小骨架前面加一段 **上升緣命令 → 規則 → 動作**:



差別只是「閘」之前多了 edge-trigger 命令 + rules/actions + Modbus 寫入。**純上報需求不需要這段。**命令訊號的值會「黏著」,所以動作要回送 `0` 重新武裝 —— 因此送 `1` 可重複、不必手動先歸 0。

04

4. 在 Web 介面怎麼做（免命令列）

不想打 API 也行,整套都能在 Web Dashboard 點出來。兩步:

4.1 建「閘訊號」(規則頁)

到 **規則** 頁新增一個訊號:來源型別選 **MQTT 訊號**、index = `N`、條件 `> 0.5`、觸發模式 **準位 (ALWAYS／`trigger=0`)**。這就是「發/停」的閘 —— 用準位才會在工作流程頁變成 on/off 卡片(§5)。

規則頁 — 建立訊號 / 動作 / 規則

4.2 建「CONVERTER 輸出通道」(配置頁 → TCP I/O 通道)

到 **配置 → TCP I/O 通道** 新增一個通道:

- **方向 = 輸出、模式 = 轉換(CONVERTER)**。
- **MQTT 主題** = 你要回報的 topic(如 `…/scale/weight`、`…/power`、`…/count`)。
- **閘訊號** = 選 4.1 建的那個訊號。
- 勾 **「間隔取自閘 payload」**(`converterIntervalFromGate`)→ 送進閘的數字就是發佈秒數。
- **轉換模板** 填 JSON,用 `${rs485.<SlaveID>.<欄位>}` 取暫存器值,例 `{"weight":${rs485.<SlaveID>.weight}}`。

下圖為實機 OPTA-KC 的「電流監視」CONVERTER 通道:推送間隔 5 秒、門控信號=「啟用用電量發佈」、JSON 模板取 Finder 9 欄電力值:

配置頁 — CONVERTER 通道編輯(間隔 / 門控信號 / JSON 模板)

> 建好後不必重開機,送 `{prefix}/cmd/signal/N = 秒數` 即開始週期上報。

05

5. 工作流程頁：把這套流程變成「一鍵白話卡片」

建好「準位(`trigger=0`)閘訊號」後,**工作流程**頁(WI-128/129)會**自動**把它合成一張 **開關型卡片**(WI-130:無需規則,只要閘是準位觸發)—— 現場操作員不必懂訊號/規則/動作、也不必打任何 MQTT 命令,**直接撥 on/off 開關**就能開始 / 停止週期回報。

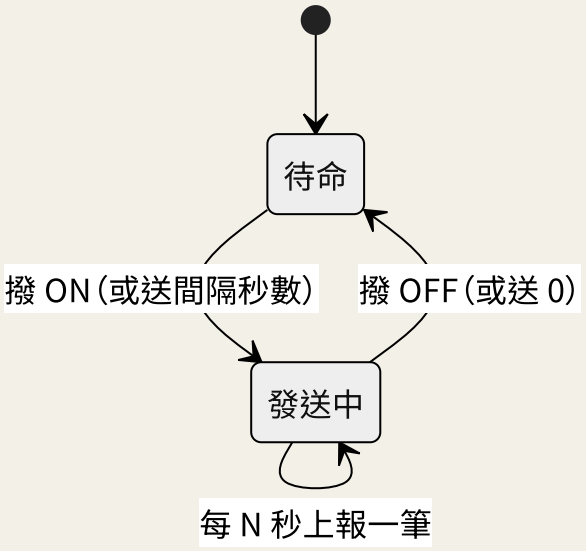
下圖為實機:一顆準位閘訊號即自動成為一張 on/off 卡片(「待命 / 已停止」):

工作流程頁 — 觸發式回報的開關型卡片

> ⚠️ **只有準位(`TRIG_ALWAYS`/0)的閘**會合成 on/off 卡片;若閘建成 `CHANGE`(3)/`RISING`(1),工作流程頁不會出現卡片(實測 confirmed)。要呈現成卡片,務必把閘訊號的觸發模式設為**準位**。

新流程也能直接從工作流程頁用**「＋ 新增工作流程」四步精靈**(命名 → 何時啟動 → 要做什麼 → 確認)建立:

工作流程頁 — 新增工作流程四步精靈



卡片角色	對應本文機制	操作
開關型(待命 / 發送中)	§2 的閘訊號 + CONVERTER 通道	撥 ON = 開始每 N 秒發、撥 OFF = 停
觸發型(▶ 執行,可填參數)	§3 的上升緣命令(如磅秤「需校正」)	按一下送一次命令

- 切 **ON** → 徽章「🟢 發送中」,下游每 N 秒送一筆;切 **OFF** → 「⚪ 待命」停止。
- 狀態由設備**即時回報**,多人 / 多裝置看到的開關狀態一致(不是只反映你按的那下)。
- 這取代了舊版「要記 topic、用 MQTT 工具手動發 `1` / `0` 」的做法。

> 📖 工作流程頁四種角色卡(觸發 / 開關 / 武裝監控 / 純被動)的完整操作見 **操作手冊 §10.5 工作流程頁操作**。

06

6. 裝置實例（可重現 recipe）

裝置	量測值	骨架	Recipe
TDA-08B 磅秤	weight / stable	進階(§3,含校正/去皮、上升緣命令)	見本文 §3 進階骨架
Finder 6M.TB 電力分析儀	9 欄電力 JSON	最小(§2,純 level 閘)	見本文 §2 最小骨架
SF965 計米器	當前計數值(米數)	最小(§2,純 level 閘)	套電力分析儀那套,換暫存器 map(見下)

SF965 計米器（套最小骨架）

SF965 沒有專屬流程,**完全比照 Finder 電力的 §2 最小骨架**,只換暫存器與模板:

- RS485 來源:slave `3`、`9600/8N1`、machineID `78`(`identifyDevice(78)` → SF965)。
- 暫存器:**當前計數值** `0x0000~0x0001`(40001~40002) `INT32` 唯讀。
  - ⚠️ 32 位元跨兩暫存器,讀出不對就換 Endianness:Modbus 同一 INT32 在不同設備可能用不同 byte / word 排列,需嘗試 byte swap 或 word swap 直到讀值正確。SF965 計數 raw 值另需乘 scale `0.00390625`(÷256)。
- 通道 JSON 模板(假設 count 對應 `rs485.<SlaveID>.count`):

```
{"count":${rs485.<SlaveID>.count}}
```

- 操作同 §2:`{prefix}/cmd/signal/N = 5` → 每 5 秒發 `{prefix}/count`;送 `0` 停。

07

7. 不必先知道 UID：whoami / info 探索（WI-131）

管理端要對全廠設備下命令、又不想預先記每台 UID:發**免前綴廣播** `mes/gateway/whoami` (或對單台發 `{prefix}/cmd/info`)→ 設備回 `{prefix}/info` (uid/name/fw/ip/online/counts)

- 每通道 `{prefix}/info/io/<id>` (含該通道的控制 / 資料 topic)。

```bash
H=mosquitto.smms.com.tw; P=8883; U=smms; PW=2ojujiru
mosquitto_sub --insecure -h $H -p $P -u $U -P $PW -t "mes/gateway/+/info" -t "mes
mosquitto_pub  --insecure -h $H -p $P -u $U -P $PW -t "mes/gateway/whoami" -m 1
```

> 對外整合合約(給第三方現場管理端)見 `../specs/spec-mqtt-integration.md` 。

08

8. 共同的坑（建流程必看）

1. **POST signal / action 一定要帶** `"enabled":true` —— 否則 API 預設 `false` ，訊號不評估、動作不執行，整條流程靜默不動(tcpio / rules 的 POST 本來就有帶)。
2. **compact JSON**:POST body 不能有空格(`separators=(',',':')`)，韌體用 `indexOf` 比對 `"key":value` 。
3. `cmd/signal/*` **絕不要 retain**;不慎發了用 `-r -n` 清除。
4. **寫入留 5s + 重試**:Opta 單執行緒，flash 存檔(WiFi-safe unmount/mount)會阻塞 6-12s，連續快速 POST 會 ConnectionReset。
5. **CH_TCP(output type 7)的 index 存 channelId**(WI-129-0a)，不是陣列位置;規則 GET 回讀欄位是單數 `signalId` / `actionId` 。
6. **payload = 間隔秒數**(WI-143， `converterIntervalFromGate=true`):送 N 每 N 秒、送 0 停。topic 用設備 **UID 前綴**(非 topicPrefix)。

04 開發者 / 整合

Modbus TCP 對應表

> 📅 最後更新：2026-05-11 | 📌 負責人：KC

此文件描述 MES I/O Gateway 的 **Modbus TCP** 雙向通訊規格：

- **Server (Slave)** 側：Gateway 開 listen socket，外部 HMI/SCADA 連進來讀寫內部 IO（§2-§4）
- **Client (Master)** 側：Gateway 主動連外部 Modbus TCP slave，把 register 值橋接到內部變數池（§7，2026-05-11 補上）

項目	值	備註
Server listen port	`502`（韌體預設）	PRD-Factory-Test v1.2 一度誤標 5000，以韌體 `Modbus TCP Server` 實作為準
Address scheme	Zero-based	詳見 §2
認證 / TLS	無	部署時靠網段隔離；未來看需求補

01

1. 架構特色 (v5.9)

為兼顧系統輕量性與最大擴充彈性，本專案採用 **0-based 固定位址對映 (Fixed Address Map)** 與 **Base-16 偏移法**，確保主機與未來擴充模組之位址空間絕不衝突。HMI 或 SCADA 得以直接對指定位址讀寫以完成控制與狀態蒐集。

02

2. 暫存器映射全域總覽 (Register Map)

類型代碼	通用名稱 (HMI 型號)	實際實體與對應位址	權限
0x	**Coils**	控制繼電器與 DO 輸出 (0-255)	R/W
1x	**Discrete Inputs**	讀取實體開關 DI 狀態 (0-255)	Read Only
3x	**Input Registers**	讀取類比訊號 AI 原始值 (0-255)	Read Only
4x	**Holding Registers**	系統內部變數池/快取區 (0-1023)	R/W

03

3. 實體 I/O 詳細位址分佈 (0x, 1x, 3x)

Gateway 主機與 5 組擴展模組的實體 I/O 透過**「每模組 16-bit 為一個區間」**進行無縫排列。若 HMI 需操作「Expansion 0」的第 1 個 DO，請寫入位址 `16 + 0 = 16`。

設備來源	Coils (0x) 位址 (寫/讀)	DIs (1x) 位址 (唯讀)	Input Regs (3x) 位址 (唯讀)
Host 主機	`0 ~ 3` (DO1-DO4)	`0 ~ 7` (DI0-DI7)	`0 ~ 7` (AI0-AI7, 乘16倍)
(保留)	*4 ~ 15*	*8 ~ 15*	*8 ~ 15*
擴充模組 0	`16 ~ 23`	`16 ~ 31`	`16 ~ 23`
擴充模組 1	`32 ~ 39`	`32 ~ 47`	`32 ~ 39`
擴充模組 2	`48 ~ 55`	`48 ~ 63`	`48 ~ 55`
擴充模組 3	`64 ~ 71`	`64 ~ 79`	`64 ~ 71`
擴充模組 4	`80 ~ 87`	`80 ~ 95`	`80 ~ 87`

> *[AI (Input Register) 補充說明]*: 內部 ADC 精度為 12-bit (0-4095) 轉換為 0-10V。為了與 16-bit 暫存器系統接軌，主機的 3x AI 數值已經乘以 16 倍，輸出範圍為 `0 ~ 65520` (等比例對應 0~10V)。

04

4. 虛擬數據位址分佈 (4x Holding Registers)

系統提供大量 (1024 槽位) 高速陣列 SRAM 空間供 Modbus TCP 存取，適合用作 **Rule Engine** 聯動目標或 **TCP Parser** 資料傾印。建議的規劃區段如下：

Holding Regs (4x)	推薦用途	實作現況 (v5.9)
`0 ~ 99`	內部系統控制變數池 (VAR0-VAR15)	支援以 Rule Engine 直接存取與覆寫。
`100 ~ 199`	**TCP 通道數據區段 (信號輸入)**	**[系統自動分配預設位址]** 給予那些在 TCP IO 被設定為「📥 輸入」的通道供外部 SCADA 寫入。
`200 ~ 499`	**TCP 通道數據區段 (狀態輸出)**	**[系統自動分配預設位址]** 給予那些在 TCP IO 被設定為「📤 輸出」的通道供外部 SCADA 讀取。
`500 ~ 1011`	*(保留供 Modbus RTU 或進階應用)*	目前作為暫存緩衝保留。
`1012 ~ 1023`	系統擴充保留區	保留供未來特定系統狀態使用。

4.1. TCP 通道 (TCP I/O) 暫存器配置原則

當通道選擇協議為 `Modbus TCP` 時，代表該通道會在 MES I/O Gateway 本機的 `4x Holding Registers` 空間中配置一塊區域，這使 Gateway 本身做為一個 **Modbus Server (Slave)** 供外部設備 (HMI / SCADA) 連線訪問。

配置規則 (自動指派機制)：

- **方向為「📩 輸入」**：如果您將 Holding Register 設為 `0` (自動分配)，韌體會從 `100` 開始尋找空位配發。外部圖控即可透過存取這段位址，將最新的狀態資訊寫入給 Gateway 以觸發內部規則（被動抓取信號）。
- **方向為「📩 輸出」**：如果您將 Holding Register 設為 `0` (自動分配)，韌體會從 `200` 開始尋找空位配發。外部圖控即可透過存取這段位址，讀取出本通道最新的產出資料或是經過 Converter 轉換完成的緩衝數據。

(註：若您有多字組長度需求，例如連續 8 個 Words，SCADA 直接對該起始分配位址進行連續區塊存取即可。)

05

5. 常見問答 (FAQ)

- **Q: HMI 面板要求打「偏移量」，我需要加 1，還是加 40001？** A: MES I/O Gateway 遵從工業標準的 `Zero-based Addressing`。請直接在「位址 (Address)」欄位填入本表列出的數字 (`0、16、32...`)。若您的軟體要求格式如 `40001`、`10017`，請注意那些通常是 `1-based` (加一偏移)，本系統一律以最純粹的 `Base-Address 0` 為設計核心。

06

6. 測試與驗證（2026-05-11 補）

由於缺乏實機 Modbus TCP 治具，採用 **pymodbus 模擬器** 跑端到端：

工具	用途
`scripts/modbus-sim/master_client.py`	模擬外部 SCADA / HMI，連到 Gateway :502 讀寫
`scripts/modbus-sim/slave_server.py`	模擬外部 Modbus TCP slave，給 Gateway 當 client 連
`scripts/modbus-sim/smoke_test.sh`	端到端 smoke test 腳本，產出缺口報告

可用上述腳本對 Gateway :502 做端到端讀寫驗證，`smoke_test.sh` 會產出涵蓋與缺口報告。

07

7. Client 側（Gateway 當 Master 連外部 Slave）

> 對應 firmware `modbusTcpTargets[]`、設定頁「Modbus TCP Targets」區塊（WI-043 / WI-051）。

Gateway 可同時對多個外部 Modbus TCP slave 設備發起連線並輪詢。每個 target 在配置中包含：

欄位	說明
`host`	slave 的 IP 或 hostname
`port`	通常 `502`
`slaveId`	Modbus unit ID（多數設備用 1）
`pollIntervalMs`	輪詢間隔（建議 ≥1000）
`registers[]`	要讀取的 holding/input register 清單

讀進來的值會橋接到內部變數池（VAR0~VAR15），可被 RuleEngine 當 signal source 使用。

7.1 限制與設計原則

- **單向讀取為主**：目前 Client 側只支援讀 (FC03/FC04)，寫入請走 RS485 路徑（透過 Converter 回寫 RS485 暫存器,32 位元值跨兩暫存器並需注意 Endianness）。Modbus TCP 寫入是後續延伸（暫無需求）
- **連線失敗 retry**：失敗會自動退避重試，連續 N 次失敗會在 UI 顯示告警
- **target 數量上限**：目前韌體預留 4 個 slot

7.2 Client telemetry（WI-117 Gap 5，v5.9.97+）

`GET /api/modbus-tcp/targets` 回傳的每個 target object 額外帶下列欄位（皆為 runtime，不存檔）：

欄位	型別	說明
`lastSuccessMs`	`uint32`	最近一次成功 poll 的 `millis()` 時間戳（0 = 從未成功）
`lastErrorMs`	`uint32`	最近一次失敗 poll 的 `millis()` 時間戳（0 = 從未失敗）
`successCount`	`uint32`	開機後成功 poll 累計數
`errorCount`	`uint32`	開機後失敗 poll 累計數
`nowMs`	`uint32`	韌體目前的 `millis()`，前端可用 `(nowMs - lastSuccessMs)` 算出「上次成功幾秒前」

POST `/api/modbus-tcp/targets` 改動目標設定時，所有上述計數會被歸零。

詳細 UI 操作流程見操作手冊 Modbus TCP 章節。

08

8. Server 側即時連線指標（WI-117 Gap 1，v5.9.97+）

`GET /api/system` 回傳新增 `modbusTcp` 物件：

```JSON
{
  "modbusTcp": {
    "port": 502,
    "running": true,
    "connections": 1,
    "requestCount": 1234,
    "errorCount": 2,
    "acceptedCount": 7,
    "rejectedUnitCount": 0,
    "lastRequestMs": 12345678,
    "lastClient": "192.168.51.10"
  }
}
```

欄位	說明
`connections`	目前活躍 socket 數（韌體單實例最多 1）
`requestCount` / `errorCount`	累計處理／錯誤請求數（含通訊錯誤與 exception 回應）
`acceptedCount`	累計接受連線次數（重連會 +1）
`rejectedUnitCount`	因 §9 unit-id 政策被拒絕的請求數
`lastRequestMs`	最近一次成功處理的 `millis()`（前端可換算「N 秒前」）
`lastClient`	最近一次接受的對端 IP，UI 顯示在 dashboard `🔌 Modbus TCP` 卡片

09

9. Unit ID 政策（WI-117 Gap 3，v5.9.97+）

Modbus TCP 慣例：

Unit ID	政策	韌體行為
`0`	broadcast	**靜默丟棄**，不回應
`1`	primary slave	接受並處理
`2`~`254`	對應其他下游 slave	回 exception code `0x0B` (Gateway Target Device Failed to Respond)
`255`	TCP-only sentinel / "any"	接受並處理

被拒絕的請求會累加到 `/api/system → modbusTcp.rejectedUnitCount`，方便 SCADA 整合排錯。

04 開發者 / 整合

TCP I/O 通道

📅 最後更新：2026-03-14 | 📌 負責人：KC

TCP I/O Channels 讓 MES Gateway 能將網路型的資料串流橋接進統一的 Rule Engine，把遠端的 topic 或 register 當成本地的 signal 與 actuator 來處理。

01

1. 雙管線架構 (v5.9)

從 5.9 版開始，TCP I/O 系統從單純的「Direct Mapping」轉變為雙管線架構，以支援複雜的 MES/SCADA 整合。

A. 入向資料管線 (Parser)

擷取遠端資料並將其轉換為內部的 signal 狀態。

- **Protocol**：主要為 MQTT (Subscribe) 或 Modbus TCP (Read)。
- **擷取策略 (Extraction Strategies)**：
  - **Boolean/String Match**：判斷 payload 是否符合目標字串(例如 `"START"`)。
  - **Numeric Parsing**：從原始 payload 或特定 JSON 欄位中擷取 float 數值。
  - **JSON Field Extraction**：使用手動子字串搜尋或輕量解析，在複雜的 JSON 主體中定位 key。
- **持久化 (Persistence)**：保留「最後已知值 (Last known value)」,並可選擇性搭配 **Stale Timeout**(若在 N 秒內沒有收到更新，便將 signal 標記為非作用中)。

B. 出向資料管線 (Converter)

使用自訂格式組裝裝置資料並推送至遠端系統。

- **Protocol**：主要為 MQTT (Publish) 或 Modbus TCP (Write)。
- **以 Template 為基礎的組裝 (Template-Based Assembly)**：
  - 使用簡單的關鍵字替換引擎: `${...}`。
  - **Variable Tags**：`${io.*}`、`${vc.*}`、`${rs485.*}`、`${sys.*}`。
  - **工作流程 (Workflow)**：使用者在 Web UI 中定義一份原始 JSON template。執行時，gateway 會以即時資料填入該 template 並發佈。
- **非阻塞邏輯 (Non-Blocking Logic)**：組裝與傳輸皆以非同步方式執行，以避免迴圈延遲。

02

2. MQTT 實作與函式庫遷移

2.1 遷移至 `arduino-mqtt` (256dpi)

為了滿足工業級可靠度需求，gateway 從 `PubSubClient` 遷移到 `arduino-mqtt` 函式庫。

- **主要驅動因素**：原生支援發佈與訂閱兩端的 **QoS 1** 與 **QoS 2**。
- **預設策略 (Default Policy)**：控制命令與狀態更新預設使用 **QoS 2** (Exactly Once)，以避免訊息重複或遺失。

2.2 關鍵的標頭衝突：`MesMQTT.h`

在遷移過程中發現了一個命名衝突:函式庫內部的標頭 `MQTTClient.h` 與 gateway 自有的 `src/MQTTClient.h` 相衝突。

- **解法 (Resolution)**：gateway 的 wrapper 類別更名為 **`src/MesMQTT.h`**。
- **Include Guard**：改為 `MES_MQTT_H`。
- **命名空間備註 (Namespace note)**：此舉可避免 `mbed` 或其他函式庫的 include 在無意間載入到本地 wrapper，而非預期的函式庫類別。

2.3 Callback 多載歧義與靜態邏輯

`arduino-mqtt` 函式庫為 `onMessageAdvanced` 提供了多個多載版本(函式指標 vs. `std::function`)。

- **難題 (Challenge)**：初版實作中的 lambda 在編譯時造成了多載歧義錯誤。
- **解法 (Solution)**：將訊息處理移至一個 **靜態全域函式** (`static void mqttMessageHandler`)。
- **結構 (Structure)**：
  1. 在標頭中宣告一個 `static void` handler。
  2. 使用全域指標 `g_mqttClient`,從靜態情境橋接回 `MQTTClientManager` 實例。
  3. 從靜態 handler 呼叫 `g_mqttClient->handleMessage()`。

03

3. Modbus TCP 實作

3.1 Register 對應的一致性

TCP I/O 點位一律對應到高位址的 Holding Registers (4xxxx)，以避免干擾實體 I/O 或系統變數。

- **Inputs (Registers 100-199)**：入向 (Inbound) channel 的唯讀狀態。
- **Outputs (Registers 200-299)**：出向 (Outbound) channel 的「寫入即致動 (write-to-actuate)」register。

3.2 雙角色設定 (Dual-Role Configuration)

Channel 可設定為 **Server** (被動) 或 **Client** (主動)。

- **Passive Input**：gateway 等待外部 PLC 寫入其 register。
- **Active Input**：gateway 主動輪詢遠端 PLC 的 IP/Register，並將其對應到一個 Signal Source。

04

3. 實作模式 (v5.9)

3.1 Converter：Template 組裝生命週期

出向管線採用「Lazy Load」模式以節省 RAM:

1. **Trigger**：由 Rule Action 或 Interval Timer 觸發 Converter。發佈「時機」有三種模式並存:(a) 固定間隔(`converterIntervalSec`,網頁對話框預設,例每 5 秒);(b) payload=秒數(`converterIntervalFromGate=true`,WI-143;送到閘的數字=幾秒發一次、送 0 停);(c) **變更時才發**(`converterOnChange=true`,v5.9.263 新增;值有變才發、沒變不重複;間隔填 0=純變更)。
2. **Load**：引擎從 QSPI Flash 開啟對應的 template 檔案 (`/fs/tmpl/{id}.json`)。
3. **Buffer Scan**：引擎將 template 讀入一個暫時的 1024-byte buffer。
4. **Replace**：非阻塞的關鍵字引擎搜尋 `${...}` tag 並進行替換:
   - `${io.I1}` → `1` 或 `0`
   - `${vc.CNT1}` → `42.00`
   - `${rs485.<SlaveID>.V}` → `220.5` (其中 `<SlaveID>` = 該 RS485 設備的 Modbus Slave ID;v5.9.262+ 起 N 為 Slave ID，**非陣列索引/第幾台**，刪或重排設備都不會跑掉)
5. **Dispatch**：組裝完成的 JSON 被推送至 MQTT/Modbus 傳輸層。
6. **Cleanup**：buffer 在派送後立即清空;不保留任何長期的 heap 配置。

3.3 原始 Payload 直通 (`${raw_payload}`)

為了支援「處理後轉發 (Process-and-Forward)」情境——gateway 必須在附加自身 metadata 的同時保留原始的入向 JSON 結構:

- **Buffer**：全域 `rawPayload[257]` buffer 儲存 MQTT client 全域收到的最新一則訊息。
- **Resolution**：當 Converter Engine 遇到 `${raw_payload}` 時，會原封不動地插入此 buffer 的內容。
- **使用情境 (Use Case)**：讓 gateway 能扮演「Decorator」角色,接收 sensor 的 JSON、加上 timestamp 或本地 IO 狀態，再轉發給上游 MES,而不需重寫原始 schema。

3.4 小數位數修飾詞 (`:N`，v6.0.5+)

數值型變數(`${rs485.*}` 等)可在變數名後加 `:N` 指定輸出小數位數,由模板作者**逐模板**控制發佈格式:

- `${rs485.1.計數值}` → `106.99` (**不加 = 預設 2 位**,相容既有模板)
- `${rs485.1.計數值:0}` → `107` (**0 位 = 整數**,四捨五入;計數器適用)
- `${rs485.1.計數值:1}` → `107.0`;`${rs485.1.計數值:3}` → `106.990` (補零)
- 範圍 0–6 位;以整數運算實作(newlib-nano 無 `%f`),見 `ConverterEngine::_formatFloat`。

> 只影響 **Converter 發佈的 payload**;設備網頁 RS485 卡片的顯示為另一路徑。通用的「每暫存器預設小數位數」設定為後續工作(同時套顯示 + 作為模板未指定 `:N` 時的 fallback,0=整數)。
> ⚠ `:N` 需韌體 **6.0.5+**;舊韌體會把 `計數值:0` 整串當暫存器名 → 找不到 → 回 0。

3.2 Web UI Template 編排 (v5.9.3)

為了在工程師的可讀性與 MCU 的儲存限制之間取得平衡，Web UI 實作了一套 **Format-on-Edit / Minify-on-Save** 策略:

- **Pretty Print**:載入 template 時，前端會自動將其格式化為人類可讀的 JSON。
- **Placeholder 變通做法**:標準的 `JSON.parse` 在含有 `${...}` placeholder 的 template 上會失敗。前端使用一套 **Tokenization 變通做法**:
  1. 暫時將所有 `${...}` 替換為唯一識別字串(例如 `"__PH0__"`)。
  2. 執行 `JSON.stringify(..., 2)` 以格式化文字。
  3. 還原原本的 placeholder。
- **自動最小化 (Auto-Minification)**:當使用者點擊 `Save` 時，前端會在將 payload 送往 `/api/converter/template` 端點之前剝除所有空白與換行,確保每份 template 在 LittleFS 上佔用的空間最小。

3.2 Parser：入向 Signal 對應

入向管線同時支援原始與結構化資料:

- **Simple Match**：若 `mqttField` 為空，則整個 payload 會與 `mqttMatchValue` 比對。
- **JSON Extraction**：
  1. 在 payload 中定位 `mqttField` key。
  2. 同時處理 String(帶引號)與 Numeric(不帶引號)兩種值。
  3. 更新與該 Signal Source 關聯的內部 `currentState`。
- **安全性 (Safety)**：若 parser 遇到格式錯誤的 JSON，會維持 **最後有效狀態 (Last Valid State)**,以避免規則振盪。
- **UI 分組 (v5.9.6)**：Web UI 透過讓使用者定義單一 MQTT Topic、再多次「Add Extraction Field」的方式，簡化了眾多 parser channel 的設定。
  - **對應 (Mapping)**：在 API POST 操作期間,前端會自動將此群組展開成個別的 TCP IO channel(每個欄位一個)。
  - **實作 (Implementation)**：`saveTcpIo()` 函式會走訪 `parserFields` 陣列，並依序發出 `POST /api/tcpio` 呼叫。
  - **編輯模式 (Modification Pattern)**：在 Edit Modal 中，目前 UI 透過將既有單一 Parser channel 的屬性對應進分組式的 UI 結構，以一致的方式支援其編輯。

05

4. 技術限制 (v5.9.6)

- **容量 (Capacity)**：最多 **64 個 TCP I/O channel**(從 8 個擴充而來，以容納高密度的 JSON 解析)。
  - **記憶體佔用 (Memory Footprint)**：從 8 個擴充到 64 個 channel,約增加 **11KB** 的 RAM 與 Flash 合計用量。在 Arduino Opta (512KB) 上的 RAM 使用率約為 **15.8%**。
- **Buffer 大小 (Buffer Size)**：MQTT 讀/寫 buffer 設為 512 bytes;Converter 組裝 buffer 設為 1024 bytes;全域 `rawPayload` buffer 設為 256 bytes 再加上 null terminator(**257B**)。
- **關鍵修正 (v5.9.1 - v5.9.6)**：
  - **JSON 截斷 (JSON Truncation)**：在 MQTT 訊息 callback 中，內部 `valStr` buffer 從 48 bytes 增加到 **256 bytes**,以避免複雜 JSON payload 被截斷。

- **Verify 支援**：`GET /api/tcpio` 現在明確包含 `currentStrValue`，以支援對 String 型 Parser channel 的驗證。

- **Mode 與 UX 重構 (v5.9.7)**：
  - **簡化的 Mode 名稱**：`⚡ 收取並觸發動作 (Legacy)`、`📩 收取並擷取數值 (Parser)`、`📩 組合並發佈訊息 (Converter)`。
  - **Mode 驅動的自動化 (Mode-Driven Automation)**：在 MQTT protocol UI 中隱藏「Direction」欄位(Input/Output)，以縮小設定錯誤的面。它會依 mode 自動設定：
    - **Legacy / Parser** → 強制 **Direction = Read (Input)**。
    - **Converter** → 強制 **Direction = Write (Output)**。
    - *備註：對 Modbus TCP 而言，Direction 仍然可見且可手動選擇。*
  - **動態 Parser 欄位 (API Explosion Pattern)**：Web UI 透過讓使用者定義單一 MQTT Topic、再多次「Add Extraction Field」的方式，簡化了眾多 parser channel 的設定。
    - **UI 到 API 的對應 (UI-to-API Mapping)**：在 `saveTcpIo()` 操作期間，前端透過依序發出 `POST /api/tcpio` 呼叫，將此邏輯群組「展開 (explodes)」成個別的 TCP IO channel。這讓複雜的多欄位設定能由韌體穩定的 1-to-1 channel 邏輯處理，而無需大幅變更 API schema。
  - **編輯驗證的 ID 一致性 (ID Consistency for Edit Validation)**：修正了一個在編輯時名稱重複檢查失效的 bug。驗證邏輯 (`checkDuplicateName`) 必須使用正確的內部識別字 `tcpio_{idx}`，以將目前 channel 排除在碰撞檢查之外。(於 v5.9.7 從舊有的 `tcp_{idx}` 修正)。

- **變數參照最佳化 (v5.9.7)**：
  - **Topic 分組 (Topic Grouping)**：為了管理 64-channel 的密度，`${tcpio.x}` tag 會依 **MQTT Topic** 標題區塊進行分組。
  - **版面 (Layout)**：使用 `flex-wrap` 與 `gap: 6px`，確保參照 tag 在各種螢幕尺寸上能自然流排，避免 UI 破版。
  - **視覺提示 (Visual Cues)**：使用 Emoji (⚡、🔢、📡、🔌) 來區分實體 vs. 虛擬 vs. 網路型 signal。
  - **開發者體驗 (Developer Experience)**：tag 套用 `cursor: pointer` 樣式並具備一鍵複製到剪貼簿的功能，以提升 template 撰寫速度。

06

5. 工業生命週期模式

5.1 動態 Topic 訂閱

為了支援免重開機的執行時設定，gateway 為 MQTT 訂閱實作了一套「Hook-on-Save」模式:

1. **POST Trigger**：當 `/api/tcpio` 收到一筆新的 Parser/Input 設定時。
2. **立即動作 (Immediate Action)**：在存檔至 LittleFS 後，若 broker 已連線，handler 會明確呼叫 `mqttClient.subscribeTopic()`。
3. **持久化 (Persistence)**：該 topic 也會被加入 `trackedTopics` 清單,以確保在後續 broker 重新連線時能自動重新訂閱。

04 開發者 / 整合

連線與協議

> 📅 最後更新：2026-03-14 | 📌 負責人：KC

本文件涵蓋 MES I/O Gateway 使用的網路基礎設施與通訊協議。

01

1. 網路抽象層（`NetworkAdapter.h`）

Gateway 透過統一抽象層支援雙堆疊網路連線（Ethernet 和 Wi-Fi），確保高階服務（Web Server、MQTT、Modbus）與實體媒介解耦。

執行期模式選擇（有線優先）

Gateway 使用「有線優先（Wired-First）」偵測邏輯來降低現場配置複雜度：

- **Ethernet 優先**：開機時系統檢查實體 Ethernet 連結（`network.hasEthernetLink()`）。偵測到網線即初始化 Ethernet（DHCP/Static），同時**停用 Wi-Fi** 以減少干擾與電源雜訊。
- **Wi-Fi 備援**：若未偵測到 Ethernet 網線或初始化失敗，系統會嘗試連線已設定的 Wi-Fi 存取點。
- **介面狀態**：Web UI 的設定面板根據執行期偵測結果顯示目前使用的介面（"Ethernet" 或 "WiFi"）。
- **統一韌體 (v5.0)**：Ethernet 和 WiFi 堆疊編譯到同一個二進位檔（`env:opta`）。這實現了「部署中立性」——同一韌體可部署至任何 Opta 裝置，自動適應可用的實體基礎設施。

核心介面

- `NetClient` / `NetServer`：網路 socket 的共用型別定義（相容 Ethernet 和 WiFi）。
- `NetworkAdapter` **類別**：多媒介連線的統一處理器。

- `hasEthernetLink()`：硬體端檢查實體網線。
- `activeNetwork()`：回傳 `NET_ETHERNET`、`NET_WIFI` 或 `NET_NONE`。
- `beginEthernet(mac, ip, gw, sn)`：初始化有線靜態 IP。
- `beginEthernetDHCP(mac)`：初始化有線 DHCP。
- `setEthernetIP(ip, gw, sn)`：動態更新有線介面。
- `beginWiFi(ssid, pass)`：初始化無線連線（v5.4.7 從 `begin` 改名以避免歧義）。
- `disconnectWiFi()`：中斷無線連線（v5.4.7 從 `disconnect` 改名）。
- `setAPMode(enabled)`：標記系統進入存取點模式。
- `isAPMode()`、`isWiFi()`、`isEthernet()`：介面識別。

- **`NetworkConfig`**：持久化 IP 設定、DHCP 偏好、Wi-Fi 憑證。自 v5.1 起，**出廠預設改為 DHCP 啟用**以利無縫初始探索。

1.4 初始化順序 (Industrial Priority)

為支援韌性，`setup()` 順序優先載入配置與安全機制檢查，在任何網路流量開始前完成。自 **v5.4.24** 起，初始化順序經過重構，將 AP Mode 的資源與所有潛在的硬體衝突（DMA/SPI/中斷）隔離。

1. **`LocalConfig::begin()`**: High-priority mount of QSPI Flash. Sets the ground for all subsequent operations.
2. **Configuration Check**: `localConfig.loadLocal(config)` is called once.
3. **AP Mode Pre-emption (v5.4.24 Critical Fix)**: Checks `config->network.forceApMode`. If `true`:
   - **Resource Gating**: Skip initialization of Modbus RTU, I/O Manager, Virtual Devices, and MQTT.
   - **Isolation**: Call `WiFi.end()` followed by a `delay(2000)` to reset the radio module.
   - **QSPI Release (Final Blocker Fix)**: Call `localConfig.unmount()` to explicitly close LittleFS and the QSPI BlockDevice. This is necessary because the Murata 1DX radio and QSPI Flash share internal bus resources on the Opta's STM32H747 core.
   - **Hotspot Start**: Call `WiFi.beginAP("MES-Gateway-Setup", "12345678")`.

- **Early Exit**: Start `server.begin()` and **return** from `setup()`.
- *Result*: This multi-stage isolation prevents the `WiFi.beginAP()` hang by ensuring no other high-load hardware managers are holding onto shared resources during the radio's sensitive initialization phase.

4. **Standard Mode Path**: If not in AP mode, proceed with:
   - `checkSafeModeButton()`: Component-level fail-safe.
   - I/O Manager, Modbus RTU, MQTT initialization.
   - `network.hasEthernetLink()` detection and network stack start.
   - Main loop execution.

1.5 AP Mode Web Server（安全區域）

當系統處於 AP Mode 時，Web Server 以最小路由集初始化以避免記憶體壓力。技術人員可透過 **192.168.3.1** 存取 dashboard 進行 WiFi SSID 掃描與憑證輸入。

1.6 建置系統需求（v5.4.1 發現）

要在統一韌體中啟用 WiFi 管理子樹（AP Mode、掃描、靜默配置），需在建置環境配置（如 `platformio.ini`）中定義以下巨集：

- **`USE_WIFI`**：啟用 `main.cpp` 中 `/api/wifi/*` 和 `/api/network/apmode` 端點的路由。

若未定義此旗標，Gateway 將預設為「Ethernet-Primary」狀態，WiFi 管理邏輯不會編入二進位檔，以在非無線部署中節省記憶體。

02

2. Wi-Fi 支援 (v4.9.5)

Arduino Opta Wi-Fi 變體已完整支援，具備執行期配置能力。

功能特性

- **統一邏輯**：程式碼中 50+ 個 network client/server 實例已重構為使用 `NetClient` / `NetServer` 抽象。
- **Dashboard UI**：設定 tab 中有專用的「Wi-Fi 設定」卡片，支援 SSID 掃描、憑證輸入和連線管理。
- **API Support**:
  - `GET /api/wifi/status`: Reports connection state, SSID, IP, and RSSI.
  - `GET /api/wifi/scan`: Triggers a hardware scan for available networks (Note: Blocked while Ethernet is active).
  - `POST /api/wifi/save`: (v5.3 New) Updates credentials in Flash without initiating a connection. Optimized for silent provisioning while on Ethernet.
  - `POST /api/wifi/connect`: Updates credentials and attempts a heartbeat connection.

2.1 「控制路徑」挑戰 (v5.2/v5.3)

實際工業測試發現一個關鍵硬體限制：Arduino Opta 在 Ethernet 連結啟用時無法可靠地掃描 WiFi。

- **硬體限制**：在高負載多媒介情境下，掃描常回傳錯誤。
- **控制路徑風險**：若系統主動解除 Ethernet 以執行 WiFi 掃描，會同時中斷 Web UI 管理介面，對操作員形成「死路」。
- **v5.3 解法：靜默配置**：
  - **「僅儲存 WiFi」UI**：允許透過 Ethernet 輸入憑證但不立即連線。
  - **AP Mode 備援**：若未偵測到 Ethernet 且未設定 WiFi，裝置自行託管 SSID（192.168.3.1）供設定。
  - **重啟驗證**：引導式工作流觸發裝置重置以驗證無線連線，確保操作員知道設定是否成功。
- **v5.4 手動 AP Mode 切換**：當 Ethernet 啟用但使用者需要掃描 WiFi 時，系統實作手動 `POST /api/network/apmode` 觸發。無論當前連結狀態，強制裝置進入 AP 設定狀態（MES-Gateway-Setup），啟用被活躍 Ethernet 堆疊阻擋的完整硬體掃描功能。

2.2 WiFi 模組韌體依賴 (v5.4.7 重大)

實際部署發現 Arduino Opta 的 WiFi 無線電模組 (Nina-W10) 需要儲存在專用 QSPI 分區的特殊韌體檔案系統。

- **症狀**：Serial 錯誤 `Failed to mount the filesystem containing the WiFi firmware`。
- **原因**：無線電韌體不是標準應用程式二進位檔的一部分，可能在批量格式化或恢復出廠設定時遺失。
- **重要警告 (v5.4.12)**：即使 `WiFiFirmwareUpdater` 回報成功，若存在 **QSPI 分區衝突**，錯誤仍可能持續。應用程式的 LittleFS 資料儲存可能與 WiFi 分區重疊。
- **解法**：使用 Arduino IDE 的 **`WiFiFirmwareUpdater`** sketch。若錯誤持續，先使用 **`QSPIFormat`** 範例 sketch 重設分區表。
- **重要**：`QSPIFormat` 會清除無線電韌體。您**必須**在 `QSPIFormat` *之後立即*再次執行 `WiFiFirmwareUpdater`，且在*重新上傳專案之前*。**注意**：PlatformIO 目前尚未原生支援 Opta 核心的此特殊分區管理。

---

3. MQTT 整合

MQTT 提供事件驅動、低延遲的遙測功能，用於 MES/SCADA 系統。

實作：`MQTTClientManager` (v5.9)

- **Library**: `arduino-mqtt` (256dpi) [取代 PubSubClient]。
  - **原因**: 原生支援 QoS 0/1/2 publish 與 subscribe，支援重要控制指令的 Exact-delivery。
- **架構優化 (Resilience)**:
  - **Header 命名衝突迴避 (Critical)**: 為了避免與 library 內部的 `MQTTClient.h` 衝突，Gateway 的包裝類別已重命名為 **`src/MesMQTT.h`**。
  - **陷阱**: `arduino-mqtt` 的 `MQTT.h` 會 `#include "MQTTClient.h"`。若專案中存在同名檔案，編譯器會優先載入本地檔案而非 library 內部的類別定義，導致 `'MQTTClient' does not name a type` 錯誤。
- **靜態 Callback (Static Handler)**: 使用靜態函式 `mqttMessageHandler` 接收進站訊息，避免了 Lambda 在 `onMessageAdvanced` 重載 (std::function vs function pointer) 時引發的

編譯歧義。

- **全域預設 QoS**: 預設採用 **QoS 2** (Exactly Once)，大幅提升控制指令的可靠性。
- **Topic Hierarchy**:
  - `mes/gateway/status`: LWT (online/offline)。
  - `mes/gateway/di/{index}`: 狀態變化推送。
  - `mes/gateway/do/{index}`: 狀態變化推送。
  - `mes/gateway/rules/{index}/trigger`: 規則觸發通知。
  - `mes/gateway/{clientId}/cmd/#`: 遠端指令訂閱。

3.2 連線生命週期與韌性 (v5.9)

為防止主應用程式迴圈在網路不穩定期間阻塞，MQTT 管理器實作了自動化生命週期：

1. **初始化**：`begin()` 綁定 broker 並註冊靜態 callback。此步驟在 `setup()` 早期執行。
2. **網路就緒觸發 (延遲連線)**：`connect()` 被刻意從 `setup()` 移除，改為在 `loop()` 中偵測到有效的 Ethernet/WiFi IP 地址及 LinkUp 狀態後才觸發。
   - **原因**：防止在網路堆疊初始化過程中嘗試 TCP 連線導致的 Block 或 HardFault，並確保所有的「Connecting to...」日誌能完整輸出到已穩定的 Serial Console，方便現場除錯。
3. **迴圈與重試**：
   - 若斷線，系統使用**指數退避** (10s、20s、30s) 重試。
   - **自動停用 (重大)**：若連續 **3 次**連線失敗，明確設定 `enabled = false`。
   - **故障排除信號**：自動停用時，Serial 日誌印出：`MQTT: Auto-disabled after 3 failures. Fix config in Web UI.` 在透過 API 更新配置之前不再嘗試連線，節省 CPU 週期並防止網路堆疊飽和。

3.3 MQTT 故障排除指標

- **開機無日誌**：若 Serial monitor 未顯示「MQTT: Broker」訊息，透過 `GET /api/config` 確認 `config->mqtt.broker` 非空。
- **認證失敗**：檢查 Serial 日誌中的 `rc` (Return Code)：
  - `rc=4`：使用者名稱/密碼錯誤。

  - `rc=5`：未授權。
- **網路失敗**：`err=-1` 通常表示逾時或 broker IP 不可達。

3.4 MQTT TCP IO 通道 (v5.6.5+)

Gateway 可將特定 MQTT topic 視為邏輯層級的 IO 點。

- **入站**：訂閱已設定的感測器。對 payload/JSON 欄位與匹配值進行評估。
- **出站**：當觸發發生時發佈填空樣板。
- **詳細技術總覽**：見 `spec-tcp-io-channels.md`。

03

4. Modbus TCP 伺服器 (v4.5)

為了與傳統工業系統互通，Gateway 在 Port 502 作為 Modbus TCP Server 運作。

架構最佳化

為防止「Port 80 飢餓」（Modbus 流量阻塞 Web Dashboard），伺服器使用嚴格的非阻塞模式：

- **無延遲**：移除請求處理迴圈中的阻塞 `delay()` 呼叫。
- **機會性輪詢**：每個系統週期恰好處理一個封包。
- **Socket 效率**：明確清空並關閉 socket 以釋放資源給 HTTP 流量。

暫存器映射（v5.6.5 擴充）

- **FC 01/02**：讀取實體 DI (0-7) 和 DO (0-3)。
- **FC 03/04**：讀取縮放後的類比輸入、系統變數（`VAR[0..15]`）及 **TCP IO 輸入 (Registers 100-199)**。
- **FC 05/06**：寫入數位輸出、系統變數及 **TCP IO 輸出 (Registers 200-299)**。

4.1 Modbus TCP IO 通道 (v5.6.5)

Gateway 將內部 TCP IO 狀態映射到專用的 Holding Register 區塊：

- **輸入 (Registers 100-199)**：映射到 TCP IO 輸入通道的 `currentState`。
- **輸出 (Registers 200-299)**：寫入 `1` 到這些暫存器會觸發對應的 TCP IO 輸出動作（如發佈 MQTT 指令）。
- **自動分配**：配置 API 處理暫存器分配以防止與其他系統變數衝突。

04

5. 開發進度

- **網路抽象層**：完成。
- **MQTT (Pub/Sub)**：完成並驗證 (v4.9.3)。
- **Wi-Fi UI 與 API**：完成並驗證 (v4.9.5)。
- **v5.3 WiFi 配置**：UI 重構完成（舊版掃描改為引導式手動輸入）。
- **Modbus TCP**：邏輯已實作；Socket 管理已最佳化 (v4.5)。
- **Modbus RTU (RS485)**：Master/Slave 支援已整合 (v4.3)。

05

6. Modbus RTU (RS485) 管理器 (v4.3)

RTU 管理器利用 Opta 內建的 RS485 終端與傳統工業硬體介面。

6.1 雙角色能力

- **Master 模式**：實作非阻塞的循環輪詢迴圈，支援最多 8 個外部 slave（`MAX_MODBUS_DEVICES = 8`）。
- **Slave 模式**：透過 16-bit Holding Registers 的位元向外部 PLC/SCADA 暴露 Gateway 狀態。

6.2 技術規格

- **介面**：`Serial1`（RS485）。
- **預設配置**：19200 Baud、8N1、標準 CRC-16。
- **暫存器映射（Slave 模式）**：
  - Addr 0：DI 狀態位元遮罩。
  - Addr 1：DO 狀態位元遮罩。
  - Addr 2-9：縮放後 AI 電壓值（×100）。
  - Addr 10-11：32-bit 系統運行時間（秒）。

6.3 RS485 韌性模式 (v5.8.5)

為確保從高密度量測裝置（如 Finder 6M）可靠取得資料，系統實作了專用的輪詢策略。

分段讀取批次策略

某些 Modbus 裝置對單次批次可讀取的暫存器數量有限制。超過此限制（如請求 22 個暫存器但限制為 18）會觸發逾時。

- **實作**：`PowerMonitor` 管理器執行兩階段讀取：
  1. **主要（Registers 192-209）**：高頻資料（V、I、P、Hz）。
  2. **擴充（Registers 210-213）**：累計計數器（Energy±）。

displayMask 與資料擷取的解耦

為平衡 UI 效能與資料完整性，顯示偏好與背景擷取解耦。

- **displayMask**：由 Web UI 用來隱藏/顯示量測卡片。
- **背景完整性**：韌體始終從 slave 輪詢所有欄位，確保 Data Logger 和 API 取得連續資料流，無論 dashboard 上顯示什麼。

UI 防禦邏輯

RS485 配置 modal 附加到 `document.body` 以避免在巢狀 CSS 版面中被裁剪。RS485 區塊的可見性邏輯以資料存在與否為範圍，確保在主機和擴充模組 tab 之間切換時元素保持可見。

統一韌體建置 (v5.0)

自 v5.0 起，Ethernet 和 WiFi 的獨立建置環境已廢棄。單一的 `env:opta` 建置針對 Opta WiFi 硬體，動態管理兩個介面。

- **依賴**：`Ethernet.h` 和 `WiFi.h` 包含在同一個二進位檔中。
- **邏輯**：有線連線始終優先。WiFi 僅在開機時未偵測到 Ethernet 網線時才初始化。

建議的上線工作流程

1. **實體優先**：新裝置直接插入 Ethernet 網線。裝置會偵測連結、使用 DHCP，立即可被發現。
2. **配置**：透過 Ethernet 存取 dashboard 配置站點專屬的靜態 IP 或 Wi-Fi 憑證。
3. **無線轉換**：WiFi 配置完成後，拔除 Ethernet 網線觸發自動 WiFi 切換（就地切換，無需重啟）。
4. **簡化**：不需要獨立的「WiFi」或「Ethernet」韌體；單一「統一」韌體處理兩種硬體配置。

06

7. 網路容錯架構 (v5.5)

Gateway 實作 **雙向就地切換** Ethernet 和 WiFi，零重啟。

7.1 開機最佳化

Path	Time	Key Technique
Ethernet	~8s	5s PHY link check + DHCP
WiFi	~14s	`ENC_TYPE_CCMP` skip-scan + 2-stage retry

- **WiFi** `ENC_TYPE_CCMP`: Passing WPA2 encryption type as 3rd arg to `WiFi.begin()` skips `scanNetworks()` (saves ~10s).
- **2-Stage WiFi Retry**: Attempt 1 (3s warm-up, expected failure) + Attempt 2 (10s real connection).

- **Ethernet 5s PHY Timeout**: Cold-boot PHY (LAN8742A) needs >2s to detect cable. Default 60s is excessive; 5s is sufficient.

7.2 容錯狀態機

7.3 就地 Ethernet 回復探測

每隔 `FO_ETH_PROBE_INTERVAL` (10 秒)，系統**無需重啟**即檢查 Ethernet 網線是否存在：

1. `_onBeforeEthernet()` — Release WiFi server (port 80) + unmount QSPI
2. `WiFi.disconnect()` + `WiFi.end()`
3. `Ethernet.begin(mac, dummy_ip, 2000)` — 2s link check
4. **Cable found** → `Ethernet.begin(mac)` DHCP → switch to Ethernet → `_onEthernetRestored()`
5. **No cable** → `WiFi.begin(ssid, pw, ENC_TYPE_CCMP)` → return to WiFi

> [!IMPORTANT] Each probe causes ~~3s WiFi downtime. The 10s interval balances detection speed~~ (5s average) with ~70% WiFi availability.

7.4 回呼介面

Callback	Trigger	Responsibility
`_onBeforeWiFi`	Before Eth→WiFi switch	Release `EthernetServer` port 80 + unmount QSPI
`_onWiFiRestored`	WiFi connected	Set `wifiServerStarted = false` (loop starts server)
`_onBeforeEthernet`	Before WiFi→Eth probe	Release `WiFiServer` port 80 + unmount QSPI
`_onEthernetRestored`	Ethernet DHCP success	Restart `EthernetServer` on port 80

7.5 硬體限制

- **雙堆疊衝突**：Arduino Opta (STM32H747) 無法同時運作 Ethernet 和 WiFi。在 WiFi 啟用時呼叫 `Ethernet.begin()` 會導致 mbed 懸停。
- **PHY 暫存器存取**：`Ethernet.linkStatus()` 在 `Ethernet.end()` 之後會懸停——必須重新初始化後才能檢查連結。
- **QSPI 化流排共享**：Murata 無線電和 QSPI Flash 共享內部 SPI 化流排。在 WiFi 操作前必須卸載 LittleFS。

7.6 容錯時間常數

Constant	Value	Purpose
`FO_LINK_DEBOUNCE`	3s	Ethernet disconnect debounce
`FO_WIFI_TIMEOUT`	20s	WiFi connection timeout
`FO_ETH_PROBE_INTERVAL`	10s	Ethernet recovery probe interval
`FO_RECOVERY_DELAY`	5s	Ethernet recovery debounce
`FO_RETRY_INTERVAL`	60s	WiFi retry after total failure

IP 分配注意事項

在生產環境（如 `192.168.50.*`）中，靜態 IP 通常在站點專屬設定時逐一分配（`.101`、`.105`、`.9`）。Gateway 的預設靜態 IP 應視為「已知入口」而非永久分配；最終 IP 應透過 Web UI 設定並持久化到 Flash。

07

9. 網路韌性與安全機制 (v5.1)

為防止「網路隔離」（裝置因無效靜態 IP 或網路變更而無法連線），整合了兩個安全機制：

9.1 硬體觸發 DHCP 重設

在開機前 3 秒內按住 **USER 按鈕**（在 Opta 核心中符號定義為 `BTN_USER`）將覆寫所有儲存設定並強制裝置進入 DHCP 模式。

- **硬體特性**：按鈕為低電位有效（內部 pull-up）。偵測需要 `pinMode(BTN_USER, INPUT)`。
- **消除彈跳**：開機時使用 **100ms 延遲**確保按鈕狀態穩定後才確認進入 Safe Mode。
- **持久化**：此變更自動透過 `LocalConfig.saveLocal()` 儲存到 QSPI Flash 上的持久化 `config.json`。

9.2 自動連線備援

若裝置嘗試以靜態 IP 初始化但未能在定義的逾時時間內（WiFi 15 秒）建立連結，它會：

1. 將作用中配置物件的 `network.dhcpEnabled` 切換為 `true`。
2. **持久化到 QSPI Flash**：執行 `LocalConfig.saveLocal()` 確保裝置在後續重啟時保持 DHCP 模式。
3. **重啟堆疊**：以 DHCP 模式再次呼叫 `network.begin()`。

此「黏性備援」確保裝置被移動到新網路且連線失敗後，持續透過 DHCP 可過，直到使用者明確重新配置，避免重複連線失敗的迴圈。

08

10. 現場除錯與故障排除

當 Web Dashboard 在部署後無法連線時：

10.1 Serial 監控

- **工具**：`pio device monitor --baud 115200`
- **啟動日誌**：尋找 `Ethernet... OK` 或 `WiFi... OK` 以及印出的 IP 地址。
- **重設技巧**：若裝置無回應或 Serial 無輸出，切換 DTR/RTS 或使用 1200-baud 重設技巧：`stty -f /dev/cu.usbmodemXXXX 1200; sleep 2`（強制 Opta 進入 bootloader/重設模式）。

10.2 命令列監控（非 TTY 環境）

在無互動終端的環境中（如 CI/CD 日誌、AI Agent、受限 SSH），使用 `stty` 和 `cat` 直接存取：

1. **識別 Port**：`ls /dev/cu.usbmodem*`（macOS）或 `ls /dev/ttyACM*`（Linux）。
2. **初始化與讀取**：

```bash
stty -f /dev/cu.usbmodemXXXXX 115200 raw -echo && cat /dev/cu.usbmodemXXXXX
```

此方法繞過對 TTY 監控器的需求，允許背景式日誌記錄。

主機端準備清單

- **子網對齊**：確保電腦的 IP 與 Opta 在同一範圍。
  - **預設情境**：若 Opta 預設為 `192.168.0.99`，您的 PC 必須在 `192.168.0.X`。
  - **已配置情境**：若設定 Opta 為 `192.168.50.9`，您的 PC 必須在 `192.168.50.X`。
  - **子網不匹配**：使用 `ifconfig`（macOS/Linux）或 `ipconfig`（Windows）驗證自己的 IP。
- **實體連結**：檢查 Opta 的 RJ45 埠 LED 指示燈。無燈通常表示網線故障或缺少電源/交換器連線。
- **Port 衝突**：確保網路上沒有其他裝置使用相同的靜態 IP（`.99`、`.101`、`.9`）。

- **探索（ARP 掃描）**：若因持久化不匹配而不知 IP，使用 ARP 表尋找裝置：

  ```bash
  arp -a | grep -E "192\.168\."
  ```

  尋找以 `aa:bb:cc` 開頭的 MAC 地址（或 Opta 標籤上印的製造商前綴）。

瀏覽器快取與 Socket 狀態

- **IP 變更延遲**：瀏覽器常「掛在」對前一個 IP（如 `.99`）的逾時連線。明確關閉分頁或執行強制重新整理（`Ctrl/Cmd + Shift + R`）。
- **Socket 重設**：某些情況下，IP 變更後需要對 Opta 進行實體電源重置以清除內部網路狀態機。

「LocalConfig 覆寫」陷阱

問題：即使上傳了包含修改後靜態 IP（如從 `192.168.0.99` 改為 `192.168.50.9`）的新韌體，裝置可能仍然以**舊** IP 開機。**原因**：`LocalConfig` 載入優先級（Flash Storage）高於硬編碼韌體預設。若裝置先前已配置並儲存，重啟時會還原舊配置，覆寫新的硬編碼值。**偵測**：檢查 Serial 日誌中的 `Config: Loaded from local storage` 後面的 IP 地址。

「API 覆寫」解法

若裝置在網路上可達但回報不正確的資料或舊 IP，可透過 REST API 強制更新持久化的 `LocalConfig`，無需重新燒錄：

```bash
curl -s -X POST "http://[CURRENT_IP]/api/config" \
  -H "Content-Type: application/json" \
  -d '{"network":{"ip":"192.168.50.9","gateway":"192.168.50.1","subnet":"255.255
```

只要裝置在*某個* IP 上可達，即可繞過實體 Serial 存取就能清除 Flash。

09

11. 部署與韌體上傳錯誤

DFU 故障排除（`Error 74`）

症狀：`dfu-util: No DFU capable USB device available`。**原因與解法**：

1. **實體連線**：確保 USB-C 線完全插入且支援資料傳輸。
2. **偵測檢查（macOS）**：執行 `ls /dev/cu.usbmodem*`。若出現如 `/dev/cu.usbmodem12201` 的 port，裝置已連線但處於 **Serial 模式**而非 DFU 模式。
3. **觸發 DFU 模式**：
   - **快速雙擊 Reset**：快速按 **Reset** 按鈕兩次。
   - **長按**：按住 **Reset** 按鈕約 2 秒。
   - **視覺確認**：Opta 的 LED（通常 L1/L2）會開始有節奏的「心跳」或閃爍模式。
4. **驅動/Port 衝突**：確保沒有 Serial Monitor 正在使用。若在心跳模式中仍然錯誤，嘗試不同的 USB 埠或線。

10

12. WiFi 無線電韌體救援 (v5.4.13)

在工業環境中，Arduino Opta 的 WiFi 無線電模組（Nina-W10）可能因 QSPI 分區表損壞或應用程式資料（LittleFS）與無線電韌體區域重疊而進入「Mount Failed」狀態。

12.1 「Failed to mount filesystem」錯誤

- **症狀**：Serial monitor 在開機時重複印出 `Failed to mount the filesystem containing the WiFi firmware`。
- **原因**：QSPI Flash 開頭的 1MB 專用分區已被覆寫或 Master Boot Record（MBR）偏移。

12.2 強制救援順序

恢復無線電需要使用 Arduino IDE（v2.2+）的特定順序。**勿跳過第 2 步。**

1. **第一階段：QSPI 分區重設**
   - **工具：**`Examples > STM32H747_System > QSPIFormat`。
   - **動作：**上傳到 **M7 Core**。開啟 Serial Monitor（115200 baud）。
   - **結果：**清除整個 16MB Flash 並恢復出廠 MBR。**注意：此操作會刪除所有使用者設定。**
2. **第二階段：無線電韌體重新佈建（重大）**
   - **工具：**`Examples > WiFi > WiFiFirmwareUpdater`。
   - **動作：**QSPIFormat 完成後立即上傳。依照 Serial Monitor 提示操作。
   - **結果：**將 Nina-W10 無線電檔案系統和 TLS 憑證重新安裝到 1MB 分區。**跳過此步驟會導致持續的 mount 失敗。**
3. **第三階段：專案恢復**
   - **動作：**返回 PlatformIO 重新上傳 **MES Gateway** 專案韌體。
   - **結果：**裝置應能正確開機、找到無線電韌體，並允許啟用 WiFi AP 模式。

12.3 資料復原說明

由於 `QSPIFormat` 會清除配置，若裝置半可運作，請在執行前記下以下設定：

- **裝置名稱**
- **靜態 IP 憑證**
- **MQTT Broker 地址**
- **自訂信號規則**

第三階段完成後，裝置將預設為 DHCP。存取 UI 即可恢復個人化設定。

11

13. 非同步 RS485 掃描 (v5.6)

由於 Modbus 掃描的長時間（最長 2.5 分鐘），同步架構會觸發 HTTP 逾時並阻塞 Gateway 的 Web Server。v5.6 引入了非同步模式。

13.1 非同步模式實作

- **Stage 1 (POST)**: Triggering a scan via `/api/modbus/scan` returns `{"success":true,"scanning":true}` immediately.
- **Stage 2 (Loop)**: The main `loop()` detects the `g_scanPending` flag.
  - **Cache Reset (v5.6.1)**: To ensure results reflect the current bus state, the logic explicitly calls `powerMonitor.clearScanResults()` and `localConfig.deleteScanResults()` before starting.
  - **Discovery**: Performs a round-robin poll of all slave IDs.
- **Stage 3 (GET)**: The frontend polls the endpoint.
  - **Configuration Awareness (v5.6.1)**: The backend iterates through the discovered devices and compares their `slaveId` against the active `DeviceConfig`. It injects an `alreadyAdded` boolean field into the JSON response for each device, enabling the UI to distinguish between new and existing equipment.

13.2 化流排韌性與鎖定

- **手動化流排鎖定**：掃描期間使用 `lockBus(true)` 暫停正常電力監控輪詢。
- **ZOMBIE 鎖定韌性**：為防止瀏覽器重新整理期間化流排被永久鎖定，`WebPage::init()` 例程和後端啟動序列會發出明確的 `lockBus(false)`。
- **逾時最佳化**：`MODBUS_TIMEOUT` 從 500ms 降低到 200ms 以加速掃描，同時維持工業相容性。

13.3 裝置專屬整合（Finder 6M）

- **同位對齊**：Finder 6M 的出廠預設為 **8N1**（SERIAL_8N1）。透過修正 `PowerMonitor` 管理器中的同位設定，穩定了高效能輪詢。
- **半雙工硬體**：Arduino Opta 內建的 RS485 收發器需要正確設定 `halfDuplex` 旗標（`_uart->begin()` 的第 3 個參數，目前為 `true`）以允許共用 TX/RX 接腳正確切換角色。

13.4 Power Monitor 讀取最佳化（Finder 6M 批次限制）

v5.8.5 中發現 RS485 裝置（特別是 Finder 6M）在單次 Modbus 批次讀取超過 18 個暫存器時會回報「Offline」。

- **根本原因**：Finder 6M 有硬體/韌體限制，單次批次讀取超過 18 個暫存器（如 192-213，共 22 個）會觸發 Modbus 逾時或錯誤。
- **解法（兩階段讀取）**：`PowerMonitor` 邏輯重構為分兩次獨立的 Modbus 呼叫：
  1. **批次 1**：Registers 192-209（18 個）涵蓋電壓、電流、功率和頻率。
  2. **批次 2**：Registers 210-213（4 個）涵蓋正向和負向能量計數器。
- **顯示與擷取解耦**：系統使用 `displayMask` 控制 UI 中顯示的量測項目。此機制與實際資料擷取解耦，確保 Data Logger 無論使用者顯示偏好如何，都持續接收所有欄位的更新。

13.5 非同步出站連線模式 (v5.9+)

§13.1 的非同步模式解決了「長時間操作阻塞 HTTP handler」的問題。v5.9 的授權啟用發現了更深層的限制：**入站 HTTP 連線期間無法建立出站 TCP 連線**。

根本原因

STM32H747 的 Mbed OS 網路堆疊（lwIP）資源有限。當設備正在處理入站 HTTP 請求時（EthernetServer socket 已佔用），嘗試建立出站 TCP 連線（`EthernetClient.connect()`）會因為 socket 資源爭用而失敗（回傳 `0`）。

通用非同步出站連線模式

此模式適用於所有需要在 HTTP handler 中觸發出站 TCP 連線的場景：

```
HTTP Handler（入站 socket 佔用中）          loop()（入站 socket 已釋放）
──────────────────────────────────          ──────────────────────────────
1. 收到 POST 請求                           4. 偵測 pending flag
2. 設定 g_pending* flag + 參數              5. 執行出站 TCP 連線
3. 回傳 202 Accepted                        6. 寫入結果到全域變數
   {"status":"activating"}                  7. 清除 pending flag

前端 polling（每 2 秒）
──────────────────────
8. GET 查詢結果 → 回傳最終狀態
```

實際應用：授權啟用

階段	實作位置	說明
排程	`handlePostLicenseActivate()` in `ApiHandlers_Connectivity.cpp`	設定 `g_pendingLicense.pending = true` + 回傳 `202`
執行	`doLicenseActivateAsync()` in `loop()`	建立出站 TCP 到 Config Server
輪詢	`activateLicense()` in `web/index.html`	前端每 2 秒 GET `/api/license/status`，最多 5 次

注意事項

- **不可**在 HTTP handler 內直接呼叫 `EthernetClient.connect()`
- 前端必須處理 `202` 狀態碼並啟動 polling
- 全域 pending 結構應包含超時保護（避免 flag 永久卡住）

12

14. OTA Bundle 機制（規劃中）

為確保韌體和 Web UI（LittleFS 資產）在遠端更新期間保持同步，實作了「Bundle」方法。

14.1 Bundle 架構

Gateway 使用單一 `.mesb` 容器而非獨立更新：

- **Header (32B)**：包含專案魔術字（`MESB`）、韌體大小、Web payload 大小及兩者的 CRC32。
- **韌體二進位檔**：標準二進位檔，用於 Internal Flash。
- **Web Payload**：壓縮的 `index.html`，用於 QSPI Partition 4 LittleFS。

14.2 更新生命週期

1. **下載**：Bundle 下載到 **QSPI Partition 2**（OTA Buffer，5MB）。
2. **驗證**：系統驗證 header 和完整性（CRC32）。
3. **韌體更新**：韌體透過 `mbed::BlockDevice` 寫入 Internal Flash，系統重啟。
4. **UI 部署**：首次開機時，新韌體偵測 Partition 2 中的 Web payload，解壓縮並覆寫 Partition 4 中的 `/web/index.html`。
5. **完成**：OTA buffer 被清除，系統恢復正常運作。

14.3 版本鎖定

此機制防止「版本漂移」（新韌體期望特定的 API 結構但舊的快取 Web UI 不支援），確保工業介面可靠性。

13

15. TCP Converter 與雙向資料 (v5.9)

TCP Converter 將 MQTT/Modbus 通道轉換為進階資料管線，整合到規則引擎中。

15.1 核心邏輯：樣板替換（出站）

系統透過對使用者定義的 JSON 樣板執行純關鍵字替換來滿足複雜的 MES schema。

- **變數標籤**：支援 `${io.*}`、`${vc.*}`、`${rs485.*}` 和 `${sys.*}`。
- **工作流**：使用者將原始 JSON 範例貼入 UI 的 `<textarea>`，將靜態值替換為標籤，儲存到 `config.json`。
- **執行**：邏輯迴圈中的非阻塞非同步組裝。

15.2 入站 JSON 解析器（信號來源）

MQTT 訊息被解析以擷取系統信號：

Mode	Strategy	Use Case
Boolean	String match	cmd == "START"
Numeric	Direct extraction	speed == 1500
String	Raw text	job_id "A123"

- **值策略**:

  1. **最後值保持**:保留狀態直到下一則訊息到達。
  2. **過期逾時**:可選視窗(N 秒);若無更新則將信號標記為 'stale'。

15.3 資源限制

- **最大樣板數:16 個作用中樣板**(由 `#define MAX_CONVERTER_TEMPLATES` 控制)。
- **RAM 安全**:每個作用中樣板分配約 1KB 以防止 heap 破碎化。
- **協議範圍**:主要針對 **MQTT**(非同步)和 **Modbus TCP**(暫存器對暫存器橋接)。原始 TCP Sockets 已廢棄。

14

17. Modbus TCP Interface (Server/Client)

為提升工廠內 HMI、PLC 與 SCADA 的互操作性,Gateway 提供標準的 Modbus TCP 支援。

17.1 Server Role (Slave) - Port 502

Gateway 作為 Server 運行,可支援最多 4 個同時連線。目前採用 **0-based 固定位址對映 (Fixed Address Map)**。經過 v5.9 擴表後,現在擁有充足的暫存器空間能容納主機、擴充模組、虛擬通道、以及十數台的 Modbus 設備:

暫存器映射 (Register Map - 實作現況 v5.9 大容量版):

暫存器類型 (HMI 類型)	地址範圍	對應數據來源	權限	說明
Coil (0x)	`0 ~ 3`	主機 DO[0..3]	R/W	讀寫主機實體繼電器輸出
Coil (0x)	`16 + e*16`	擴充模組 DO	R/W	讀寫擴充模組輸出 (Exp0: 16-23, Exp1: 32-39...)
Discrete Input (1x)	`0 ~ 7`	主機 DI[0..7]	R	讀取主機實體輸入 High/Low
Discrete Input (1x)	`16 + e*16`	擴充模組 DI	R	讀取擴充模組輸入 (Exp0: 16-31, Exp1: 32-47...)
Input Register (3x)	`0 ~ 7`	主機 AI[0..7]	R	讀取類比輸入原始值 (經倍率轉換 0~65520)
Holding Register (4x)	`0 ~ 1023`	內部系統 Holding 區	R/W	提供高達 1024 個通用的 Holding 區 (`MODBUS_HOLDING_REG_COUNT=1024`)，可供外界讀寫。可作為 Rule Engine 與 TCP Converter 輸出目標的共通變數區。

推薦 Holding Register (4x) 應用區間規劃： 為避免設定混亂，建議 SCADA/HMI 規劃位址時遵循以下使用分佈：

- `0 ~ 99`: 系統內部保留與通用控制 VAR 變數。
- `100 ~ 199`: 虛擬通道 (Virtual Channels: Counter / Timer) 推送值。
- `200 ~ 499`: 實體 RS485 (Modbus RTU) 設備快取區 (支援 10+ 台設備，每台保留 20-30 Registers)。
- `500 ~ 999`: 其他 TCP IO 通道 (Parser) 所擷取提取的機台實時生產數據暫地。

(註：目前 Holding Register 僅作為大區塊供外界存取與系統對接，開發者可將 Parser 擷取到的值以 rule 行為自由倒入上述任一位址中。)

17.2 Client Role (Master)

Gateway 作為 Master 可以主動向外讀取數據（**目前 TCP 部分尚未實作**）。

- 若須收集其他 Modbus TCP 設備數據至系統，請透過第三方軟體(如 Node-RED)橋接為 MQTT Parser 輸入。
- 實體序列線路的 **Modbus RTU Master** 則已完整實作於自定義的 `ModbusManager` 中，支援 RS485 輪詢。

04 開發者 / 整合

開發環境建置

📅 最後更新：2026-06-16 | 📌 負責人：KC

本指南幫助開發者建立完整的 MES Gateway 開發環境，從安裝工具到編譯上傳韌體。

01

前置需求

工具	版本	用途
PlatformIO	Core 6.x	韌體編譯與上傳
VS Code	最新版	編輯器（搭配 PlatformIO IDE 外掛）
Node.js	18+	Config Server
Python	3.8+	測試腳本與 Web 上傳工具
Git	最新版	版本控制

02

Step 1：Clone 專案

```bash
git clone <repo-url> PillarArduno
cd PillarArduno
```

03

Step 2：安裝依賴

韌體端

PlatformIO 會在首次編譯時自動安裝 Arduino Core 和 library 依賴（定義在 `platformio.ini`）。

Config Server

```bash
cd config-server
npm install
cd ..
```

Python 工具

```bash
pip install requests pyserial
```

04

Step 3：編譯韌體

```bash
# 編譯（不上傳）— 確認資源使用
pio run

# 確認輸出中的 RAM/Flash 比例在安全範圍：
# RAM:   < 25% 🟢
# Flash: < 85% 🟢
```

05

Step 4：上傳韌體

```bash
# 用 USB 線連接 Opta，確認 port
ls /dev/cu.usbmodem*

# 編譯 + 上傳
pio run -t upload

# 若自動偵測失敗，指定 port
pio run -t upload --upload-port /dev/cu.usbmodemXXXXX
```

> ⚠️ 上傳失敗？嘗試**快速雙擊 Reset 按鈕**進入 DFU 模式。

06

Step 5：上傳 Web UI

> ⚠️ **`upload_web.py` 已廢除**。Web 資產不再單獨上傳：`index.html` ＋核心 `app.js` 編進韌體 PROGMEM（`build_webpage.py` pre-hook），CSS／語系／`workflow.html` 走 QSPI，全部透過 `.mesb` 一檔帶齊。開發機補資產有兩條路：

```bash
# (A) 推薦：灌同版 .mesb (fw+6 資產一次到位，會重燒重開，約 90 秒)
curl -sS -X POST "http://<DEVICE_IP>/api/ota/bundle" \
  -H "Authorization: Bearer smmsadmin" \
  -H "Content-Type: application/octet-stream" \
  --data-binary @release/mes-gateway-v<版本>.mesb

# (B) 純補資產不重燒：逐一 POST web/build/*.gz (先跑 build_webpage.py)
for f in app.js.gz workflow.html.gz styles.css.gz \
         i18n-en.json.gz i18n-zh-TW.json.gz i18n-zh-CN.json.gz; do
  curl -sS -X POST "http://<DEVICE_IP>/api/web?name=$f" \
    -H "Authorization: Bearer smmsadmin" --data-binary @web/build/$f
done
```

> 詳見 `guide-cli-opta.md` §5。

07

Step 6：查看 Serial Log

```bash
# 方式 A：PlatformIO（需要 TTY 環境）
pio device monitor -b 115200

# 方式 B：直接讀取（適用非 TTY 環境，如 Agent 終端機）
ls /dev/cu.usbmodem*
stty -f /dev/cu.usbmodem12201 115200 raw -echo && cat /dev/cu.usbmodem12201

# 方式 C：網頁遠端看（v6.0.1+；設備只能經 VPN、無法插 USB 時）— 需 admin token
curl -s http://<設備IP>/api/log  -H "Authorization: Bearer <admin-token>"   # 純文
curl -s http://<設備IP>/api/diag -H "Authorization: Bearer <admin-token>"   # 開機
```

> 方式 C 也有網頁版：瀏覽器開 `http://<設備IP>/log.html`（或設定頁「📟 設備日誌」鈕），每 5 秒自動刷新 + 頂部橫幅顯示重開原因。log ring 即時吐 RS485 輪詢/MQTT/開機序列；敏感值（密碼/token）已遮罩。**限制**：設備網路死掉（reboot loop）時進不去，改看雲端心跳的 `bootReason` 或現場 USB。

08

Step 7：啟動 Config Server

```bash
cd config-server
npm start
# 預設在 http://localhost:8888
```

09

專案結構速覽

```
PillarArduno/
├── src/                ← 韌體原始碼（32 個 .h/.cpp 檔案）
├── web/                ← Web UI（index.html）
├── config-server/      ← 集中管理伺服器（Node.js）
├── scripts/            ← 工具腳本（上傳 Web、打包 OTA）
├── test/               ← 測試腳本
├── docs/               ← 技術文件（你現在讀的就是）
├── data/               ← 範例資料和配置
└── platformio.ini      ← PlatformIO 專案設定
```

10

常見問題

問題	解法
`pio run` 報錯找不到 board	確認 `platformio.ini` 中 `board = opta`
Serial 無輸出	確認鮑率 115200，重新 Reset 設備
Web UI 空白	QSPI 缺資產 → 灌 `.mesb` 或 POST `web/build/*.gz`（見 Step 5；`upload_web.py` 已廢除）
Config Server 起不來	確認已執行 `npm install`

11

下一步

- 首次部署指南 — 從開箱到上線
- API 參考 — REST API 端點

04 開發者 / 整合

CLI 指令參考

本指南說明如何透過終端機 (Terminal/命令提示字元) 對 MES Gateway 設備執行 OTA (Over-The-Air) 韌體更新。這在無法使用 Web UI，或是在自動化流水線中進行批次部署時非常有用。

情境說明：

- 如果您是**終端客戶或運保人員**，且手上只有一個 `.bin` 韌體檔案，請直接參考 **策略一 (cURL)**。這不需要安裝任何額外環境。
- 如果您是**內部開發人員**，且擁有完整原始碼專案，請參考 **策略二 (PlatformIO)**。

01

策略一：直接推送韌體二進位檔 (客戶最推薦)

如果您手上只有編譯好的 `.bin` 檔案 (例如 `mes-gateway-x.x.x.bin`)，最簡單的方式是使用 `curl` 指令直接推送給設備。現代操作系統 (Windows 10+, macOS, Linux) 都已經內建了 `curl`，**您完全不需要安裝 PlatformIO 或撰寫腳本**。

1. 準備前置資訊

- 開啟終端機 (Windows 請開 CMD 或 PowerShell，macOS 請開 Terminal)。
- 確認您的設備目前的 IP 位址 (例如 `192.168.51.32`)。
- 確認設備的 API 授權 Token (預設為 `admin-token`)。

2. 執行更新

執行以下 `POST` 請求到 `/api/ota/upload`：

```bash
# 請將 IP、Token 與 "@檔案路徑" 替換成您的實際參數
curl -X POST "http://192.168.51.32/api/ota/upload" \
  -H "Authorization: Bearer admin-token" \
  -H "Content-Type: application/octet-stream" \
  -H "X-Firmware-Version: 5.9.9" \
  --data-binary "@firmware.bin" \
  --connect-timeout 5 \
  --max-time 120
```

- **`--data-binary "@firmware.bin"`**：請修改 `@` 後方的路徑指向您的 `.bin` 檔案，若檔案與終端機不在同一個資料夾，請使用絕對路徑 (例如 `@C:/Users/abc/Desktop/firmware.bin`)。
- **`--max-time 120`**：上傳與寫入 Flash 時間較長，建議給予至少 60-120 秒以免連線提早中斷。

3. 預期行為

- 寫入成功後，設備會回傳 HTTP 200 OK，並在 3 秒內自動閃爍燈號並重啟。
- 若您的指令因為 Timeout 斷線 (`curl: (28) Operation timed out`)，但設備隨後有成功重啟亮燈，請不用擔心，這通常代表設備忙於重啟而未關閉連線，實際上已經寫入成功。

02

策略二：使用 Python 腳本推送 (附帶進度與防呆)

如果您覺得 `curl` 指令太長、容易打錯，開發團隊也可以將專案內的 `scripts/pio_ota.py` 與 `.bin` 一起打包交給客戶 (客戶電腦需安裝 Python 3)。

這支腳本內建了完整的容錯機制與中文進度提示。

執行更新

在終端機中，切換到檔案所在目錄，並執行以下指令：

```bash
# 用法：python3 pio_ota.py <韌體檔案> <設備IP>
python3 pio_ota.py mes-gateway-v5.9.9.bin 192.168.51.32

# 如果設備自訂了 admin token（與預設 'admin-token' 不同），用 OTA_TOKEN env：
OTA_TOKEN=smmsadmin python3 pio_ota.py mes-gateway-v5.9.120.bin 192.168.72.77
```

腳本預設讀 `OTA_TOKEN` env，沒設就 fallback 到 `admin-token`。中途若斷線也會顯示友善的提示，減少客戶的疑慮。

03

策略三：使用 PlatformIO 開發環境直接上傳 (開發者專用)

如果您是內部團隊且擁有完整的 PlatformIO (PIO) 原始碼專案，您可以直接透過指令觸發本機端編譯與上傳，專案已為此內建了處理邏輯。此策略適合開發階段使用。

前置作業：安裝 PlatformIO CLI

如果您尚未安裝 PlatformIO 的命令列工具，可以透過以下任一方式安裝：

- **使用 Homebrew (macOS/Linux)：**

```bash
brew install platformio
```

- **使用 Python `pip` (全平台)：**

```bash
pip install -U platformio
```

安裝完成後，可執行 `pio --version` 確認。

執行 PIO 更新

在專案根目錄執行以下指令：

```bash
pio run -e opta_ota -t upload
```

- 這個指令會自動觸發編譯（若程式碼有更動），並使用自訂腳本 `scripts/pio_ota.py` 將剛編譯出的 `.bin` 發送到預設的設備 IP（`192.168.51.32`）。
- 如果您需要上傳至不同 IP 的設備，可以覆寫環境變數：

```bash
pio run -e opta_ota -t upload --upload-port 192.168.1.100
```

- 腳本內建了容錯處理。若上傳成功但設備為加速重啟而主動斷線，PlatformIO 依然會正常提示 **SUCCESS**。

背後運作機制 (若需要客製化)

這套 PIO 的對接機制依賴了專案中的兩項設定：

1. **`platformio.ini` 內的環境變數設定**：利用了 `upload_protocol = custom` 攔截原生的 USB 燒錄行為。
2. **自訂上傳腳本 `scripts/pio_ota.py`**：負責讀取 `.bin`，並打 API 給 `/api/ota/upload`。Token 讀 env `OTA_TOKEN`，沒設就 fallback 到 `admin-token`（向下相容）。

04

策略四：觸發設備向 Config Server 拉取 (Pull from Config Server)

若設備已經正確設定 `Config Server`（例如官方伺服器 `192.168.51.32:3001`），您可以要求設備自行向 Config Server 檢查版本並下載更新，節省本機頻寬。

1. 觸發版本檢查清單

通知設備開始向 Config Server 查詢可用的韌體版本：

```bash
curl -X POST "http://$DEVICE_IP/api/ota/versions" \
  -H "Authorization: Bearer $TOKEN"
```

(此為非同步操作，會回傳 `{"status":"fetching"}`)

2. 獲取版本資訊與下載連結

等待 2~3 秒後，使用 `GET` 查詢結果：

```bash
curl -s "http://$DEVICE_IP/api/ota/versions" \
  -H "Authorization: Bearer $TOKEN" | grep -A 5 -B 5 "versions"
```

此指令會列出支援的版本與對應的 `downloadUrl`（或內部邏輯保留的最新版本資訊）。

3. 指示設備啟動下載並更新

您可以指定特定的下載 URL 與版本號，迫使設備啟動下載任務：

```bash
curl -X POST "http://$DEVICE_IP/api/ota/trigger" \
  -H "Authorization: Bearer $TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "url": "http://192.168.51.32:3001/api/firmware/download/mes-gateway-v5.9.9-2(
    "version": "5.9.9"
  }'
```

(若未在 Body 中指定 URL，設備將預設抓取它剛剛查詢到的「最新可用版本」)

設備收到此指令後，會回傳 HTTP 200 `{"ok":true,"message":"OTA download started"}`。
隨後設備會在背景連線至指定的 URL 下載韌體，下載並寫入完成後會自動重啟。

05

特例情境：全新出廠設備實體安裝 (USB Bootstrap)

對於**剛出廠的全新 Opta 設備**，或是無法連網而需要「救磚」的設備，因為裡面尚未包含本專案的 Web Server 功能，故**在此之前提到的網路 OTA (策略一～四) 皆不可用**。

針對這種只拿到 `.bin` 檔案且第一次必須插 USB 線的「開荒」情境，**強烈建議直接使用 Arduino 官方命令列工具 `arduino-cli`**，這比設定 PlatformIO 簡單且穩定許多：

1. 安裝 `arduino-cli`

- **macOS / Linux**: 可使用 Homebrew 快速安裝：

```bash
brew install arduino-cli
```

- **Windows**: 請至 Arduino CLI GitHub 首頁 下載執行檔並加入環境變數。

2. 安裝 Opta 核心通訊支援

初次使用 `arduino-cli`，需要讓它下載並認識 Opta 設備的底層驅動：

```bash
arduino-cli core update-index
arduino-cli core install arduino:mbed_opta
```

3. 接線並尋找連線 Port

將 Opta 設備透過一條能傳輸資料的 USB-C 線接上電腦，接著執行：

```bash
arduino-cli board list
```

在輸出的清單中，找出標示為 Opta 或是 mbed 設備的 Port 號。*(例如 Windows 為 `COM3` 等，macOS/Linux 為 `/dev/cu.usbmodem12201` 這類字眼)*

4. 一行指令燒錄 `.bin`

準備好後，在 `.bin` 檔案所在的資料夾下執行這行指令（請把 `<PORT>` 改成剛才查到的埠號）：

```bash
# 此處的檔案名稱以 mes-gateway.bin 為例
arduino-cli upload -b arduino:mbed_opta:opta -p <PORT> -i mes-gateway.bin
```

這個指令背後會自動幫你：

1. 將原本設備的 USB Port 以 1200 bps 進行觸碰，強迫設備進入 DFU Bootloader 燒錄模式（此時設備 LED 亮起漸暗漸亮）。
2. 背景呼叫 `dfu-util` 將 `.bin` 準確寫進 Flash `0x08040000` 的正確位置。

畫面上看到 `Download done.` 以及 `File downloaded successfully` 字樣即大功告成，設備會自動重啟並套用全新韌體！接著您就可以改用前面的「OTA 策略」，未來都不再需要插 USB 了。

06

驗證更新結果

等待設備重啟（約需 10~15 秒）後，您可以透過 `/api/status` 或 `/api/config` 查詢當前的韌體版本，確認更新是否成功。

```bash
# 查詢設備狀態與版本
curl -s "http://$DEVICE_IP/api/status" \
  -H "Authorization: Bearer $TOKEN"

# 預期會看到類似以下的 JSON 片段：
# {
#   "firmwareVersion": "5.9.9",
#   "uptime": 25,
#   ...
# }
```

04 開發者 / 整合

OTA 與內嵌 Web

📅 最後更新：2026-06-16 | 📌 負責人：KC

本文件說明 Arduino Opta MES Gateway 的韌體如何整合 Web UI，以及相關的編譯與 OTA 更新操作流程。

⚠️ **v5.9.155+ 重大更新** 本文件早期版本描述的「全面廢除 `.mesb`、所有前端常駐 PROGMEM」**已部分反轉**。Flash 逼近 100% 上限後，**CSS 與三套語系字典（zh-TW/en/zh-CN）已從 PROGMEM 搬到 QSPI** runtime 服務，且 **`.mesb` 單一檔 OTA 已復活**（一檔帶 firmware + web 資產）。以下內容已更新。

01

1. 架構變更摘要

混合架構（v5.9.155+）：

資源	位置	服務方式
`index.html` + `app.js`（核心）	韌體 PROGMEM（`src/WebPage.h`）	`QSPIWebManager` HTTP GZIP 串流
`styles.css`	**QSPI P4 `/cfg/web/`**	`serveFile()`（ETag 304 快取）
`i18n-{zh-TW,en,zh-CN}.json`	**QSPI P4 `/cfg/web/`**	`serveFile()`（ETag 304 快取）
極小英文 fallback 字典	韌體 PROGMEM	QSPI 缺檔時兜底，UI 不全白

- 核心 HTML+JS 仍在 `pio run` 的 pre-build hook（`scripts/build_webpage.py`）被 Gzip + 轉 C byte array 寫入 `src/WebPage.h`；同一腳本**另外**輸出 `web/build/styles.css.gz`、`web/build/i18n-*.json.gz` 供打包進 `.mesb` 投遞到 QSPI。
- **為何要搬**：整包內嵌時 binary 788KB > 780KB OTA brick 上限 → 只能 USB DFU。把 CSS+語系 搬 QSPI 後 binary 跌破 780KB，**OTA 復活**。QSPI/WiFi 並存的 doctrine 也一併被實證推翻（ADR-014）。

02

2. Web UI 開發與建置流程

前端開發者工作流

前端開發者（Vue / Vite 專案）維持原本的開發習慣，唯一的要求是：**將最終編譯出且 Inline 所有 CSS/JS 的單一 `index.html` 檔案，放置或替換掉本專案目錄下的 `web/index.html`**。

韌體編譯者工作流

無需執行任何額外命令，建置與封裝已全自動化！

```bash
# 編譯韌體（不上傳）
pio run -e opta

# 編譯 + 上傳到板子
pio run -e opta -t upload

# 指定 USB port（若自動偵測失敗）
pio run -e opta -t upload --upload-port /dev/cu.usbmodemXXXXX
```

[!IMPORTANT] **務必加上 `-e opta`**。專案的 `platformio.ini` 同時定義了 `opta`（Arduino MCU）與 `native`（本機 Unit Test）兩個環境。若直接執行 `pio run` 不帶 `-e` 參數，PlatformIO 會同時編譯兩個環境，而 `native` 環境因為缺少 `Arduino.h` 標頭檔必然報錯。這個錯誤**不影響韌體編譯結果**，但會讓終端機輸出看起來有紅字 FAILED。加上 `-e opta` 即可避免。

自動化原理：`platformio.ini` 中已加掛 `extra_scripts = pre:scripts/build_webpage.py`。因此在每一次編譯前，腳本會自動檢查 `web/index.html`，並將其壓縮寫入 `src/WebPage.h`，確保韌體時刻包含最新的前端介面。

上傳後驗證

每次上傳後，建議透過 Serial Monitor 確認韌體正常啟動：

```bash
# 先確認 port 名稱
ls /dev/cu.usbmodem*

# 連線監聽開機日誌
stty -f /dev/cu.usbmodem12201 115200 raw -echo && cat /dev/cu.usbmodem12201
```

03

3. OTA 遠端更新操作

📌 **發佈程序的權威 SOP 即本文（見下方各節）**。發佈分兩條程序：「程序 A 一般版本（`.mesb`）」與「程序 B 拓荒包（無版號 `bootstrap .bin`）」，兩者清楚分開。本節只講建置與封裝的技術細節。

因 CSS/語系已搬 QSPI，OTA 需同時帶 firmware + 這些資產，故**使用 `.mesb` 單一檔**（`.bin` 只更新 firmware，不帶資產，僅適用資產未變動或乙太網 raw OTA）。

取得更新檔

`pio run -e opta` 編譯後 firmware 位於 `.pio/build/opta/firmware.bin`；web 資產位於 `web/build/*.gz`（同一次 build 由 `build_webpage.py` 產出）。打包成 `.mesb`：

```bash
python3 scripts/build_ota_bundle.py \
  --fw .pio/build/opta/firmware.bin \
  --asset web/build/styles.css.gz \
  --asset web/build/i18n-en.json.gz \
  --asset web/build/i18n-zh-TW.json.gz \
  --asset web/build/i18n-zh-CN.json.gz \
  --out release/firmware.mesb
python3 scripts/build_ota_bundle.py --verify release/firmware.mesb   # 驗 magic +
```

> **`.mesb` 內含什麼**：韌體 + 6 個網頁資產（`app.js` / `workflow.html` / `styles.css` / `i18n-{en,zh-TW,zh-CN}.json`）+ `favicon.ico`，打包成單一檔。上傳後 firmware → QSPI P2、資產 → P4 `/cfg/web/`，重開即韌體與前端同版本一次到位。

拓荒包（Pioneer）一鍵完整安裝閉環（WI-149）

新設備 / 救磚的端到端開荒流程（已實機驗證）：

1. **USB 首燒**拓荒包 `mes-gateway-bootstrap.bin`（**無版號、固定一顆**；精簡 `.bin`，自包單一開荒頁，不依賴 QSPI 資產。雲端下載連結永不變，詳見本文程序 B）。
2. 開機 → 設備網頁 = **拓荒設定頁**（UID + 網路設定 + 大大的「上傳 .mesb」鈕）。8310 自動過濾 WiFi 欄（型號感知 WI-151）。
3. 在拓荒頁（或完整版）**上傳官方 `.mesb`** → `POST /api/ota/bundle` → `handleMesbUpload` → 設備寫 QSPI + 重開 → **完整版（韌體 + 完整 UI）**。

favicon：拓荒頁內嵌真 `favicon.ico`（base64）；`.mesb` 也打包 `favicon.ico` 給完整版（韌體 `GET /favicon.ico` 由 QSPI `/cfg/web/favicon.ico` 服務）。

ETag 修正（WI-149）：根網頁 ETag 已併入頁面大小（不只 `FW_VERSION`），避免 pioneer → 完整版（同版本號）OTA 後瀏覽器仍顯示舊快取的拓荒頁。

發佈更新

透過 Config Server 上傳 `release/firmware.mesb`（雲端依 `MESB` magic 自動命名 `.mesb`）→ 選設備「部署」。設備依下載 URL 副檔名分流：`.mesb` → `handleMesbUpload`（firmware → QSPI P2 `UPDATE.BIN`、資產 → P4 `/cfg/web/`，含強制 reformat P2 + mbed File API）；`.bin` → `handleOtaUpload`（raw 寫 P2）。詳見使用者手冊 §13。

韌體與前端同版本保證：`.mesb` 一檔內含兩者，OTA 後 QSPI 資產與 firmware 必然同版本，杜絕版本不一致導致 API 溝通失敗。**OTA 寫 QSPI 仍有 soft-brick 風險（可 USB DFU 救回）。**

04

4. 資源佔用與安全水位監控

每次 `pio run -e opta` 完畢時，請務必關注終端機最後輸出的資源使用率：

```
RAM:   [====      ]  42.7% (used 223816 bytes from 523624 bytes)
Flash: [==========]  98.2% (used 772584 bytes from 786432 bytes)   ← v5.9.172 實
```

⚠️ **本專案 Flash 長期貼近上限**：功能持續累積，Flash 已達 ~98%。**真正的硬限制是 `firmware.bin` 必須 < 780000 bytes（OTA partition cap 786432 − overhead），否則 OTA 會被 413 拒絕、只能 USB DFU。** WI-124 把 CSS+語系搬 QSPI 正是為了把 binary 從 788KB 壓回 ~777KB（< 780KB），讓 OTA 復活。**新增功能前務必評估 size，守住 780000 上限。**

資源	🟢 安全	🟡 警戒	🔴 危險
RAM	< 25%	25-40%	> 40%
Flash (binary)	< 740KB	740–780KB	**> 780KB (OTA brick 線)**

若逼近 780KB，優先考慮把更多靜態資源（圖檔、字典）搬 QSPI，而非加大 PROGMEM。

04 開發者 / 整合

Config Server 部署

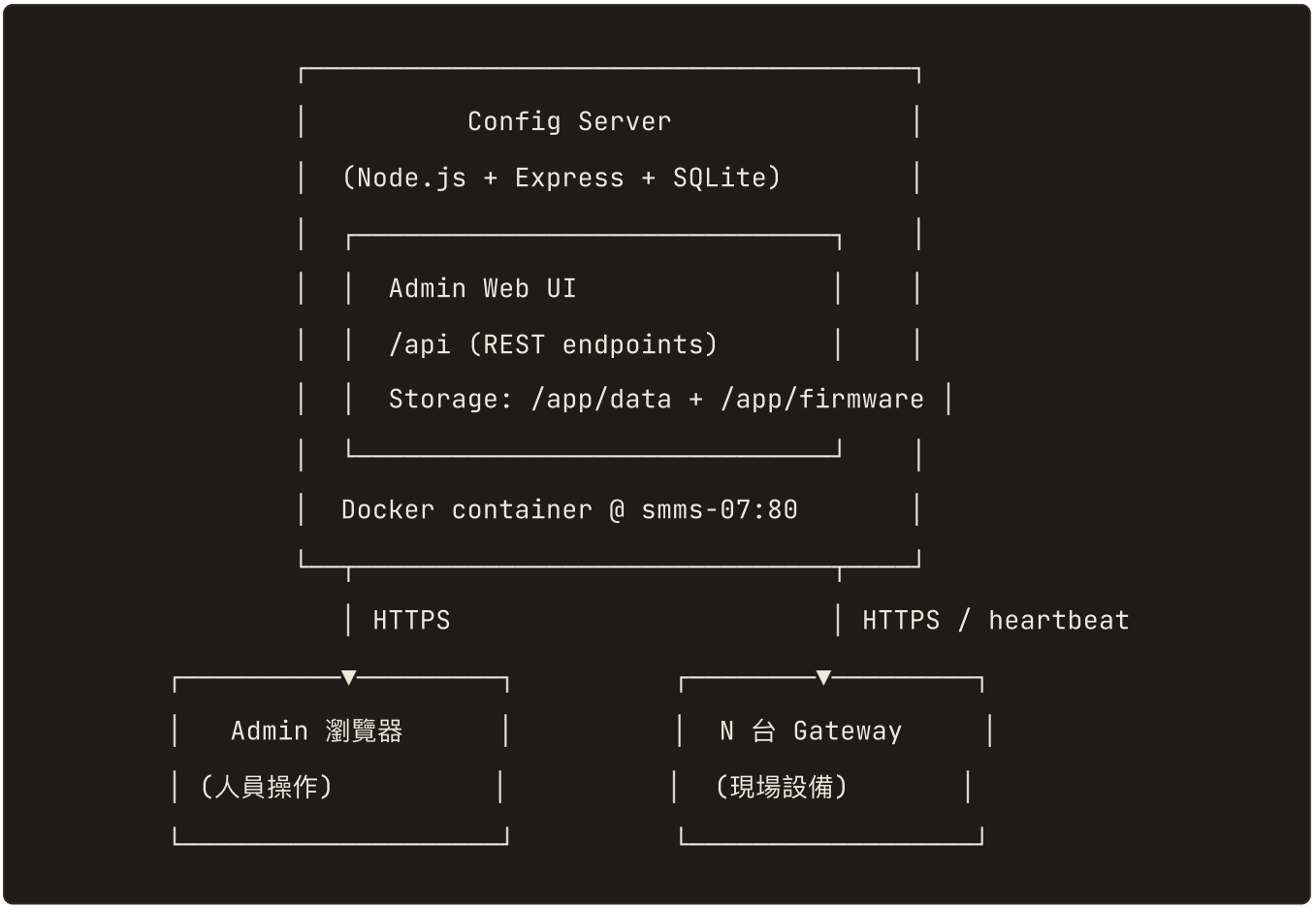
更新於 Wed May 06 2026 08:00:00 GMT+0800（台北標準時間）

> 適用版本：config-server v1.x（與韌體 v5.9.63 同步）
> 適用對象：系統管理員、後端維運、SaaS 平台管理者

雲端 Config Server 是管理多台 MES I/O Gateway 的中央後台，提供設備清冊、授權發放、韌體倉庫、設定備份、遠端 OTA 等功能。本手冊涵蓋日常 admin 操作與部署維護。

01

1. 系統定位與架構



核心責任：

- **設備管理**：認領、清冊、刪除、解除授權
- **授權**：License 簽發、撤銷、續期
- **韌體倉庫**：上傳 .bin、版本歷史、發起遠端 OTA
- **設定備份**：每台 Gateway 上傳的 config.json 快照保存（含 diff 比對）
- **MQTT bridge**（選用）：把現場 IO 資料中繼到上層系統

02

2. 部署環境

2.1 生產環境（已部署）

項目	值
主機	`smms-07` (10.0.0.9)
Docker container	`opta-config-server`
對外 URL	`https://config.metasource.tw`（透過反向 proxy + Let's Encrypt）
內網 URL	`http://10.0.0.9`
Image registry	`smmsregistry.azurecr.io/kc-opta-config-server:latest`
資料持久化	Docker volumes：`opta-data`（SQLite + backups）、`opta-firmware`（.bin 倉庫）

2.2 開發環境

```bash
cd config-server
npm install
npm start  # listen on :3000
```

預設 admin token：`admin-token`（生產環境應在 docker compose 設 `DEVICE_TOKEN=...`）

03

3. Admin 登入

打開 `http://10.0.0.9` 或 `https://config.metasource.tw`，會跳出登入框。

Admin 帳號（v2.2.4+）：

- `admin@localhost`（cloud 用）、`local@localhost`（local 用）
- 初始密碼：`DEFAULT_ADMIN_PASSWORD` env 沒設 → docker log 印一次 random；有設 → 那個值（`.env.example` 預設 `smmsadmin`）
- 第一次登入後請走右上角 🔑 **改密碼** button（或 `POST /api/auth/password`）改成強密碼

設備用 API Token（向下相容，跟 admin 登入無關）：env `DEVICE_TOKEN`，預設 `admin-token`（**生產環境必改強隨機字串**）。

登入後右上角顯示角色徽章 + 🔑 改密碼 + 登出 button。

04

4. 主要頁面

頁面	用途
📡 設備 (Devices)	看所有 heartbeat 過的 Gateway，刪除、解除授權
🎟️ 授權 (Licenses)	簽發 / 撤銷 / 續期 License
📦 韌體 (Firmware)	上傳 .bin、看 changelog、部署到指定設備
💾 備份 (Backups)	看每台設備的 config 歷史快照、diff 比對

05

5. 設備管理（Devices）

5.1 設備如何出現在清冊？

Gateway 啟用 Config Server 後，每 30 秒送一次 heartbeat 到雲端，雲端會自動建立或更新清冊（WI-098 BUG-3）。**無需手動 claim**。

5.2 設備卡片資訊

每張設備卡片顯示：

- UID（H7 的 96-bit unique ID）
- 設備名稱（用戶自訂）
- 狀態：🟢 online（最近 60s 有 heartbeat） / 🔴 offline
- 韌體版本
- 最後 heartbeat 時間
- 待派發 OTA（若有）

5.3 操作

動作	場景	注意
🚀 部署韌體	從韌體頁的「🚀 部署」按鈕觸發，設備下次 heartbeat 自動取得	OTA 失敗會 fallback 到 DFU
🗑️ 刪除設備	設備永久退役、UID 衝突	不會清掉設備本機的 config，只是雲端清冊移除
⛔ 解除授權	License 撤銷，下次 heartbeat 設備會收到並停用授權功能	Gateway 停用後僅可進入 AP 重新認領

> **大量設備清理**：列表上方有「🧹 清理離線 24h+ 設備」批次操作。

06

6. 授權管理（Licenses）

6.1 授權狀態

狀態	意義
`unseen`	License 已產生但未被任何設備啟用
`pending`	設備首次連線，等 admin 核准
`active`	已核准，設備可正常運作
`deactivated`	被 admin 解除授權，設備進入限制模式
`expired`	已過期

6.2 操作

- 🟢 **核准 (Approve)**：把 `unseen` / `pending` → `active`
- 🟠 **解除授權 (Deactivate)**：`active` → `deactivated`，設備下次 heartbeat 收到並停用控制功能
- 🔴 **拒絕 (Reject)**：把 `unseen` / `pending` 標記為拒絕
- 🗑️ **刪除 (Delete)**：永久刪除 License 記錄（任何狀態都可，含確認 modal）

> 詳見 WI-109 / WI-110 / WI-111 規格。

07

7. 韌體倉庫（Firmware）

v5.9.59 起改成 **card 排版**（WI-115d）：

7.1 上傳韌體

兩種方式：

1. **拖放區**：直接把 `.bin` 拖到頁面上方的 drop zone
2. **表單**：填版本號 + 選檔案 + 按「⬆️ 上傳」

強制要求（WI-114）：

- 必須夾帶 changelog（可在 release 腳本中用 `--changelog "..."` 帶入）
- changelog 至少 10 字元
- 透過 `X-Firmware-Changelog-B64` header 上傳（base64 編碼，支援多行 + 中文）

release 腳本（建議方式）：

```bash
python3 scripts/release_firmware.py --upload http://10.0.0.9 \
  --changelog "v5.9.x — 修補 ABC + 改善 DEF"
```

7.2 部署到設備

點某個版本卡片的「🚀 部署」按鈕 → 選目標設備 → 確認。設備下次 heartbeat 取得指令，用戶在 Gateway Web UI「設定 → OTA」按「📥 更新」開始下載。

> ⚠️ **OTA size 限制**（v5.9.63 起）：binary > 770KB 時 Gateway 端 `/api/ota/trigger` 會回 413，須改用 USB DFU 升級。Cloud admin UI 不會擋（依設備行為），但用戶會看到 toast 錯誤。

7.3 刪除韌體

- 點某個版本卡片的「🗑」→ 確認 modal
- 若該版本正在派發給設備（in-flight OTA），會回 409 + 錯誤訊息「該版本正在派發給 N 台設備，請先取消 OTA」

7.4 下載 .bin

點「⬇」直接下載，方便離線 USB DFU 燒錄。

08

8. 設定備份（Backups）

每次 Gateway config.json 變動（且啟用「每次設定儲存後自動推送」），會推送一份快照到雲端。

8.1 看快照歷史

設備卡片點「💾 看備份」→ 列出所有快照，每筆顯示：

- 時間戳
- 設定 CRC（hash）
- 變更摘要（與前一版的 diff，欄位級別）

8.2 還原

選某筆快照 → 點「↩ 還原到此版本」→ 雲端會把該版本當作 pending config，設備下次 heartbeat 拉回。

> 設定還原會覆蓋現場設定。執行前請通知現場人員。

09

9. 部署 / 升級流程

9.1 開發機 → 推送到生產

```bash
cd config-server
# 1. 打包 source（排除 node_modules 等）
tar -czf source.tar.gz --exclude=node_modules --exclude=.git --exclude=data .

# 2. SCP + SSH build + run（已包成 expect script）
expect upload_and_build.exp
```

`upload_and_build.exp` 做的事：

1. SCP source.tar.gz 到 smms-07
2. SSH 上去解壓
3. `docker build` 重 build image
4. `docker stop / rm / run` 換 container（用 `;` 串接 + `__DEPLOY_DONE__` sentinel 避免 race）
5. 確認 container `Up X seconds`

9.2 升級後 smoke test

```bash
curl -s -o /dev/null -w "Cloud HTTP %{http_code}\n" http://10.0.0.9/
curl -s -m 5 -H "Authorization: Bearer admin-token" http://10.0.0.9/api/firmware
curl -s -m 5 -H "Authorization: Bearer admin-token" http://10.0.0.9/api/devices
```

9.3 Volume 持久化

Volume	內容	何時備份
`opta-data`	SQLite (devices, licenses, backups history)	每週、重大設定變更前
`opta-firmware`	.bin 倉庫 + .json manifests	上傳新版本後

備份指令範例：

```bash
ssh smmsadmin@10.0.0.9 'sudo docker run --rm -v opta-data:/data -v $(pwd):/backu
```

10

10. 反向 Proxy + HTTPS

對外 `config.metasource.tw` 由前端 nginx 反向 proxy 到 `127.0.0.1:80`，由 Let's Encrypt 簽發憑證。Gateway 透過 RTC fallback 機制（WI-100）解決 H7 開機時間不準導致的 cert 驗證失敗。

設備端的 ISRG Root X1 cert 已內建在 firmware（`src/Cert.h`）。

11

11. API 端點速查

完整定義見 `docs/api/api-config-server.md`。常用：

Method	Path	用途
GET	`/api/devices`	設備清冊
DELETE	`/api/admin/devices/:id`	刪除設備（WI-110）
GET	`/api/firmware`	韌體列表 + changelog
POST	`/api/firmware/upload`	上傳 .bin（含 X-Firmware-Version + X-Firmware-Changelog-B64）
DELETE	`/api/admin/firmware/:filename`	刪除韌體（WI-110）
POST	`/api/firmware/deploy/:deviceId`	派發 OTA
GET	`/api/firmware/download/:filename`	下載 bin
GET	`/api/licenses`	License 列表
POST	`/api/licenses/:id/approve`	核准
POST	`/api/licenses/:id/deactivate`	解除授權（WI-110）
POST	`/api/licenses/:id/reject`	拒絕
DELETE	`/api/licenses/:id`	刪除
GET	`/api/snapshots/:deviceId`	設定備份歷史
POST	`/api/heartbeat`	Gateway 上傳 heartbeat（無需 admin token，憑 license）

所有 admin 端點需 `Authorization: Bearer <admin-token>`。Gateway 自身的 heartbeat 用 `Authorization: Bearer <license-id>`（WI-098 + WI-065 中介層）。

12

12. 常見問題

12.1 設備在清冊上但 status 永遠 offline

可能原因：

- Gateway 端 Config Server URL 錯誤 / 未啟用「☁️ 雲端 Server」
- Gateway 端網路斷線（檢查 sysConn 狀態）
- License 已 deactivated（heartbeat 會被 reject）

排查：用 SSH 進 smms-07 看 docker logs：

```bash
ssh smmsadmin@10.0.0.9 'sudo docker logs --tail 100 opta-config-server'
```

找對應 UID 的 heartbeat 紀錄。

12.2 上傳韌體失敗

最常見：changelog 太短（< 10 字元）→ 回 400 + `"Changelog too short"`. 修正後重 upload。

12.3 OTA 部署後設備一直在 pending

設備可能：

- heartbeat 還沒拉到（30s 間隔）
- 用戶還沒在 Gateway Web UI 按「📩 更新」（OTA-V2 是 user-triggered，不是 push）
- 韌體 binary > 770KB → Gateway 端 OTA guard 回 413，需改 USB DFU

12.4 Cloud UI 上的 firmware card 沒看到 changelog

舊版上傳的韌體（WI-114 之前）沒帶 changelog，會顯示 `(legacy: ...)`。新版必填，從 `release_firmware.py` 上傳會強制夾帶。

12.5 想砍掉某個 firmware 但 in-flight OTA

回 409 時，先在「設備」頁找到等待 OTA 的設備 → 取消 OTA → 再回韌體頁刪。

13

13. 維護清單

頻率	項目
每週	備份 `opta-data` volume + `opta-firmware` volume
每月	Docker image 重 pull / 升級基底
每季	清理 > 6 個月離線設備
每年	Let's Encrypt cert 自動續期確認（certbot）

14

14. 相關文件

- `docs/guides/guide-config-server-cloud-deployment.md` — 從零部署到 Azure / 自架
- `docs/guides/guide-config-server-backup.md` — 備份還原 SOP
- `docs/api/api-config-server.md` — 完整 API 規格
- `docs/sprints/cards/sprint-27/wi-110-cloud-admin-p0-endpoints.md` — admin endpoints 設計
- `docs/sprints/cards/sprint-27/wi-111-cloud-admin-ui-buttons.md` — UI 按鈕設計
- `docs/sprints/cards/sprint-27/wi-114-firmware-changelog-system.md` — changelog 強制夾帶機制
- `docs/sprints/cards/sprint-27/wi-115-firmware-login-password-rename.md` — 帳號 / 密碼系統

若 Cloud Server 行為與本手冊不一致，請以 git log + 現行 source code 為準，並回頭修訂本手冊。

04 開發者 / 整合

韌體變更紀錄機制

v6.0.83 — 2026-08-28 — WI-211b 通知改 QoS 0 + 週期重發兜底 + 壓制窗 1 秒

v6.0.82 上機後業主回報「進步很多,只卡了一次」。追那一次:

```
t=49817  ⛔ 卡 1009ms   最大段 mqtt=1007ms   ← 先卡
t=49818  🔌 MQTT 重新連線                    ← 才斷線
```

`1007ms` 這個數字在先前多份 log 裡反覆出現(1006/1007/1008)—— 又是固定逾時的指紋。翻函式庫:`MQTTClient.h:75 uint32_t timeout = 1000;`。

真因是網路延遲,不是爆發:QoS 1 每則發送都要等 broker 回 `PUBACK`,broker 在 網際網路上,只要那一秒剛好塞車、ack 沒回來 → 函式庫判定失敗 → 連線被當成死的 → 重連。佇列沒滿、合併正常運作 —— 這一次純粹是外部延遲。

01

業主的提案與最終決定

業主問:「採取 QoS 0,然後發送的時候發兩次呢?」

發兩次幫助有限 —— 兩份相隔幾毫秒的複本會**一起死在同一條斷掉的連線上**, 而連線斷掉正是 QoS 0 唯一會發生的遺失情境。它擋不到唯一會發生的那種遺失, 卻讓通知量與主迴圈阻塞加倍。

正解是週期重發現況:涵蓋所有遺失原因,成本每分鐘一則,而且給下游一個 **有界的過時上限** —— 那才是能寫進合約的東西。業主採納。

02

三個改動

1. `rules/*/trigger` **改 QoS 0** —— 沒有 ack 要等,1 秒逾時的故障模式直接消失。對一個**設計上就可丟**的通知(設備已刻意合併中間值),付 ack 的代價換保證本身就不一致。
2. **週期重發**:每 60 秒把每個狀態類 topic 的現況原樣重送一次(期間有更新則不重送)。已被週期推送的資料(RS485 值)會不斷刷新時間戳,自然不會觸發 —— 機制自我調節。
3. **壓制窗 500ms → 1000ms**:發送次數再減半。

另修:訂閱的日誌誤印成發送 QoS(`Subscribed to ... (QoS 1)`)—— 實際訂閱用的一直是 `subQos`(=2,正確),只是 v6.0.82 做變數改名時把那行的變數換錯了。行為無誤,日誌誤導。

03

實測(同一台 .77,30 個事件 5 秒內)

版本	飛出訊息	重連	佇列滿	mqtt 段最大卡頓
v6.0.79	20 則	**4 次**	有	**4398ms**
v6.0.82	16 則	0 次	無	501ms
v6.0.83	**10 則**	**0 次**	**無**	**無 >150ms**

週期重發實測:最後一次發送的 **+66 秒**,兩個 topic 的現況都自動重送。

---

v6.0.82 — 2026-08-27 — WI-210/211 快速切換打掉 MQTT:QoS 配合語意 + 同 topic 合併 + payload 夾帶歷史

業主人工驗收時快速切換 Toggle 開關十幾次,呼吸燈進入快閃(=**MQTT 斷線中**,不是重開)。追下去是一條完整的故障鏈:每條規則觸發都發一則通知 → 邊緣模式不節流 → 灌爆 16 格佇列 → 每則 TLS 發送阻塞主迴圈 265~~400ms → 連線撐不住 → 重連風暴(握手 1.6~~4.4 秒且逐次變長) → 若持續約 90 秒,self-ping 連兩次無 echo → **設備自我重開**。

同一個故障鏈 `main.cpp:2620` 的註解已描述並修過一半 —— 但只節流位準模式, 明確寫著「邊緣觸發不節流,每個事件都有意義不可丟」。那句是破口:對慢速事件成立, 但人手快切或現場**抖動的接點**的邊緣密度足以複製同一個故障(本站 `debounceMs=0`)。

業主定下的原則:**「控制是現場的問題,訊息可以晚到。」**依此重做發送路徑。

04

一、QoS 配合資料語意(最大的槓桿)

`defaultQos` 寫死 2,而 `setDefaultQos()` **全庫從來沒有人呼叫過** —— `config.mqtt.qos` 早就有欄位、有存檔、Converter 也在用,就是沒接到管理器上。

發出去的絕大多數是「最後狀態」:**重複送達會被下一筆覆蓋,無害** —— 需要的是「一定會到」,不是「恰好一次」。用 QoS 2 等於付**兩趟往返**買一個用不到的保證。

→ 發送預設 **QoS 1**、訂閱維持 **QoS 2**(命令不可重複執行),兩者皆可在網頁設定。

05

二、同 topic 合併 + 每 topic 最小發送間隔 500ms

同一個 topic 還在佇列裡就直接取代,**並移到隊尾** —— 移到隊尾是關鍵: 狀態由「哪個 topic 最後更新」表達,原地取代會讓先進佇列的舊 topic 晚於後更新的送出, 下游停在**錯的最終狀態**。

最小間隔同樣必要:沒有它, `drainQueue()` 每 50ms 排空,佇列永遠只有 0~1 筆, 「同 topic 取代」等於不存在 —— **實測 20 則全數飛出,零合併**。

單次變化不受影響(該 topic 若 500ms 內沒送過就立刻送),只有連續變化才被壓。

06

三、payload 夾帶歷史(opt-in,預設關)

```JSON
{"v":"A2-MQTT滅","t":1787839126,"h":[[354,"A2-MQTT滅"]]}
```

`v` =最後狀態、`t` =本批起點 epoch 秒(0=時鐘未同步)、`h` = `[相對毫秒, 當時狀態]`。每筆帶**時間與狀態**。用相對毫秒 + 單一絕對錨點,是因為 payload 硬上限 128 bytes, 且時鐘未同步時仍要可用。超過 8 筆標 `trunc:true` ,誠實告知不完整。

合併因此從有損變無損 —— 中間過程不是丟掉,是換個位置帶走。 預設關閉:開啟會把 payload 從裸值變 JSON,既有解析端會爆。升級**不會**自動開啟。

07

實測(同一台 .77,30 個事件 5 秒內)

	重連	佇列滿	mqtt 段最大卡頓
v6.0.79(改前)	**4 次**	**有**	**4398ms**
v6.0.82(改後)	**0 次**	**無**	**501ms**

08

量測方法踩到的兩個坑

- **用 `mosquitto_pub` 逐次呼叫做「爆發」是假的**:每次都是新行程 + 完整 TLS 握手, 實際只有 2~3 則/秒,比壓制窗還慢,合併根本不會被觸發。要用 `-l` 單一連線連續送。

- **先前用 QoS 2 進向做壓力測試,測到的是進向握手成本**,不是出向通知成本 —— 業主的情境是純出向(撥開關),兩者不可混為一談。

另:MQTT 設定改為每 3 秒收斂到管理器。原本只在建立連線時套用, 使用者在網頁改了 QoS 或歷史夾帶卻沒重連,設定永遠不生效(實測踩到)。

---

v6.0.79 — 2026-08-27 — WI-208b 雲端每次心跳重送舊授權,設備每次白驗簽兩次

業主在 v6.0.78 上回報「呼吸燈大概 115 次會卡一次」。心跳燈每 500ms 翻一次 (一次呼吸 = 1 秒),所以那是**約 115 秒一次** —— 對得上雲端心跳的 127 秒週期。

實測把每次卡頓與 log 事件對起來,關聯性零例外:

```
t=3019 ☁️ CS-Cloud Connecting → t=3029 📜 license/state rejected → t=3029 ⛔ 卡 1334ms
t=3146 ☁️                     → t=3153 📜                        → t=3153 ⛔ 卡 1340ms
t=3341 ☁️                     → t=3350 📜                        → t=3350 ⛔ 卡 1341ms
t=3407 ☁️                     → t=3414 📜                        → t=3414 ⛔ 卡 1338ms
```

因果鏈:設備心跳連雲端 → 雲端回頭發一則 `license/state` → 該 token 序號固定 (`tokenId=1786621672`) 且**永遠被判 stale 丟掉** → 但丟掉之前付了**兩次**驗簽: `licenseTokenAcceptAndStore()` 驗一次(663ms),它內部為了讀「既有序號」呼叫 `licenseTokenStoredTokenId()` 又**完整驗簽一次**(663ms)—— 只為了取 4 個 byte。`663 × 2 = 1326ms`,與實測的 1330ms 完全吻合。

三項修正:

1. **收訊前先看序號,不新就不驗**。新增 `licenseTokenPeekTokenId()`,只解析標頭、零密碼學運算。序號不比手上的新 → 驗了也是拒,直接丟。
2. **既有序號改用快取**(`licenseTokenStoredTokenIdCached()`)。序號只在存入新 token 時改變,沒有理由每次重算。
3. **只有真的存了新 token 才重評門控**。原本無條件 invalidate + refresh, 等於每則被拒的訊息再多付一次驗簽。

> ⚠️ **同時修掉一個 v6.0.78 引入的正確性 bug**:WI-208 的三個快取變數被寫成 header 裡的 `static` —— 那是「每個 .cpp 各一份」,而 `licenseGateInvalidate()` / `licenseGateRefresh()` 是 `inline`(外部連結),連結器只留一份定義並綁到其中一個 .cpp 的變數。結果**兩者可能在動不同的變數**:實測 v6.0.78 收到 license/state 後 明明呼叫了 invalidate,卻完全沒有重驗。那不只是效能問題 —— **存入新 token 後門控不會重新評估,撤銷授權會延遲到重開才生效**。改為 `extern` + 在 main.cpp 定義(比照同檔既有的 `g_licenseEnforce`)。**教訓:header 裡的 `static` 配 `inline` 函式,是 ODR 陷阱。**

實測(v6.0.79,544 秒觀測窗、涵蓋 **8 次**雲端心跳):

```
net 卡頓 0 次 → 0.00%
[WI-145] license/state 略過(序號不新,未驗簽) tokenId=1786621672 已存=1786621672
```

主迴圈凍結的完整歷程:**6.8%(v6.0.74)→ 1.7%(v6.0.78)→ 0.00%(v6.0.79,net 段)**。`/api/license` 全程 `status=ACTIVE, licensed=true`。

根因仍在雲端:`publishLicenseState` 每次心跳都重送一則設備早就拒絕過的舊 token。設備端現在擋得很便宜,但雲端不該一直發。已記在 WI-208 卡上,未修。

---

v6.0.78 — 2026-08-27 — WI-208 授權門控每 30 秒凍結主迴圈 663ms(+WI-207 語系鍵)

業主回報「藍色呼吸燈會卡住,卡住的時候按鈕就延遲、甚至沒反應」。用 WI-163 的段別儀器 量了 210 秒:`net` 段卡頓 15 次、合計 14.4 秒 —— **主迴圈有 6.8% 的時間完全凍結**, 期間呼吸燈不更新、DI 不被輪詢,實體按鈕的動作就這樣被吃掉。卡頓時長只有兩種且極度規律 (661~671ms / 約 2050ms),是固定成本的指紋而非負載。

`net` 段裡有兩個 30 秒任務且在**同一輪迭代**,靠時間軸分不開,故先上臨時儀器各自量測:

```
lic=659/665/663/662ms   ← licenseGateRefresh()
wifiStat=0ms            ← 診斷用 WiFi.status()
failoverStatMax=0ms     ← 沒有時間閘門、每 loop 都戳的那個
```

兩處 `WiFi.status()` 都是 0ms —— 原本最被懷疑的它是無罪的(程式碼裡那段註解描述的是 已經修好的舊病)。再拆一層:`load=1ms verify=663ms`,成本全在驗簽。

修正:門控 verdict 是 `(token 內容, hwUid, nowSec)` 的純函式。前兩者只在存 token 時改變 (那些路徑改為呼叫 `licenseGateInvalidate()` 強制重驗), `nowSec` 只能讓「未到期」翻成「到期」。因此改為**到期驅動** —— 只在「還沒驗過」或「先前有效且已跨過 `expiresAt`」時 才真的重驗;到期/簽錯/UID 錯都是終局,標記後不再重算。NTP 首次同步成功時一併失效快取, 避免「校時前誤判到期並鎖死」。

> ⚠️ 第一版(未出版的 6.0.76)完全沒生效:本站是永久授權(`expiresAt=0`), 而判斷式寫成 `nowSec < expiresAt` → `nowSec < 0` 永不成立 → 照樣每 30 秒重驗。**`expiresAt == 0` 的語意是「永不到期」**,韌體原本的算式 (`out.expiresAt != 0 && ...`)早就寫明了。終局判定補上這條才真正生效。

實測:主迴圈凍結 **6.8% → 約 0.8%**(net 段),665ms/30s 的卡頓完全消失, `/api/license` 仍為 `status=ACTIVE, licensed=true`。

同版順帶修掉 **WI-207**:`io.saving` 這個 i18n 鍵三個語系檔全缺,而 `t()` 查不到時回傳 鍵本身(truthy),讓 `|| "儲存中"` 這個 fallback 變成死碼 —— 存檔佇列徽章因此顯示「💾 bH 1」而不是「💾 儲存中 1」。三份語系檔補齊,實機確認已對到「儲存中」。

尚未解決:`net` 段仍有約 1.3 秒的偶發卡頓,與對外 TLS(`[CS-Cloud] Connecting`)同時發生,屬「對外連線在主迴圈裡做」的老家族,要動架構才能根治。

---

v6.0.74 — 2026-08-26 — WI-197 回復重開的標記寫錯方向

`intentReset()` 在**有線回復**路徑上標的是 `net-failover` —— 但那次重開是「WiFi → 有線」,與 failover 的「有線 → WiFi」**方向相反**。同一台設備連續兩次相鄰重開因此標記相同,事後看 `bootDetail` 分不出它是掉線還是恢復。v6.0.1 建 BKPSRAM intent marker 的初衷,正是因為 Opta(MCUboot)開機前會清掉 `RCC->RSR`、硬體 reset 旗標救不回來,只能靠自己 留字條 —— 字條寫錯方向,這個機制在這條路徑上就等於白做。回復路徑改標 `net-recovery`。全庫三處寫入該字串,另兩處(452/472)是名副其實的 failover,不動;確認無任何地方解析 這個字串,改名安全。

---

v6.0.73 — 2026-08-26 — WI-206 Parser「值類型」下拉三個選項全部對錯韌體列舉

韌體 `ParserValueType` 是 `BOOL=0 / NUMBER=1 / STRING=2`,而網頁下拉是 `0=數值 / 1=字串 / 2=布林` —— **三個都錯,沒有一個對**。使用者選「數值」送出 `0`,韌體當成布林,**任何非零數字都變成 `1.0`**。而 Parser 的主要用途就是抽數值、新增解析欄位的預設 type 又正好是 `0` —— 不動下拉、直接用預設的人一定中獎。拿到的 `1.0` 看起來像個正常的值、不會報錯,後續拿去做門檻比較或塞進 Converter 發佈,送出去的就是一串恆為 1 的假資料。

修的是**前端**不是韌體列舉:`parserType` 是已落盤的持久化契約,改列舉會讓既有設備上 已存的值被重新解讀成別的型別。同時把新增解析欄位的預設由布林改為數值。**遷移影響**:既有通道行為不變,但畫面顯示的型別會變成它一直在做的那個 —— 升級後請回頭檢查 Parser 通道的型別設定。

---

v6.0.72 — 2026-08-26 — WI-200 直接控制 API 涵蓋擴充板輸出(順帶修好一個從沒被測到的舊 bug)

`POST /api/io/do` 舊碼只認主機 DO 0-3,送擴充板一律回 `Invalid index` —— 但規則引擎 明明驅動得了擴充輸出。同一個能力規則路徑有、直接控制的 API 沒有,出廠測試與現場調機 想單獨點一路擴充輸出只能繞道「改輸出群組 → 借規則 → 發 MQTT → 再改回去」。現在接受擴充 index(`16+ch` / `32+ch` …),錯誤訊息附合法範圍,模組不在時明確報錯。

順帶:沿用共用的 `modbusTcpWriteDO()` 時發現它的擴充分支本身是壞的 —— `auto ext = OptaController.getExpansion(i); (DigitalExpansion*)&ext` 是**對區域暫存副本 取址再向下轉型**。也就

是說 **Modbus TCP 寫擴充輸出一直沒有作用**,而且從沒被任何測項 涵蓋到。規則引擎那份用的是 `getExpansionPtr`(實測會動),兩份實作走樣多時。統一成已驗證有效的那條路 —— 這也修好了 Modbus TCP 那一側。

---

v6.0.71 — 2026-08-26 — WI-203 存 Converter 模板後長度一併落盤

`saveTemplate()` 把模板文字 `fwrite` 進 `/cfg/tmpl/{idx}.json`(立刻落地),同時更新 `config` 的 `converterTemplateLen` —— 但**只在記憶體**,沒有 `setConfigDirty()`。長度要靠之後某次別的設定存檔順帶刷進去;而網頁的儲存順序是先 `POST /api/tcpio`(會刷盤)、再 `POST /api/converter/template`,剛好讓長度更新落在刷盤之後 → 重開後讀回舊值。舊值若恰好是 `0`,`GET /api/tcpio` 的內聯模板匯出(WI-133,供跨機 複製整組帶走換算公式)會變成空字串,而檔案裡明明有內容 —— **靜默遺失公式**。

---

v6.0.70 — 2026-08-26 — 三個「回應說一套、實際做另一套」的缺陷

WI-205(高,破壞性)— 被拒絕的規則 POST 會毀掉既有規則 `POST /api/rules` 的解析**直接寫進** `config->rules[idx]`,WI-174 守衛在解析之後才檢查 並回 400,**而且不回滾**。舊碼的緩解只有 `if (idx >= config->ruleCount) rule->enabled = false;` —— 只涵蓋新槽位。覆蓋既有規則時既不回滾也不停用,結果:一個回 400 的請求把使用者 原本的規則銷毀,並留下一條 `sig=0/act=0` 卻 `enabled=true` 的死規則。使用者收到 400 會合理推論「我的操作沒生效」—— 實際上它生效了,而且是破壞性的。修法:進 handler 時整份備份,拒絕路徑完整還原。拒絕就是什麼都沒發生。

WI-202(高)— 一個引號讓整頁設定人間蒸發 22 處使用者可編輯的字串欄位未做 JSON 跳脫。名稱或設定值裡出現一個 `"`,整包 API 回應 就變成不合法 JSON,網頁 `JSON.parse` 失敗後**靜默 render 成空清單** —— 使用者看到「尚無 TCP IO 通道」,以為自己沒有設定過。HTTP 仍是 200,畫面上沒有任何錯誤。含 WiFi SSID、MQTT 設定、規則與群組名稱、TCP IO 各欄位。`currentStrValue` 的值來自**進來的 MQTT payload** —— 任何能發訊息到該設備訂閱 topic 的人,payload 帶一個引號就能讓設定頁消失。寫入端同步修:`parseStr` 用 `indexOf("\"")` 找「下一個引號」、不認跳脫,值含引號就被 截斷在跳脫序列中間(實測壞資料 `{\` 正是如此)。新增 `jsonExtractString()`。讀寫兩端是一對,只修一邊都不夠。

> 這個跳脫 helper 是 **WI-191** 為了同一種事故建立的(`/api/poll` 設備名被截斷 → 整頁數值消失)。專案已經吃過一模一樣的虧、修好了一處,兄弟端點卻從沒跟上。

WI-199(中高)— 連續快速新增規則會靜默覆蓋前一條 `saveRule` 在 save mutex **外**用本地快取 `rules.length` 算新規則 index,而存檔後的 `loadRules()` 沒有 `await`、不在 mutex 內。連續新增時第二次在第一次的刷新回來之前 取得 mutex → 算出同一個 index → 第二條覆蓋第一條,**兩次都回 200**。修法:index 改在 mutex 內、`await loadRules()` 之後才算;刷新也改成 `await`。`saveGroup` 同源一併修。

---

v6.0.69 — 2026-08-25 — WI-193 插線不自動切回有線 / WI-194 恢復原廠後雲端還原被 401 擋掉

兩個缺陷都是「同一個保護只做在部分路徑上」,而漏掉的那條剛好是現場最常見的。

WI-193 — AUTO 模式「開機時就沒線」不會進入有線回復監看 `interfaceMode=AUTO` 的註解是「有線優先 + WiFi 備援」,但介面選擇只在 `NetworkStateManager::initializeNetwork()` 跑一次。執行期的 failover 狀態機**確實寫了** 回切有線的路(`FO_WIFI_ACTIVE → FO_RECOVERY_PENDING → 重開走 Ethernet`),但唯一入口 `enterRecoveryMonitoring()` 只在 `if (isFailoverBoot)` 下被呼叫 —— 也就是只認「曾在有線上、線被拔掉 → 重開」那一條。**開機時就沒插線**而 fallback 到 WiFi 的機器 不設旗標,`_foState` 停在 `FO_IDLE`,而 `FO_IDLE` 的監看條件是 `_activeNetwork==NET_ETHERNET` —— 跑在 WiFi 上永遠不成立。實測:插上網路線 11.8 小時完全無反應,人工重開才切過去。 修法:啟用條件改成守性質 —— 「AUTO 模式且現在跑在 WiFi 上」。`WIFI_ONLY` 與 AP 模式排除。

WI-194 — 恢復原廠之後,雲端還原永遠失敗且無聲 L2 恢復原廠會把 `cloudToken` 清成空字串,而 `downloadConfigFrom()` 在 token 為空時 **不送 Authorization**,雲端該端點要求授權 → 401。雲端那側一切正常(rollback 回 `success`、`pendingCommand` 送達並被消耗),於是「後台顯示還原成功、設備什麼都沒做、兩邊都沒有錯誤提示」—— 而且正好命中人最需要還原的那一刻。 推送與心跳早就有「token 空就改用 licenseId」的 fallback,只有下載漏掉。 修法:補上同一段 fallback(雲端 `middleware/auth.js` 本來就接受 license token,不需改); 下載失敗改印出目標與 HTTP 狀態碼,401 額外提示去填 Cloud Token。

同時發現、尚未修(已開卡):

- **WI-195** — `autoPush` 會在數秒內覆蓋掉雲端剛寫入的還原,使還原變成無效操作。
- 發佈程序文件的 `.mesb` 資產清單漏了 `log.html.gz`(實際應為 8 個),已更正。
- L2 恢復原廠會連 QSPI 網頁資產一起清掉,設備只剩「資產救援」頁 —— 已寫進驗收單 § R.4。

---

v6.0.68 — 2026-08-21 — WI-192 WiFi 連上卻沒拿到 IP(0.0.0.0)的有上限自救

設備開機快於上游網路就緒時,會連上 WiFi 但拿不到 DHCP,停在 `0.0.0.0` 不會自己恢復。 加入偵測與自救重開。第一版節奏設計錯誤 —— 會剛好在 router 準備好的那一刻放棄, 改為 5 分鐘首次寬限 + 加倍退避 + 移除次數上限。

---

v6.0.67 — 2026-08-21 — WI-191 JSON 字串未跳脫,一個壞名稱會讓整個畫面空白

設備名稱/通道名稱等自由文字未做 JSON 跳脫,含有引號或反斜線的名稱會產出不合法 JSON, 前端解析失敗 → 整頁空白。修:輸出前一律跳脫。

---

v6.0.66 — 2026-08-21 — WI-190 存檔序列化不完整會把設定連同備份一起毀掉

現場事故(2026-08-20 22:05:18,序列埠 log 完整記錄): 設定檔從 6332 bytes 變成 2108 bytes,主檔與備份在同一秒被同一份殘缺內容填滿, 設備等同回到無設定狀態 —— 隔天現場 RS485 四台掃描得到卻一台都讀不到值,因為「已設定的設備」是 0 台。

失效機制: ArduinoJson v7 的 `JsonDocument` 記憶體不足時**不會報錯**,只是裝不下。`serializeJson` 仍吐出格式完全合法、CRC 完全正確的 JSON,只是內容少一大半。於是它通過了下游每一道檢查 —— `configJsonIsValid()` 只驗格式與 CRC,不驗完整性。接著「備份不可用 → 用本次內容重建」把唯一的好備份也覆蓋掉,救援路徑自我摧毀。WI-180b 的兩道防護前後矛盾:上一行才判定「主檔內容不完整」,下一行就拿同一份內容重建備份。

修正:

1. 存檔前對帳 —— `doc.overflowed()` 加上「宣告數量 vs 實際寫出筆數」比對(rules/actions/tokens)。對不上就**拒絕存檔,主檔與備份都不動**。取捨:不存 → 這次修改沒生效,使用者會發現;存下去 → 設定全毀,沒有人會發現。
2. 重建備份前先驗證**新內容本身**;沒過就不重建 —— 壞的備份仍勝過用壞內容覆蓋。

教訓:「格式正確」不等於「內容完整」。 唯一可靠的判準是拿序列化結果跟記憶體裡 真正持有的數量對帳。

韌體 Changelog 指南 (WI-114)

> 從 cloud server 5.9.50 起,**每次上傳韌體都必須夾帶 changelog**。沒帶或太短 (< 10 chars) 會被 cloud 直接 400 拒絕。設計目的:避免 manifest / git log / wiki 等多處紀錄碎片化。

09

為什麼這樣設計？

單一來源原則:

```
release_firmware.py        →    Upload API (header)    →    Cloud Manifest    →    Admin UI
  --changelog ...                 X-Firmware-Changelog-B64    firmware.json          顯示
  (or auto-git-log)               (base64-encoded UTF-8)      changelog field         /api/firmw
```

- **唯一寫入點** = `release_firmware.py`
- **唯一儲存** = cloud `/app/firmware/<version>.json` manifest 的 `changelog` 欄位

- **唯一顯示** = admin UI 韌體列表展開
- 不維護獨立 markdown / wiki / git 抓取（會跟 manifest 不同步）

10

三種使用方式

A) 直接字串（單行 / 短描述）

```bash
python3 scripts/release_firmware.py --bump patch --upload https://opta.smms.com.tw \
  --changelog "WI-114: 強制 changelog upload + admin UI 展開"
```

B) 從檔案讀（多行 markdown）

最推薦的方式 — `CHANGELOG.md` 跟 commit 一起入 git：

```bash
# 寫一個 CHANGELOG.md（隨手寫，Markdown 格式）
cat > CHANGELOG.md <<EOF
## v5.9.50 — 2026-05-05

### Features
- WI-114: Cloud changelog 強制夾帶（單一來源防碎片化）
- Cloud admin UI 韌體列表加展開區塊

### Bug Fixes
- 修正 X-Firmware-Changelog header 多行字元編碼

### 升級注意
- 新版 cloud 拒絕沒 changelog 的 upload
EOF

# 用 @ 字首把檔讀進去
python3 scripts/release_firmware.py --bump patch --upload https://opta.smms.com.tw \
  --changelog @CHANGELOG.md
```

C) 從 stdin 讀（CI / pipe）

```bash
git log v5.9.49..HEAD --pretty='- %s' --no-merges \
  | python3 scripts/release_firmware.py --bump patch --upload https://opta.smms.com.tw \
      --changelog @-
```

D) 自動從 git log 產生（沒帶 `--changelog` 時）

```bash
python3 scripts/release_firmware.py --bump patch --upload https://opta.smms.com.tw
# 不帶 --changelog 時：
#   1. 嘗試 `git log v<prev>..HEAD --pretty='- %s' --no-merges`
#   2. 若 v<prev> tag 不存在，fallback 到最近 5 個 commits
#   3. 全部失敗 → 互動式 prompt 強制要求輸入
```

11

編寫慣例（建議）

不強制，但 admin 看 changelog 時的可讀性很重要：

```markdown
## v<version> — <YYYY-MM-DD>

### Features
- WI-XXX: short description
- 關鍵新功能或 endpoint

### Bug Fixes
- 修了什麼

### 升級注意 / Breaking Changes
- 客戶端需要做什麼動作（重 login？重新校時？）

### 內部變更
- refactor / 文件 / 工具

### Deploy notes
- ⚠️ Cloud 需要重啟 docker container？
- ⚠️ 此版本之後不允許降級？
```

> 對應「升級注意」段落，下次可考慮在 manifest 加 `breakingChanges: bool` 讓 admin UI 標紅醒目（sprint-28 候選）

12

Cloud API 變動（v5.9.50+）

POST `/api/firmware/upload`

新 header（推薦）：

```
X-Firmware-Changelog-B64: <base64-encoded UTF-8 string>
```

- 支援多行（`\n`）+ 中文 / 日文 / 任何 UTF-8
- 解碼後 trim 至少 10 字元

舊 header（向後相容）：

```
X-Firmware-Changelog: <plain string, URL-encoded>
```

- 適合單行短描述
- 不建議帶 newline / CJK

400 失敗回應：

```JSON
{
  "error": "Changelog required",
  "message": "Send X-Firmware-Changelog-B64 (base64 UTF-8, recommended for multi-line/CJK) or
  "examples": {
    "base64": "echo -n \"WI-114: bug fixes\" | base64",
    "cli": "release_firmware.py --changelog \"...\" or --changelog @CHANGELOG.md"
  }
}
```

GET `/api/firmware`

回傳列表，每筆含完整 `changelog` 欄位（可能多行 UTF-8）。

13

既有韌體（5.9.49 以前）

Backfill 一次性處理：在 cloud 升 5.9.50 之前，sprint-27 已執行：

```JS
// 對 changelog 缺失或太短的 manifest 補上
m.changelog = "(legacy, no changelog record — see git log v5.9.X)";
```

4 筆被補（5.9.8 / 5.9.40 / 5.9.41 / 5.9.42），其餘 9 筆原本就有有意義的 changelog。

> Admin UI 對 `(legacy` 開頭的 changelog 顯示為灰色斜體，方便辨識

14

Troubleshooting

Q1: `release_firmware.py` 沒帶 `--changelog`，又沒 git tag？

腳本順序：

1. `git log v<prev>..HEAD` — 嘗試找 prev tag（沒有 tag 表示沒打 release tag）
2. 若失敗，fallback `git log HEAD~5..HEAD`
3. 若仍失敗（lab 機器沒 git），互動式 prompt 強制輸入

如果連互動式 prompt 也沒人輸入，最終 sys.exit(1) — 不會白編譯 + 丟個沒記錄的 binary 上去。

Q2: 我已經 build 好 .bin 了，可以直接 curl upload 嗎？

可以，但要自己處理 base64：

```bash
CHANGELOG=$(cat CHANGELOG.md | base64)
curl -X POST https://opta.smms.com.tw/api/firmware/upload \
  -H "Authorization: Bearer admin-token" \
  -H "Content-Type: application/octet-stream" \
  -H "X-Firmware-Version: 5.9.50" \
  -H "X-Firmware-Changelog-B64: $CHANGELOG" \
  --data-binary @release/mes-gateway-v5.9.50-20260505.bin
```

Q3: Changelog 改完了想覆寫 manifest？

目前沒 PUT/PATCH endpoint。兩種選擇：

- 直接 SSH 到 cloud 改 `/app/firmware/<version>.json`（admin 可做）
- 重新跑 release_firmware.py（會以新時間戳產出新 .bin，舊的還在）

> 未來如果有需求，可以加 `PATCH /api/admin/firmware/:filename/changelog` endpoint（sprint-28 候選）

Q4: changelog 欄位 size 上限？

Cloud server 沒設限（manifest JSON 寫多大都可以）。實務上保持在 5KB 內，admin UI 才好看。

15

近期版本重點（v6.0.41 ~ v6.0.65 — 穩定性補強與「靜默失敗」清剿）

> 這一段的主題可以用一句話概括:**把「回成功但其實沒發生」的路徑一條條堵掉。**這類缺陷比功能壞掉更難查 —— 畫面正常、API 回 200,只有行為不對。

韌性 / 不再自己把自己弄死

版本	內容
6.0.41	WI-164 重開前刷寫統計(否則業主看到的計數會歸零);probe 預設 8N1;掃描逾時
6.0.44	WI-167 Modbus TCP server 改可設定且**預設關**。實測 WiFi inbound 失敗率 13% → 0%(省下 1 個 TCP socket,全機只有 4 個)
6.0.46	WI-169 WiFi 帳密被無聲清空的**三條路徑**(只有一條原本有守衛);WI-168b 卡點雙槽防 torn-write
6.0.51	**WI-175** OTA 撞上背景雲端 TLS 必掛。三次 OTA 兩次死在 `Pre-unlink` 後 7.8x 秒(硬體 IWDG 8s 窗)。修:`unmount→mount→fopen` 每步餵狗 + 進場前等雲端連線落地
6.0.57	**WI-180/181** 設定檔原子存檔(暫存檔→讀回驗證→rename);備份只在主檔完好時才降級;WiFi 帳密獨立存 `/cfg/wifi.cred`,設定全毀時仍連得上 → 可遠端救援
6.0.61	**WI-183** 對外連線硬性總時限 + 卡死自我復原。黑洞實測:`connect()` 可卡 5 分鐘不返回,`setSocketTimeout` 完全管不住

靜默失敗清剿

版本	內容
6.0.48	WI-173 `/api/modbus` 不再說謊(回報實際 RS485 參數,而非無關舊物件的硬編值)
6.0.51	**WI-174** 規則可存成 `signalId=0` → 回 200、列表正常、**永遠不會觸發**。前後端都擋
6.0.55	**WI-179** 欄位名打錯不再靜默照收。`{"channel":0,"state":true}` 曾被當成 `value:false` 並回 success
6.0.58	**WI-182** 只接官方雲端的設備**永遠無法還原設定** —— `downloadConfigFromServer` 只認地端 URL,沒設就靜默放棄。而 url 留空正是預設
6.0.62	WI-184 擋下「條件永遠不可能成立」的設定(TCP 通道來源只有布林語意,設 `>50` 永不觸發)
6.0.65	WI-188 改 token 名稱/權限被迫連帶換密碼 —— 畫面寫「留空＝不變」但兩端都沒實作

網頁 / 使用者體驗

版本	內容
6.0.47	WI-170/171 畫面操作流暢性:併發閘門放到唯一出口
6.0.52	**WI-177** 剛開頁面按導覽鈕沒反應(app.js 未載完就可以按);**WI-178** index.html 外置 QSPI,flash 97.1% → 94.0%
6.0.54	WI-185b index.html 也帶 i18n 短碼,納入同版指紋檢查(漏更會讓整頁標籤指到錯字串,**且無錯誤訊息**)
—	WI-189(純前端)頁面切到背景時連 telemetry 一起停。實測主迴圈卡頓 **−57%**

⚠ 已知特性(非缺陷,但要讓業主知道)

- **控制延遲**:典型 < 200 ms,但 p99 約 2~3 秒(實測極值 3.7 秒),每分鐘都會發生。成因是單核單迴圈 + 主迴圈裡的阻塞式 TLS。詳見 `docs/testing/qa-latency-characteristics.md`
- **還原舊快照會帶回當時的 cloudToken**:若那份快照的 token 為空,雲端備份會變成「推得出去、讀不回來」。畫面有明確提示,出廠測試 ST.15 / BK.3 會攔截

16

近期版本重點（v6.0.37 ~ v6.0.40 — 主迴圈卡死時，記下「卡在哪一段」）

> 🟡 **影響範圍:診斷能力為主;v6.0.38 另含一項 HTTP 解析的穩定性修正。**一般使用者不會看到行為差異,但設備若再發生 `UNEXPECTED` 重開,原因會直接寫在設備管理頁上。

問題:v6.0.36 把誤殺修掉之後,剩下的 `UNEXPECTED` 重開就是「真的卡死」了 —— 但**卡在哪裡看不到**。`[SLOW LOOP]` 這行是在一輪主迴圈**跑完**才印的,而真正把設備卡死的那一輪永遠跑不完, 所以最關鍵的那筆資料按定義不存在。實測 .46 兩次超過 30 秒的卡死,翻遍設備 log 看到的最大值只有 4.5 秒 —— 全是無關的雜訊。

版本	重點	關聯
v6.0.37	主迴圈每通過一個檢查點,就把段名寫進**重置後仍會保留的備份記憶體(BKPSRAM)**。看門狗重置後開機即可回讀,並由 `GET /api/diag` 帶出(`lastSeg`),遠端免接 USB 即可定位。	WI-163
	實戰命中:.46 在 v6.0.37 上跑滿 **23 小時 09 分**後發生 `UNEXPECTED` 重開,卡點成功存活過重置被讀出來 —— 這是這個問題第一次留下實體證據。	—
v6.0.38	**把檢查點切細**。原本名為 `http` 的那一段其實橫跨 1390 行(Modbus TCP、擴充模組 I2C、規則引擎、網頁伺服器、非同步測試全在裡面),回報「卡在 http」等於什麼都沒定位到。現已拆成 15 段:`led/net/io/mqtt/modbus/mqpub/scan/mbtcp/exp/rules/save/web/async/cs/ota`。	WI-163b
	順手補上一個**死角**:`checkOtaReboot` / OTA 下載 / `network.maintain()` 原本在所有檢查點之外,卡在那裡會無從得知,現歸入 `ota` 段。	—
	`lastSeg` 語意改為**「卡住的那一段」本身**(原本存的是「最後跑完」的段,要自己推下一段)。這個 off-by-one 極易誤讀 —— .46 首次回報 `lastSeg="modbus"` 第一眼就會被判成卡在 Modbus,實際卡在它後面那段。	—
	🔧 **穩定性修正**:網頁伺服器解析 `Content-Length` 標頭數值的內層迴圈**沒有時間上界** —— 條件被 `available()` 短路掉,而唯一的出口是讀到換行。若 socket 在狀態轉換的瞬間回報「有資料」但實際讀出 -1,這個迴圈會永遠轉下去、吃光記憶體,並讓主迴圈再也回不去餵狗。已改用與其餘標頭解析相同的硬性逾時。	—
v6.0.39	**卡點一併上雲**。設備 BKPSRAM 只留**最後一次**(開機即清空重新累計),而 `lastSeg` 先前只有 `GET /api/diag` 拿得到 —— 等於「沒有人在線上盯著的那幾次卡死」段名永久遺失,偏偏那正是最需要它的時候。心跳改帶 `lastSeg` / `lastSegUptime` → 雲端 `devices.meta`,遠端/離線也查得到。需搭配 config-server ≥ 本版(心跳欄位是明列白名單,舊版會直接丟棄)。	WI-163b
	⚠ 殘留限制:`devices.meta` 每次心跳覆蓋同一列,雲端只留**最新一次**。多次卡死只看得到最後一次;`bootCount` 單調遞增,故「卡了幾次」仍然準確。完整歷史需雲端另建事件表(未做)。	—
v6.0.40	**`exp` 段再切兩半**。實測 .46 的 `UNEXPECTED` 落在 `exp`,但該段混著兩件性質完全不同的事:`expupd`(`expansionManager.update()` → 熱插拔偵測,內含 `OptaController.cpp:630` 沒有時間上界的 `while (enter_while)`)與 `expio`(5 槽 I2C 讀取;唯一真走匯流排的是 `updateDigitalInputs()`,`digitalOutRead()` / `digitalRead()` 讀的是快取暫存器)。兩者修法完全不同,不切開分不出來。	WI-163c
	段序共 16 段:`led→net→io→mqtt→modbus→mqpub→scan→mbtcp→expupd→expio→rules→save→web→async→cs→ota`。	—

實機驗證(.250,`-D WEDGE_TEST` 測試韌體):在**網頁伺服器段內**故意讓主迴圈忙碌卡死 45 秒 ——

檢查項	預測	實測
重開原因	`UNEXPECTED`	`UNEXPECTED` ✅
卡點段名	`web` (wedge 端點所在段)	`lastSeg="web"` ✅
重置時機	30s 軟體期限 + 8s IWDG	觸發後約 37 秒 ✅
卡死當下 uptime	103s	`prevUptimeSec=103` ✅

`Content-Length` 修正的回歸測試:2/3/4 位數長度的 POST body 各送一次,回傳內容與送出完全一致;765,608 bytes 的 OTA(6 位數長度)亦正常完成。

17

近期版本重點（v6.0.36 — 看門狗誤殺:對外 TLS 連線一慢就自己重開）

> 🔴 **影響範圍:所有連雲端 / 連 MQTT broker 的設備**,不分 8310 / 8320、不分有線 / WiFi。這是業主長期回報「設備不定時自己重開、API 不穩定」的其中一個根因。

問題:硬體看門狗(IWDG)的期限是 **8 秒**,但一次對外 TLS 握手可以合法地花上 **10 秒以上**。`createWebClient()` 的 `setSocketTimeout(5000)` 只管「單次 socket 收送」,而 TLS 握手要好幾個 來回、每一段各自可以吃滿 5 秒。**逾時預算比看門狗還長**,所以只要雲端或 broker 剛好慢一下, 設備就會在一件完全正常的工作中途被自己的看門狗打死。

實測證據(v6.0.35,兩台設備一整夜):4 次 `UNEXPECTED` 重開,指紋完全一致 —— 全部發生在一次對外 TLS 連線開始後的 10~11 秒:

時間	設備	重開前最後一筆	間隔
01:32	.46 (WiFi)	`[CS-Cloud] Connecting to opta.smms.com.tw:443`	10s
02:21	.46 (WiFi)	`[CS-Cloud] Connecting to opta.smms.com.tw:443`	10s
03:41	.46 (WiFi)	`[CS-Cloud] Connecting to opta.smms.com.tw:443`	11s
04:46	.250 (有線)	`MQTT: Connecting to mosquitto.smms.com.tw` (TLS)	10s

設備並沒有當機 —— 它在做一件正常的事,只是做得比 8 秒久。

版本	重點	關聯
v6.0.36	**「卡多久算當機」由硬體 8s 改為軟體 30s**。主迴圈只蓋「我還活著」時戳,改由獨立高優先權執行緒依時戳決定餵不餵狗。保護沒有變弱,只是把誤判窗口拉開。	WI-162
	保護矩陣:整個 RTOS 死掉(含餵狗執行緒)→ 沒人餵 → IWDG **8s 內 reset(不變)**;主迴圈卡死但 RTOS 還活著 → 逾 30s 停餵 → **最晚 38s reset**(原 8s);合法的 10s TLS 握手 → **不再重開**(原必定重開)。	—
	實機主動驗證(.250):把設定伺服器指向一個「接受 TCP 但永不回應」的黑洞,誘發 `[SLOW LOOP] total=11144ms`(主迴圈停頓 11.1 秒)。`bootCount` **全程不變 238**,設備未重開。同一情境在 v6.0.35 是必定重置。	—

兩側都驗過(改的是安全機制本身,只驗「不再誤報」不夠 —— 若改壞了等於把看門狗拆掉, 而且症狀靜默:要等現場某台真的卡死、永遠不恢復才會有人發現):

測試	手法	結果
不再誤殺(false positive)	設定伺服器指向「接受 TCP 但永不回應」的黑洞,誘發 `[SLOW LOOP] total=11144ms`	`bootCount` 不變,**未重開** ✅
未達期限仍存活	`-D WEDGE_TEST` 測試韌體,主迴圈忙碌卡死 15 秒	`bootCount` 不變,uptime 連續 ✅
真當機仍抓得到 (true positive)	同上,卡死 **45 秒**	**約 37 秒被重置**,`bootCount` 239→240、`reason=UNEXPECTED`,與預測的 30s+8s=38s 吻合 ✅
新韌體仍可 OTA	跑在 6.0.36 上再發起 OTA	成功 ✅

回歸測試方式:`PLATFORMIO_BUILD_FLAGS="-D WEDGE_TEST" pio run -e opta`,燒錄後 `GET /api/debug/wedge?ms=45000`。該端點以 `#ifdef WEDGE_TEST` 保護,正式版編不進去。

代價:真正當機的自動復原從 8 秒變成最長 38 秒。換掉的是「每隔幾小時被誤殺一次」。更長時間的迴圈卡死另有 v5.9.250 的 gateway 探測自救(>3 分鐘自重開)接手。

現場自我檢查:設備管理頁看「重開原因」,若出現 `UNEXPECTED` 而設備當時明明在正常運作, 且重開時間點對得上雲端心跳或 MQTT 重連,即為此缺陷。升級至 v6.0.36 以上即解。

18

近期版本重點（v6.0.32 — WiFi 上線後未掛回 QSPI:開機卡死 / 網頁失效 / 授權無法還原）

🔴 **影響範圍:** `interfaceMode=AUTO` **、未插網線、靠 WiFi 上線的設備**(WIFI_ONLY 與純有線不受影響)。此缺陷自 WI-124/125 落實 QSPI 並存時就存在(當時只改了 WIFI_ONLY 路徑),於 v6.0.32 修正。⚠️ **v6.0.21 已因此自 stable 降回 dev**;stable 目前為 v6.0.9。

版本	重點	關聯
v6.0.32	**WiFi 連線成功後重新掛載 QSPI:** `NetworkStateManager` 在 AUTO 模式 Ethernet 失敗改走 WiFi 時會 `unmount()` QSPI,但**只有 WiFi 連線失敗才** `remount()` **,成功反而不掛回**→ 整個運行階段 QSPI 皆為 unmounted。	ADR-014 追記
	症狀一:開機永久卡死。`licenseManager.begin()` 停在 `License...` 不再前進,且卡點在 IWDG 啟用之前,看門狗救不回 —— 現場只能拆機 USB DFU。	—
	症狀二:網頁只剩裸 HTML。`/app.js`、`/styles.css`、語系檔全部 404(首頁由韌體 PROGMEM 提供,故「看得到畫面卻完全不能用」)。	—
	症狀三:授權無法由持久 token 還原,連帶統計/模板等 QSPI 檔案皆讀不到。	—
	實機驗證(.46): `qspiMounted` False→True、`/app.js` 404→200、開機由卡死→走完 `[IWDG] enabled`、授權由無法還原→ `ACTIVE`。	—

現場自我檢查:若設備網頁打開後版面全亂(無樣式)、或重開後停在啟動階段連不上, 且該設備是「WiFi 連線 + 未插網線」,即為此缺陷。升級至 v6.0.32 以上即解; 無法連線者需以 USB DFU 燒錄。

19

近期版本重點（v6.0.20 — 通知列改為文件流內，取代 v6.0.19 的浮動版）

v6.0.19 把右下角 toast 改成浮動的置頂通知列,但**實機量到它壓住頁首的 🔄 按鈕** —— 等於把 死區從右下角搬到上面。根因:容器可以 `pointer-events:none`,但卡片本身必須可互動(× 要能按), 所以只要是浮動定位就一定會蓋住底下的東西。**v6.0.19 請勿發佈給客戶,用本版取代。**

版本	重點	關聯
v6.0.20	**通知列改放 `<main>` 文件流內(`position:sticky`)**:通知佔自己的版面空間、把內容往下推,結構上不可能遮到任何控制項;`:empty{display:none}` 讓無通知時不留空白條;沒有 `<main>` 的頁面(AP 模式/開荒頁)退回浮動版。這才是 AWS flashbar 的實際做法。	業主回報
	.46 實機驗證:9 顆可見按鈕 0 遮擋(6 顆被往下推);通知在畫面上時點按鈕實際送出 5 個 API 請求;×關閉、同訊息合併 ×N、重開後自動重建皆正常。	—

20

近期版本重點（v6.0.19 — 設定持久化修正 + 網頁操作阻塞修正）

> 三個症狀來自業主端回報:「勾選存不住」「密碼送不出去」「通知還在就不能按」。前兩者是真 bug,第三者的真因與通知機制無關 — 是 CSS 命中測試。dev channel 先行。
>
> ⚠ 本表在 v6.0.12 之後、v6.0.19 之前(v6.0.13 ~ v6.0.18)尚未補記,待回填。

版本	重點	關聯
v6.0.19	**「變更時才發」勾選重開後消失**:`converterOnChange` 在 `LocalConfig` 的存檔與讀檔兩處都漏(v5.9.263 新增時遺漏)。POST 成功、GET 讀得到,但寫進 flash 的 JSON 沒這個 key → 下次自 flash 載入即回未勾選;症狀「時有時無」取決於中間有無經過重開/設定重載。`TcpIoChannel` 同層 22 欄位已逐一稽核,此為唯一缺漏。	業主回報
	Toast 永久攔截右下角點擊 → 改置頂通知列:舊 `.toast` 是 `position:fixed` + `z-index:99999` 卻無 `pointer-events:none`,且 `.show` 只切 `opacity` 不切 `display` → 元素永遠佔著右下角矩形吃掉點擊。**API 沒被卡住,是點擊沒落到按鈕上**。改 AWS flashbar:置頂堆疊、容器 `pointer-events:none`、多則並存、可 × 關、同訊息併 ×N、上限 4 則。	業主回報
	登入密碼送出顯示「網路錯誤」:驗證的第一發 `/api/system` 無任何重試 — 全站唯一裸 fetch 的關鍵路徑(`fetchWithRetry` / `apiSave` 皆為同一毛病而生,同函式第二發 `/api/tokens` 早有 3 次重試)。設備忙碌時 lwIP backlog 直接 reject → 一次失敗即判死。改 4 次重試+遞增退避+每次自帶 8s timeout;401 仍即判密碼錯誤不重試;加 in-flight 閘門與按鈕 disable(登入鈕與 Enter 同綁一函式,連按會開並發連線)。	業主回報
	登入文案自帶三語系:語系字典自 QSPI 載失敗時 `t()` 會把壓縮短碼(如 `c6`)直接顯示給使用者;登入是第一個畫面也最易撞上,故整個登入流程不再依賴外部字典。	業主回報

21

近期版本重點（v6.0.10 ~ v6.0.12 — MQTT 在線可觀測性:LWT + RS485 設備狀態事件 + UI 離線呈現）

> 📡 針對「客戶機台/控制器非 24h 在線(夜間斷電)」的觀測盲點。dev channel 先行,134 驗證後 promote stable。v6.0.11/12 為 134 dogfood 即時修正:

版本	重點	關聯
v6.0.12	**MQTT keepalive 10s(庫預設)→60s**:單執行緒 loop 卡頓(HTTP/掃描/TLS,3~~42s 常態)使 10s keepalive 頻繁被 broker 踢線 → 加上 LWT 後變 retained offline/online 拍動雜訊(134 實測 21~~75s 週期)。60s 容忍卡頓;真斷電偵測上限 ~90s。	WI-158
v6.0.11	**TCP IO 編輯「找不到通道」誤報修正**:`fetchWithRetry` 從不 throw — 重試耗盡/401 一律 resolve `null`,v6.0.10 誤流入「找不到此通道資料」→ 現以 null 判「載入失敗(設備忙碌)」+重試鈕(134 Playwright 實證誤導)。	WI-158
	狀態列排版:插入點從 `.modal-h`(flex 標題列,被擠成直排)改 `.modal-b` 頂部;清除時同步清 textContent(避免讀殘字誤判)。	WI-158
	resync 收斂:connect-success 不再重複設旗標,統一由 reconnect-edge 觸發。	WI-158

版本	重點	關聯
v6.0.10	**MQTT LWT(遺囑)**:`connect()` 前 `setWill({prefix}/status, "offline", retained)` — 控制器斷電/斷網(不經 disconnect)時 broker 於 keepalive 逾時自動翻 retained `offline`。舊行為:retained `online` 在設備斷電後**整晚誤導 MES**。	WI-158
	RS485 設備斷線/回線事件:狀態轉變 → retained `{prefix}/rs485/slave/{slaveId}/status`=online/offline;PowerMonitor 集中偵測轉變(涵蓋全機型 flip 點)入佇列,main loop 發佈(每 loop ≤4 筆);MQTT (re)connect 重發全部現況。**掛 slaveId 穩定身分**,與既有 `rs485/{deviceIdx}/{field}`(WI-074,索引會因刪機台位移)分樹。	WI-158
	UI 離線呈現:離線設備數值變暗(明示舊值)+ 徽章「離線・最後回應 HH:MM」(琥珀,沿 v6.0.6 過時慣例);`lastUpdate==0` 顯「從未連線」;回線復原。	WI-158

22

近期版本重點（v6.0.9 — 出廠預設 9600 + 「載入失敗仍可儲存」防呆 + 掃描涵蓋 4800/2400）

> 🚨 源於現場事故:cfg 載入失敗時通訊埠 modal 以預設值 19200 渲染、且變更確認被繞過,按儲存把 19200 靜默寫進設備 → RS485 全離線。本版根治整族「載入失敗 → 顯示預設值 → 儲存 = 靜默改設定」。

版本	重點	關聯
v6.0.9	**RS485 出廠預設 19200→9600**:現場電表以 9600 為大宗;`initDefaultConfig`、載入缺鍵 fallback、無 config 掃描起點三處對齊。	WI-157
	通訊埠設定 modal 防呆(根治幽靈存檔):cfg 未載入 → 顯示「載入中/重試」,**絕不以預設值渲染表單**;`saveBusSettings` / `_forceSaveBus` 雙守衛,未載入一律拒存(舊行為:oldB0 與表單同時 fallback 19200 → hasChanged=false → **跳過確認直接寫入**)。	WI-157
	TCP IO 編輯 modal 防呆:`/api/tcpio` 載入失敗不再靜默留全空白(補 `.catch`)→ modal 頂部狀態列 +「🔄 重新載入」鈕;通道資料未回填完成前 `saveTcpIo` 拒存(防以空白覆蓋原通道)。	WI-157
	掃描涵蓋 4800/2400:`scanBauds` 加入兩檔(電表常見出廠值,先前永遠掃不到);組合 15→21,掃描上限 ~~23s~~→54s;`params[]` 擴 24 防溢位、組合迴圈改 sizeof 推導;`rs485Buses` baud 白名單同步加 2400/4800(**UI 選 4800/2400 需韌體 ≥6.0.9**)。	WI-157
	rs485Buses 解析陣列界限修正:以 `]` 為界,防陣列物件數 < 2 時掃進後續 JSON 物件(潛伏 bug)。	WI-157
	baud 變更審計 log:`Updated rs485Buses[i]: baud 9600->19200` 帶 old→new(事後可從 /api/log 追認誰動過)。	WI-157

23

近期版本重點（v6.0.8 — SF965 原生解碼 + 移除/新增暫存器保證）

> 🔢 徹底解決 SF965 計數值的小數/整數與設備增刪串台問題。與 6.0.x 向後相容。

版本	重點	關聯
v6.0.8	**SF965 原生解碼(新 dataType `VAR_SF965_PV` =8)**:SF965 PV(addr 4)為特規 32-bit —— **高 3 byte=計數值、低 byte=小數點位數(dot)** → 值 = `(raw>>8) / 10^dot`。原生正確:dot=0 即乾淨整數、dot≠0 自動小數,**不再靠 `÷256` scale 湊**(舊法只在 scale 精確=1/256 且 dot=0 才對)。SF965 型號預設暫存器改用此型別 → **選型號、套預設即正確**(顯示 + 發佈一致)。	晉榮 SF965 解碼實證
	移除一台→再新增,暫存器不串台(保證):暫存器以 **slaveId 綁定**(6.0.7 起),刪設備觸發陣列重編號時,其他設備暫存器不受影響;再新增同 slaveId 的設備會**自動掛回**其暫存器。46 實測:刪 slave50 → slave51 暫存器完好 → re-add slave50 自動掛回。	WI-153

24

近期版本重點（v6.0.7 — RS485 離線行為改善）

> 🔌 針對「工廠下班/午休關機 → RS485 slave 離線」的兩個困擾。與 6.0.x 向後相容。

版本	重點	關聯
v6.0.7	**離線來源不發假 0**:converter 模板引用到**離線** RS485 設備時,整筆**跳過發佈**(不送 `{"counter":0}` 讓 MES 誤判為「計數歸零」)。機台關機/斷線期間保持沉默,回線自動恢復。實作:`ConverterEngine` 填充時若來源 `isOnlineByIndex==false` 設旗標 → `pushTemplate` 跳過。	—
	離線退避輪詢 10s→30s:`OFFLINE_INTERVAL` 拉長,關機期間離線 slave 少拖累單執行緒 loop(每台 poll 逐次 timeout),網頁較不卡;機台回線 30s 內偵測恢復。	—
	前提:SF965 計數器**斷電保留計數**(業主確認)→ 關機一晚不丟每日累計,回線續讀。	—

25

近期版本重點（v6.0.6 — 畫面更新健壯化 + 模板 :N 說明）

> 🩺 針對「Opta API 偶爾失敗導致畫面難判斷狀況」。與 6.0.x 向後相容(app.js + index.html,無新韌體邏輯)。

版本	重點	關聯
v6.0.6	**失敗保留舊值(根)**:`config` / `signals` / `actions` / `rules` 抓取失敗時不再 `|| {}` / `|| []` 清空畫面,改保留上次好資料 →「空」不再偽裝成「失敗」。根治「總覽 RS485 區塊過一分鐘消失」類。輪詢 io/power 本就只在成功時更新。	—
	每頁針對性重整:標題旁「🔄 重新整理此頁」只重抓當前頁資料(非整頁 reload)—— 單執行緒設備一次少打幾個請求,較不易失敗。	—
	資料新鮮度顯示:標題旁「更新於 N 秒前」,超過 2× 輪詢間隔變**琥珀=可能過時**;成功取得資料即回綠。讓使用者一眼知道畫面是否即時。	—
	模板 `:N` 說明上線:JSON 模板欄提示 + 可用變數參考範例(`${rs485.1.計數值:0}`=整數);文件(spec §3.4 / 操作手冊變數表)同步。	—

26

近期版本重點（v6.0.5 — Converter 存檔穩定性 + 模板小數位數修飾詞）

> 🔧 修 Converter 模板存檔失敗、總覽 RS485 區塊消失;新增模板變數小數位數控制。與 6.0.x 向後相容。

版本	重點	關聯
v6.0.5	**Converter 模板存檔失敗修正**:存完 `/api/tcpio` 會觸發 MQTT 重訂閱(遠端 broker TLS 握手阻塞單執行緒 loop 數秒),緊接的模板 POST 撞死窗 →「Failed to fetch」。前端在兩 POST 間插入 `_waitDeviceReady`(輪詢 `/api/system` 到連兩次 OK)才送模板。	—
	apiSave 檢查 `success`:韌體存檔 handler 恆回 HTTP 200,成敗在 body `success`。前端改為 `success:false` 視為可重試錯誤並報錯,杜絕「說成功其實沒存」的靜默失敗。	—
	總覽 RS485 區塊消失修正:`refreshOverview` 週期重載 config,fetch 失敗時原本 `|| {}` 把快取清空 → RS485 卡片在忙碌設備上「過一分鐘消失」。改 `|| cfg`(失敗保留舊值,對齊其他 loader)。	—
	模板變數小數位數修飾詞:`${rs485.x.field:N}` — `:0`=整數(計數器)、`:N`=N 位小數、無修飾詞=預設 2 位(相容既有模板)。由模板作者逐次控制發佈格式;通用的「每暫存器小數位數」設定為後續工作。	—

27

近期版本重點（v6.0.4 — MQTT 連線測試 TLS 修正 + 小 UI）

> 🔧 修「設定頁 MQTT 測試連線在 TLS 時必失敗(直接存檔卻成功)」。與 6.0.x 向後相容。

版本	重點	關聯
v6.0.4	**MQTT 測試改 deferred + 現有連線短路**:(1) 測試排程到主 loop 執行(POST 回 202 → GET `/api/mqtt/test` 輪詢,前端顯示「測試中 n/30」),不在 inbound handler 內開 TLS; (2) **真兇**=設備單執行緒 + 有限 mbedTLS,對「已連線的同一 broker」再開第二條 TLS 必握手失敗 → 若主 MQTT 已連到相同 broker:port+帳號+用已存密碼,直接以**現有連線**判定成功(既有連線本身即證明);任一參數改動則落到真實連線測試。	[[device_outbound_tls_in_handler_fails]]
	測試參數補洞:測試現在帶 `useTls/tlsInsecure`(以前只送 plain 參數);**密碼欄留空 = 用已儲存密碼**(對齊存檔語意,以前拿空密碼認證必失敗);補 Authorization header。	—
	小 UI:設定頁「儲存名稱」與「📟 設備日誌」改同一列並排(原各佔一行)。	—

28

近期版本重點（v6.0.2 — RS485 同型號多台防串台）

> 🔧 修 WI-153 crosstalk 家族「重開機後復發」的根因。與 6.0.x 向後相容。

版本	重點	關聯
v6.0.2	**register↔device 綁定 slaveId 持久化**:`ModbusRegister.slaveId` 執行期有值(存檔當下正常)但**沒存進 flash** → 重開機後全歸 0 → 退回 `deviceIndex`(陣列位置)比對 → 同型號多台(如兩台 SF965)在刪台/重排後重開機會串台。存/讀各補一行(`src/LocalConfig.h`),缺鍵給 0 向後相容。	WI-153
	正確用法:同型號多台各設**不同 slaveId(≤32)**;存檔即以 slaveId 穩定綁定,跨重開機不退化、不串台。	—

29

近期版本重點（v6.0.1 — 遠端診斷：網頁 Serial Log + 開機原因）

> 🩺 **遠端維運**：針對「設備只能經 VPN 進入、現場無法插 USB 看 serial」的痛點。與 6.0.0 向後相容。

版本	重點	關聯
v6.0.1	**網頁即時 Serial Log**(Tier 1):`GET /api/log`(PERM_ADMIN)把韌體 8KB RAM ring buffer 吐成純文字;設定頁「📟 設備日誌」按鈕或直接開 `/log.html`(走 QSPI 熱更)即可遠端看即時 log(RS485 輪詢 / MQTT / 開機序列)。	[[opta_default_admin_token]]
	開機診斷(Tier 3):`GET /api/diag` + 心跳帶 `bootReason/bootDetail/prevUptime/bootCount` → 雲端 `devices.meta`。設備隨後失聯,雲端仍看得到「上次為何重開 / 撐多久 / 累積開機次數」。	—
	為何重開的判定:此 Opta(MCUboot)在 app 啟動前已清 RCC->RSR(實測 rawRsr 恆 0),硬體旗標不可用;改用 **BKPSRAM「有意重開」標記** —— 有標記 = `SOFTWARE`(附細節 ota / loop-wedge / mac-revert / api-reboot / net-failover / …),無標記 = `UNEXPECTED`(看門狗 IWDG / 斷電 / HardFault = 當機、重開迴圈的關鍵訊號)。	—
	安全:`/api/log`、`/api/diag` 需 PERM_ADMIN;修 3 處明文外洩(MQTT 密碼 POST body/mqttJson 改 `redactSecrets` 遮罩、license 回應改印長度、AP 密碼印 `****`)—— 避免經 log ring 由網頁外洩。	[[webpage_blank_terser_fallback_footgun]]

30

近期版本重點（v6.0.0 — WI-155 多專案 × 多環境平台化）

> 🚩 **v6.0.0 里程碑**：自 5.9.285 起跳大版號,整合 WI-155 全系列(對外 MQTT 命名空間 + MAC 身分)。與 5.9.x **向後相容**(未設命名空間即沿用 legacy `mes/gateway/{uid}`)。權威 per-version changelog 仍在 cloud manifest。

版本	重點	卡片 / 記憶
v6.0.0	**對外 MQTT topic 命名空間**:可設 `{projectname}-{env}/gateway/{mac12hex}`(全小寫;設定頁填專案名稱+環境),未設則 legacy `mes/gateway/{uid}`,向後相容。	WI-155、[[mqtt_protocol_revamp_wi155]]
	MAC 對外身分:心跳 / whoami / info 帶 `mac`;雲端建 **MAC↔UID 查表**,relay 訂 `+/gateway/#` 並把命名空間 topic 改寫成 canonical(內部仍以 UID 為鍵,不破 WI-147)。	WI-155
	MAC 撞號自動解決(AC4):後到者自動降級回 legacy topic(不動硬體 MAC、不誤路由)+ 撞號排除後自癒;`GET /api/devices/mac-collisions`。	WI-155
	MAC 位址軟體覆寫(設定頁進階):指定軟體 MAC(不碰硬體 OTP)+ 即時驗證 + **試用期 90s 無 IP 自動回退**(防失聯);複製鈕給對外格式(無冒號小寫)。	WI-155、[[mac_change_arp_router_interference]]
	payload `{"value":N}` 雙支援(cmd/signal,相容裸數字);新欄位 **i18n** 三語系;修 MQTT 設定頁**留空清密碼** **footgun**;build_webpage **node --check** 後盾。	WI-155

31

近期版本重點（v5.9.241 ~ v5.9.244 — 8310 機型支援）

> ℹ️ 權威 changelog 仍在 cloud manifest（單一來源，見上）。本段為**文件側快速索引**，方便對照本批 8310 相關改動，對應 sprint cards WI-149 ~ WI-152。

版本	重點	卡片 / 記憶
v5.9.241	**8310 安裝根治**：QSPI 空白 MBR 回 `-3101`，舊韌體只救 `-3102` → FS 不掛載 → 設定/網路存不了。`src/LocalConfig.h` 救援涵蓋 `-3101 + -3102`，任何空白 QSPI 設備第一次燒就自我建分割區 + 格式化（相容 8320，init 成功就跳過）。	WI-149 前置、[[opta_8310_qspi_blank_mbr_3101]]
v5.9.242 ~ .244	**拓荒包（Pioneer）韌體變體 + 開荒設定頁**（`env:opta_pioneer`，WI-149）：自包單一網頁（UID + 網路 + 大「上傳 .mesb」鈕），USB 首燒即可用。	WI-149
	型號感知 UI 過濾（WI-151）：OTP 讀板型 → `g_hasWifi`，曝光 `/api/system`、`/api/license` 的 `hasWifi` / `boardModel`；8310 拓荒頁與完整版網路設定過濾掉 WiFi（隱藏 SSID/密碼 + 介面模式選單，強制乙太網路）。	WI-151、[[opta_otp_board_functionalities]]
	8310 心跳燈（WI-152）：藍燈是 BLE 模組燈（僅 WiFi 版能亮），8310 改用**琥珀色（紅+綠同亮）**心跳，8320 維持藍燈。	WI-152
	.mesb 一鍵完整安裝 + favicon + ETag 修正：`.mesb` 打包韌體 + 6 資產 + `favicon.ico`；拓荒頁內嵌真 favicon；根網頁 ETag 併入頁面大小，避免 pioneer→完整版（同版號）OTA 後顯示舊快取頁。	WI-149

> **注意**：AP 模式需要 WiFi，只有 8320 能進；8310 無 WiFi 進不了 AP（會 fallback）。User button 開機長按：0-5s 正常 / 5-10s AP / 10s+ 原廠重置。

32

相關程式（原始碼,非本站文件）

- `scripts/release_firmware.py` — 打包腳本（含 `resolve_changelog`）
- `config-server/modules/firmware.js` — Cloud upload endpoint
- `config-server/public/index.html` — Admin UI 韌體列表

04 開發者 / 整合

控制延遲特性

> **版本**:1.0 | **日期**:2026-08-20 | **韌體**:6.0.65 | **量測機**:OPTA-GW-001(8320 / WiFi)

01

為什麼要有這份文件

業主觀察到「燈號會停滯 1~3 秒」。追查後確認**這不是顯示問題,是控制迴圈的延遲**, 所以有必要把它量化成規格,而不是用「應該很快」帶過。

02

架構事實(先釐清一個常見誤解)

這個韌體跑在單一核心(M7)、單一主迴圈,不是「控制一顆核心、網頁另一顆」。每輪迴圈依序跑完:

```
led → net → io → mqtt → modbus → mqpub → scan → mbtcp
    → expupd → expio → rules → save → web → async → cs → ota
```

- **LED 在 `led` 段更新,一輪只更新一次**
- **規則評估、DO 輸出也在同一輪**

所以任何一段卡住,燈與控制**同時**被延後。燈號停滯是徵兆,不是獨立問題。

(硬體上確實有第二顆 M4 核心,第二個 MB 的 flash 也保留給它,但本韌體未使用; 改用它屬重新設計等級,且會動到 flash layout 使既有設備的 OTA 失效。)

03

實測數據

序列只記錄**超過 200ms** 的迴圈,低於此值不列入(那是正常情況)。

主迴圈週期(= 控制延遲的上界)

指標	值
超過 200ms 的迴圈	約每分鐘 4~5 次
p50(超標者)	668 ms
p90	2088 ms
p99	**2879 ms**
實測極值	**3711 ms**

結論:DI 觸發到 DO 動作,典型 < 200 ms,最差 2~3 秒(極值 3.7 秒)。這不是罕見事件 —— 每分鐘都在發生。

卡頓發生在哪一段

段	說明
`net`	最大宗。WiFi 驅動 + TLS 加密收送
`web`	服務網頁請求(有人看畫面時才有)
`modbus`	RS485 輪詢(每 2 秒一次,佔比小)

儀表板的代價(WI-189 前後對照)

同一台設備,期間完全不碰:

	有儀表板	無儀表板	變化
卡頓次數/分鐘	7.0	3.0	**−57%**
`web` 卡頓/分鐘	2.31 s	0.61 s	**−73%**
`net` 卡頓/分鐘	3.07 s	1.87 s	−39%
最久單次	2879 ms	2092 ms	

WI-189 已讓頁面切到背景時停止 telemetry(韌體本就是 keepalive 制,不續命就自停), 所以**現在只有「真的在看畫面」時才付這個代價**。

04

根本原因

主迴圈裡有**阻塞式網路操作**,而 TLS 沒有有效的時間上界: `setSocketTimeout` 只管單次 socket 收送,一次握手是多個來回,總時長無界 (WI-183 的黑洞實驗證實 `connect()` 可卡 5 分鐘不返回)。

分執行緒解不了 —— `csThread` 已是獨立執行緒,WI-175 事故正是它在做 TLS 時 把主迴圈一起卡住,因為兩者搶同一個 WiFi 驅動鎖。

05

給業主的使用建議

1. **不看畫面時把頁面關掉或切到背景** —— 卡頓次數減半(WI-189 自動處理)
2. **儀表板更新頻率**:畫面上的「快速(2s)/標準(3s)/省電(5s)/慢速(10s)/暫停」直接影響設備負載
3. **應用適配**:
   - 狀態指示、三色燈、非即時回報 → 目前特性可接受
   - 計數、安全連鎖、需要確定性時序 → **需先確認 2~3 秒的最差延遲是否可接受**

06

後續改善方向(交付後)

依數據,改善順序應為:

1. 讓網路操作有真正的時間上界(非阻塞 socket + 狀態機)
2. 控制路徑與網路路徑不共用驅動鎖 —— 這才是分執行緒有意義的前提
3. (大改)啟用 M4 核心做控制/網路分離

不建議把 LED 搬到獨立執行緒當解法:板上 L1-L4 是鏡射 DO 狀態, DO 本身被延遲時 LED 只會忠實顯示延遲後的狀態,真正變順的只有心跳燈 —— 那是把溫度計拆掉,不是退燒。

07

文件修訂紀錄

日期	版本	變更
2026-08-20	1.0	首版。POC 實測 + WI-189 前後對照

04 開發者 / 整合

現場驗收流程

版本:2.7 | **日期**:2026-08-28 | **韌體基線**:6.0.83 **對象**:安裝工程師、業主見證人 **相關**:出廠測試規範、控制延遲特性

從一台空機開始,到設備上線送出第一筆資料。每一步都寫明**要做什麼、看到什麼才算通過、沒通過時怎麼辦**,由業主全程見證、逐項打勾。

⚠️ 本文件中的帳號、密碼、IP 一律以 `<...>` 佔位符表示,實際值請向專案負責人索取,不要寫進公開文件或版本控制。

01

給人工驗收者:今天從這裡開始

這一節是**人工驗收的作業指引**。底下 1500 行是完整規範(遇到問題回頭查),但你**實際要動手的只有兩件事**:先讓機器自己證明一遍,再親自走機器碰不到的部分。

第 0 步 — 先跑自動驗收,綠了才開始(約 13 分鐘)

```bash
python3 scripts/acceptance_run.py --dut http://<設備IP> --token <admin token>
```

看到這行才往下走:

```
結果：28 PASS · 0 FAIL · 0 SKIP
✅ 驗收流程一口氣跑完，無中斷
```

為什麼先跑它：它把本文件所有能自動化的測項(階段 0 / 2B.2 / 3 / 4 / 5 / 6、情境 M、情境 B、情境 D、8.1)串成一次不中斷的執行,測完自清。先確定這些是綠的,**你人工驗的才是機器本身,不是在幫程式抓 bug**。任一項紅燈就先停下處理,不要帶著紅燈進人工階段。

第 1 步 — 你親自走這 7 項

自動化**本質上做不到**的部分。逐項打勾,不通過就寫在備註欄。

#	你要做的動作	通過標準	詳見	桌上這台
①	**供電與接地目視** —— 看電源極性、接地線、端子有無鬆脫	接線牢固、有接地、無裸露	階段 0	可做
②	**歸零成出廠狀態** —— 破壞性,會清掉設定與授權憑證	清空後能重新設定	階段 R	⚠️ **先別做**(會清掉現有設定)
③	**空機燒拓荒包** —— 需要一台全新未設定的機器	燒錄成功、能連上網頁	階段 1	❌ 無空機
④	**實體撥開關** —— 手撥 DI1 的 Toggle 開關	撥上→輸出動作;撥回→復歸。**連續 3 次一致**	情境 A・A1	✅ **要做**
⑤	**計數器現場觸發** —— 實體觸發計數輸入	計數值遞增	階段 5.4	❌ 本站無計數器
⑥	**實體斷電復原** —— 真的拔電源(見下方合併程序)	自行開機、設定一項不少、MQTT 自己接回	階段 7	✅ **要做**
⑦	**拍照 + 雙方簽名**	照片歸檔、逐項核對簽名	階段 8	✅ 要做

> ⑤ 本站沒有計數器硬體,不是跳過不驗,是**沒有這個受測對象**。驗收單上請註明「本站無此配置」,不要留空白讓人以為漏測。

> 📌 **改通道名稱時,畫面上方會跳一個藍色徽章「💾 儲存中 N」** —— 那是還沒送完的筆數。**等數字歸零再離開該頁**,否則最後幾筆可能還在路上。(v6.0.78 之前它會顯示成「💾 bH N」,`bH` 是漏掉的語系鍵,已修 —— WI-207。)

⑥ 的建議做法：一次操作驗三件事

階段 7 拔電源之前,**順手把網路故障切換一起驗掉**(WI-197 的修正尚未經實機確認)。先開一個終端機掛著看：

```bash
watch -n 5 'curl -s $DUT/api/diag | python3 -m json.tool | grep -E "resetReason|
```

步驟	動作	應該看到
1	**拔掉網路線**	設備切到 WiFi,`bootDetail` = `net-failover`
2	等它在 WiFi 上穩定,網頁能開	WiFi IP 可連
3	**插回網路線**	自動切回有線,`bootDetail` = **`net-recovery`** ← 這是本次要驗的修正
4	**拔掉電源**,等 10 秒,插回	自行開機回到原 IP,`bootCount` +1,`resetReason` = `UNEXPECTED`
5	逐項清點設定(階段 7.2)	**全部與斷電前一致**,無一走失

> 步驟 4 的 `resetReason=UNEXPECTED` ／ `bootDetail` 空 **是正確的**,不是異常 —— 硬體斷電不會留下重開原因的字條(BKPSRAM intent marker 的預期行為), 這正好和軟體重開的 `SOFTWARE/api-reboot` 分得出來。
>
> ⚠️ **長按 reset 不等於斷電**。2026-08-26 那次是以長按 reset 完成的, 乙太 PHY／WiFi 模組沒有真正斷電,**冷啟動路徑仍未涵蓋** —— 這台家族史上有冷開機競爭問題,所以這次請**真的拔電源**。

驗完之後

全部打勾 → 階段 8 結案。有任何一項沒過 → 寫進驗收單的未通過欄,並取得業主書面同意,或**不予交付**。

02

開始前：環境準備與連線設定

驗收過程會用到命令列工具,**先裝好再到現場**,不要在業主面前才發現少東西。

macOS / Linux | Windows

```
# Node（跑自動化測試）
brew install node

# 測試套件與瀏覽器核心
cd <專案>/web
npm install
npx playwright install chromium

# MQTT 命令列工具（發布 / 訂閱，驗證資料真的送出去）
brew install mosquitto
```

設備的 IP 與 UID 要等裝好之後才會知道（階段 3.2 會拿到）。現在先把檔案建好、把已知的填上；`DUT` 和 `UID_DEV` 留空,裝好設備再回來補。

每次開新終端機都要載入這些設定（後面所有指令都靠它們）：

📦 **現成檔案下載**：mqenv.zip

這是一個加密壓縮檔，僅供內部人員使用。解壓密碼不公開於本文件，請向專案負責人索取。裡面是已經填好本站實際連線參數的 `mqenv.sh`，解開後放到家目錄即可直接 `source ~/mqenv.sh` 使用，省去手動填值。

外部讀者請依下方範本自行建立，把 `<...>` 換成你自己環境的值。

macOS / Linux 存成 `~/mqenv.sh`，每開終端機 `source ~/mqenv.sh`；**Windows** 存成 `mqenv.ps1`，每開 PowerShell 執行 `. .\mqenv.ps1`（前面有一個點加空白）：

macOS / Linux | Windows

```
# ── 設備 ──
export DUT=http://<設備IP>
export TOK=<設備管理token>
export UID_DEV=<設備UID，見設定頁>
export PREFIX=mes/gateway/$UID_DEV

# ── 雲端（版本歷史 / 備份驗收要用，與設備 token 不同）──
export CLOUD=https://<雲端網址>
export CLOUD_TOK=<雲端token>

# ── MQTT ──
mq_sub(){ mosquitto_sub -h <broker主機> -p 8883 -u <帳號> -P <密碼> --insecure "$@"
mq_pub(){ mosquitto_pub -h <broker主機> -p 8883 -u <帳號> -P <密碼> --insecure "$@"
```

⚠ **兩個實測踩過的坑，都會給出誤導性的錯誤訊息：**

① 用 `--insecure`，不要去指定憑證檔。本系統的 MQTT **走 TLS 但不驗憑證**（設備出廠預設 `tlsInsecure=true`，業主指定），所以命令列也該用 `--insecure` —— 行為與設備一致，而且**不依賴任何檔案路徑**，換電腦一樣能跑。

反例（實測踩過）：`--capath /etc/ssl/certs` 會回 `Error: Protocol error`。那個目錄在 macOS 上**確實存在**，所以肉眼看不出是參數錯，而錯誤訊息會把人導向「broker 或網路壞掉」的方向。改用 `--cafile /etc/ssl/cert.pem` 雖然能動，但那條路徑不保證每台電腦都有 —— 依賴它只是把問題延後。

② 用函式包，不要用 `export MQ="-h ... -u ..."` 這種「把一串參數塞進變數」的寫法。macOS 預設是 zsh，而 **zsh 不會對未加引號的變數做詞彙拆分** —— 整串會被當成單一參數，報 `Unknown option '-h ...'`。同樣的寫法在 bash 可以、zsh 不行。函式兩種 shell 都能用。

`PREFIX` 不要用手拼。若本站啟用了 MQTT 命名空間（`projectName` / `env`），實際前綴就不是 `mes/gateway/<UID>`。設備裝好後用這條確認真正的前綴：

```bash
curl -s -H "Authorization: Bearer $TOK" $DUT/api/config | python3 -c "
import sys,json
d=json.load(sys.stdin); m=d.get('mqtt') or {}
proj,env=m.get('projectName') or '',m.get('env') or ''
print('  UID         :',d.get('uid'))
print('  projectName:',proj or '(空)')
print('  env         :',env or '(空)')
print('  → 實際前綴 :', f'{proj}-{env}/gateway/'+(d.get('mac') or '').replace(':',
"
```

兩者留空 = legacy 模式，前綴才是 `mes/gateway/<UID>`。**前綴錯了的症狀是「訂閱端一筆都收不到」**，而設備那邊看起來一切正常 —— 很難查。

確認連得到再往下：

macOS / Linux | Windows

```
curl -s -H "Authorization: Bearer $TOK" $DUT/api/system | python3 -m json.tool |
```

> **通過**：看得到 `version` / `boardModel`，且 `version` 與交付文件一致。

03

開始前：本站硬體表

驗收的每一個數字都要對得上實際接的東西，否則後面每一步都在猜。

編號	設備	接法	位址 / 通道	正常範圍
1	計數器 SF965（HB965SF）	RS485	Slave ID ____	計數值遞增
2	電流環 #1	4–20mA → AI 或 RS485	____	____ A
3	電流環 #2	4–20mA → AI 或 RS485	____	____ A
4	電流計（Finder）	RS485	Slave ID ____	____ A

電流環有兩種接法，驗收步驟不同，先確認實際是哪一種：

1. **帶 RS485 的電流計**（如 Finder 6M）→ 走**階段 5**，設 Slave ID 與型號模板。
2. **4–20mA 類比訊號**直接進類比輸入 → **本版尚未涵蓋**。若本站確實是這種接法，先與專案負責人確認量程換算方式，不要照 RS485 的步驟做。

04

階段 R：歸零 —— 把設備恢復成出廠狀態

> 🟢 全新未開封的設備：**整個階段 R 跳過，直接到「這台機器走哪一條路」**
>
> 全新機本來就是空的，沒有東西可以清。做歸零不會出錯，但是白費五分鐘。

以下只適用於「動過的機器」 —— 展示機、測試機、退回重配的機器。這類機器**必須先歸零**，否則舊設定會混進新現場，日後查問題時分不清哪些是這次設的。

先理解：設備上有兩塊儲存區

區域	內容	重燒韌體會清掉嗎
韌體區	程式本身	✅ 會
設定區	所有設定、網路帳密、**授權憑證**	❌ **不會**

> 「我重燒過韌體了，應該乾淨了吧」是**錯的** —— 設定原封不動還在。

三個層級

層級	怎麼做	清掉什麼	什麼時候用
L1	按住機身按鈕開機，長按 10 秒以上	只清設定檔，**不清授權憑證**	自己排錯用
L2	設定頁按「恢復原廠」	格式化整個設定區：設定 + 網路帳密 + **授權憑證全清**	**交機／轉售標準做法**
L3	USB 重燒韌體，**再做一次 L2**	韌體 + 設定區都換新	換版本，或狀態不明

⚠️ **L1 是交機時最容易踩的坑。**它看起來設定都清光了，但**授權憑證還留著** —— 設備重開後會自己把授權狀態還原回去，下一手會拿到一台帶著上一手憑證的機器。**交機一律用 L2。**

驗收標準

R.1 確認機器來歷

macOS / Linux | Windows

```
# 先看它身上有沒有東西 —— 有設定就是動過的
curl -s -H "Authorization: Bearer $TOK" $DUT/api/config | python3 -c "
import sys,json
d=json.load(sys.stdin)
print('  設備名稱  :',d.get('deviceName'))
print('  RS485 設備:',len([x for x in (d.get('modbusDevices') or []) if x.get('na
print('  Wi-Fi SSID:',(d.get('wifi') or {}).get('ssid') or '(空)')
"
```

通過：全新（全部為空）→ 跳過本階段；有任何殘留 → 往下做 R.2。**沒過**：連不上、讀不到 → **來歷不明一律當成動過**，執行 L3。

R.2 歸零前備份（如果這台上面有還需要的東西）

macOS / Linux | Windows

```
# ① 叫設備推上去
curl -s -X POST -H "Authorization: Bearer $TOK" $DUT/api/configserver/push
sleep 10

# ② 確認設備這一側沒問題(cloudToken 空白是最常見的失敗)
curl -s -H "Authorization: Bearer $TOK" $DUT/api/configserver | python3 -c "
import sys,json
d=json.load(sys.stdin); c=(d.get('push') or {}).get('cloud') or {}
print('  雲端位址   :',d.get('cloudUrl') or '(空)')
print('  Cloud Token:','(空!)' if not d.get('cloudToken') else '已填')
print('  上次推送   :',c.get('result'),'HTTP',c.get('httpCode'))
"

# ③ 真正的驗收點:跟【雲端】要快照清單,看得到剛推的那一筆
curl -s -H "Authorization: Bearer $CLOUD_TOK" \
  "$CLOUD/api/devices/$UID_DEV/snapshots?limit=5" | python3 -c "
import sys,json
s=json.load(sys.stdin)
print('  快照筆數:',len(s))
for x in s[:5]: print('   -',x.get('filename'))
"
```

通過:② 的「上次推送」是 `OK` / HTTP `200`,**而且** ③ 列得出快照,檔名長得像 `snapshot.<一長串數字>.v5.json`。**沒過**:確定不需要保留就直接往下。**歸零之後就真的沒了**,這一步不要含糊。

⚠️ **③ 一定要打雲端,不能打設備。**設備端所有 `/api/configserver...` 開頭的 GET —— 含 `/versions`、`/history` —— **回的都是同一包設定,永遠沒有清單**。拿它來驗會誤以為「查過了」,但你只驗到「推得上去」,沒驗到「讀得回來」,而這一段的全部意義就在後者。

R.3 執行 L2「恢復原廠」

macOS / Linux | Windows

```
curl -s -X POST -H "Authorization: Bearer $TOK" $DUT/api/system/factory-reset
```

也可以在設定頁按「恢復原廠」按鈕，效果相同。⚠️ **這會格式化整個設定區**（設定 + 網路帳密 + 授權憑證全清），且**無法復原**。執行前確認 R.2 已完成。

🛑 歸零之後，雲端還原「不會自己成功」—— 必須先做這一步

恢復原廠**連 Cloud Token 一起清掉**。而設備去雲端抓設定時，token 是空的就 **不會送出授權標頭**，雲端直接回 `401`。結果是：

- 雲端後台顯示**還原成功**、「已排入佇列」
- 指令確實送到設備、也被消耗掉
- **設備什麼都沒做，而且不會有任何錯誤提示**

所以要還原，順序一定是：先填回 Cloud Token，再觸發還原。 走設定頁 §4.1 填，或用指令：

BASH

```
# 注意：JSON 不能有空格 —— 韌體是字串比對，有空格會靜默失敗
curl -s -X POST -H "Authorization: Bearer $TOK" -H "Content-Type: applicati
  -d "{\"enabled\":true,\"cloudUrl\":\"$CLOUD\",\"cloudToken\":\"$CLOUD_TOK
  $DUT/api/configserver

# 讀回確認再往下，不要憑「回了 success」就相信
curl -s -H "Authorization: Bearer $TOK" $DUT/api/configserver | python3 -c
import sys,json;d=json.load(sys.stdin)
print('  cloudUrl  :',d.get('cloudUrl') or '(空)')
print('  cloudToken:','已填' if d.get('cloudToken') else '(空!)')"
```

填好之後才觸發還原（見 §R.5）。實測時間：排入後約 2 分鐘設備心跳取件，下載完會自行重開，再約 20 秒設定回來。

（此缺陷已開卡 WI-194。修好之前，這一步都必須手動做。）

通過(8320 且未接網路線):設備自行重開並**開出 Wi-Fi 熱點** —— 它已經不記得任何網路了。
通過(8310,或 8320 插著網路線):這類機器**不會開熱點**,也不該期待它開。改用下面這條判準:重開後用 §2B.2 掃描找到它,開啟網頁應看到**設定全空**(RS485 清單、規則、通道皆為空)。
沒過:8320 拔掉網路線重開後仍自動連上原本的 Wi-Fi → 沒清乾淨(可能只做到 L1),改用 L3。

R.4 確認歸零結果

先確定要打哪個位址 —— 這一步最容易打錯:

情況	位址
8320 且未接網路線(已開熱點)	連上熱點後用 `http://192.168.3.1`
8310,或接著網路線的 8320	**不會開熱點**,用 §2B.2 掃到的新 IP

下面用 `$DUT` 代表該位址,先把它設成上表對應的值。

macOS / Linux | Windows

```
# 通道與規則各有自己的端點,不在 /api/config 裡面
for ep in tcpio rules signals actions; do
  printf '  %-8s %s\n' "$ep" "$(curl -s -H "Authorization: Bearer $TOK" $DUT/api/
done
curl -s -H "Authorization: Bearer $TOK" $DUT/api/config | python3 -c "
import sys,json
d=json.load(sys.stdin)
print('  RS485 設備:',len([x for x in (d.get('modbusDevices') or []) if x.get('na
print('  Wi-Fi SSID:',(d.get('wifi') or {}).get('ssid') or '(空)')
"
```

通過:五項全空,長得像這樣 ——

```
tcpio    {"count":0,"channels":[]}
rules    {"rules":[],"ruleCount":0}
signals  {"signals":[],"signalCount":0}
actions  {"actions":[],"actionCount":0}
RS485 設備: 0 台
Wi-Fi SSID: (空)
```

沒過:任何一項還留著舊資料 → 沒清乾淨,重做 L3。**不要帶著殘留設定往下走**,那會讓後面每一個異常都多一種可能原因。

⚠ **不要拿 `/api/config` 裡的 `tcp` 欄位來判斷通道**。那個欄位是 `{"port":5000}` —— 監聽埠,不是通道清單,**通道有幾條它都長一樣**。真正的清單只在 `GET /api/tcpio`。同理,規則/訊號/動作也都不在 `/api/config` 裡。

⚠ 歸零之後打開網頁,你會看到「資產救援」頁 —— 這是正常的

L2 會**格式化整個 QSPI**,連網頁介面本身(`index.html` ／ `app.js` ／ `styles.css` ／ 語系檔／ `log.html` ／ `workflow.html`)一起清掉。所以歸零後的設備:

- **API 完全正常** —— 上面 § R.4 那些 `curl` 全部照跑不誤
- **但沒有可用的網頁 UI** —— `/` 會回「MES Gateway — 資產救援」頁,`/app.js`、`/styles.css`、`/log.html`、`/workflow.html` 全部 404

不要以為機器壞了。網頁介面是靠上傳 `.mesb` 帶回來的 —— 交機流程往下走到 § 2C 上傳正式版韌體時就會恢復。

⚠ **但如果你是「清錯了要救回來」**:§ R.5 的雲端還原**只還原設定,不還原網頁資產**。設定回來了、畫面還是救援頁 —— 這時要另外推一次 `.mesb`(救援頁上就有上傳鈕)。

> R.3 的判準刻意選「會不會開熱點」，因為那是**設定真的被清光的外部證據** —— 它已經不記得任何網路了。比「畫面看起來是空的」可靠得多。

R.5 從雲端還原(只在「清錯了、要救回來」時做;正常交機不需要)

前提: § R.3 的 🛑 區塊已完成 —— **Cloud Token 已填回並讀回確認**。

```bash
# ① 列出可用快照,挑一筆(檔名長得像 snapshot.<一長串數字>.v5.json)
curl -s -H "Authorization: Bearer $CLOUD_TOK" \
  "$CLOUD/api/devices/$UID_DEV/snapshots?limit=10" | python3 -c "
import sys,json
for x in json.load(sys.stdin): print('  ',x.get('filename'))"

# ② 【重要】先確認設備沒有待推送,否則還原會被蓋掉(見下方警告)
curl -s -H "Authorization: Bearer $TOK" $DUT/api/configserver | python3 -c "
import sys,json;d=json.load(sys.stdin);c=(d.get('push') or {}).get('cloud') or {}
print('  needsPush :',d.get('needsPush'),'  ← 必須是 False')
print('  上次推送  : ',c.get('ageS'),'秒前')"

# ③ 觸發還原(把 <檔名> 換成上面挑的那一筆)
curl -s -X POST -H "Authorization: Bearer $CLOUD_TOK" \
  "$CLOUD/api/devices/$UID_DEV/snapshots/<檔名>/rollback"

# ④ 等設備心跳取件(約 2 分鐘),它會自己下載並重開。輪詢到不是 0 為止
for i in $(seq 1 12); do
  sleep 20
  printf '  %s ruleCount=' "$(date +%H:%M:%S)"
  curl -s -H "Authorization: Bearer $TOK" $DUT/api/rules | python3 -c "
import sys,json;print(json.load(sys.stdin).get('ruleCount'))"
done
```

通過:規則/訊號/動作/RS485/SSID 全部回到歸零前的數量 —— 用 § R.4 那條指令再跑一次比對。**沒過(最常見)**:`ruleCount` 一直是 0 → **十之八九是 Cloud Token 沒填**。回 § R.3 的 🛑 區塊確認,然後**重新觸發一次還原** —— `pendingCommand` 是一次性的,失敗那次已經被消耗掉,不會自己重試。**查證方法**:`GET $DUT/api/log` 找 `[CS-Cloud] Command: download_update` 與 `[ConfigServer] (Cloud) Downloading update from …`。兩行都有 = 設備確實在做事。⚠️ 該日誌是環形緩衝、會被 `[ALIVE]` 洗掉,要在觸發後幾十秒內去看。

🛑 千萬不要「剛改完設定就馬上觸發還原」

還原的做法是**把快照寫進雲端的設定檔,再叫設備來下載**。而設備的自動推送 (autoPush)會把**它自己當下的設定推上同一個檔案** —— 只要這兩件事撞在一起, 還原內容就被蓋掉,設備下載回來的是它自己剛推上去的東西。

實測:填完 Cloud Token 後立刻觸發還原 → 12 秒後設備推送 → 還原被覆蓋 → 看起來「還原成功但設定沒變」。

而 § R.3 的 🛑 區塊正好要你先填 Cloud Token —— **填 token 就是改設定,會排一次推送**。所以順序必須是:

1. 填回 Cloud Token
2. **等到 `needsPush` 變回 `False`** (通常十幾秒,用上面第 ② 步確認)
3. 才觸發還原

三個環節都會回報成功(rollback `success`、push `OK HTTP 200`、下載也真的完成), 錯的只有順序 —— 所以最容易誤判成「還原功能壞了」。

(此缺陷已開卡 WI-195。)

05

第一部：安裝與設定

階段 0：開工前現場條件確認

步驟	動作	通過	沒過
0.1	確認供電與接地	電壓在銘牌範圍內，機殼確實接地	停止安裝。供電不穩造成的間歇重開，日後極難查
0.2	RS485 佈線：A/B 極性、終端電阻、Slave ID 不重複	全線 A 接 A；兩端各一顆 120Ω；ID 互不重複	**ID 重複會兩台搶答**，現象是數值跳動或時有時無，看起來像設備壞掉
0.3	網路：網段、DHCP、對外連線	同網段筆電能上網	若現場不開放對外，雲端備份與 MQTT 要改內網方案，先與業主確認

這台機器走哪一條路？先分流

兩個問題決定你接下來要做什麼，走錯會白費半小時：

問題	選項 A	選項 B
機器狀態	全新未開封 → **不需要歸零**，直接做階段 1	動過的 → 先做**階段 R 歸零**，再回到這裡
連線方式	**8320（有 Wi-Fi）** → 走 §2A「無線」	**8310（無 Wi-Fi，只有網路孔）** → 走 §2B「有線」

怎麼確認機型？機身標籤有寫；設備能連上之後也可以從畫面或 API 確認（`boardModel`）。

> **8310 沒有 Wi-Fi，永遠不會開出熱點。**文件裡凡是提到「連上設備的熱點」，對 8310 都不適用 —— 它一律走網路線。這是最常見的誤解來源。

階段 1：空機燒入韌體（拓荒包）

全新設備出廠時**沒有我們的韌體**，要先燒一個「拓荒包」進去。拓荒包是一個精簡版，它只做一件事：讓你能連上設備、設定網路，然後**把正式版韌體上傳進去**。

> **為什麼分兩段？** 正式版韌體很大，第一次只能靠 USB 燒；但正式版更新頻繁，每次都接 USB 不實際。拓荒包很少改動，燒一次就好，之後的版本都用網頁上傳。

1.1 安裝工具（每台電腦只需做一次）

macOS / Linux | Windows

```
brew install arduino-cli
arduino-cli core update-index
arduino-cli core install arduino:mbed_opta
```

> Windows 使用者請至 Arduino CLI 官網下載執行檔並加入環境變數。

1.2 取得拓荒包

macOS / Linux | Windows

```
curl -O https://opta.smms.com.tw/api/firmware/download/mes-gateway-bootstrap.bin
```

> **通過**：檔案下載完成且大小合理（數百 KB）。也可以自己建：在專案目錄執行 `bash scripts/build_bootstrap.sh`，產出在 `release/mes-gateway-bootstrap.bin`。

1.3 用 USB 接上設備，找出它的 Port

```
arduino-cli board list
```
BASH

通過:清單中出現標示 **Opta** 或 **mbed** 的項目,記下它的 Port (macOS 長得像 `/dev/cu.usbmodem212201`,Windows 像 `COM3`)。**沒過**:沒看到任何裝置 → 換一條**能傳資料**的 USB 線。只能充電的線不會被電腦看到。

1.4 燒錄

macOS / Linux | Windows

```
arduino-cli upload -b arduino:mbed_opta:opta \
  -p /dev/cu.usbmodem212201 \
  -i mes-gateway-bootstrap.bin
```

把 `-p` 後面換成 1.3 查到的 Port。**通過**:指令回報成功,設備自動重開。**沒過**:權限錯誤 → 確認沒有其他程式(Arduino IDE、序列埠監控)佔用該 Port。

1.5 等首次開機完成

通過:**心跳燈**開始規律閃爍 —— 亮滅各約 0.5 秒,**1 秒一個週期**。

⚠ **心跳燈是哪一顆、什麼顏色,取決於機型** —— 盯錯燈會以為機器沒活過來:

機型	心跳燈
8320(有 Wi-Fi)	機身 **USER LED**,亮**藍**色
8310(無 Wi-Fi)	沒有藍燈硬體,改用**琥珀色**(紅+綠同時亮)

純紅 = 故障、純綠 = DFU,兩者都不是心跳。

⚠ **快閃(約 0.2 秒一個週期)不是故障。**它代表「MQTT 已啟用但還沒連上」,在剛設好 MQTT、還沒連上 broker 的期間出現是正常的。全新設備還沒設 MQTT,看到的會是慢閃。

沒過:全新設備第一次開機要建立檔案系統,可能要 1–2 分鐘,**期間絕對不要斷電**。超過 3 分鐘心跳燈仍完全沒有動靜(不閃也不亮)→ 回到 1.4 重燒。

階段 2A：連上設備 —— 無線機型（8320）

剛燒好的設備**還不知道你的網路**，所以它會自己開一個 Wi-Fi 熱點等你進去設定。

> **同樣的情況也發生在做完「階段 R 歸零」之後** —— 網路帳密被清光，設備就回到這個狀態。這是正常的，不是故障。

2A.1 用手機或筆電連上設備開出的熱點

項目	值
熱點名稱（SSID）	`MES-Gateway-Setup`
密碼	`12345678`
連上後開	`http://192.168.3.1`

> **通過**：連上後取得 `192.168.3.x` 網段的 IP。**沒過**：找不到熱點 → 設備可能還記得舊的 Wi-Fi 而直接連上了現場網路，改用 §2B.2 的掃描工具找它的 IP。

2A.2 瀏覽器開 `http://192.168.3.1`

> **通過**：看到**「🧭 拓荒包・MES Gateway 開荒」**頁面，上面有設備 UID、網路設定、以及「上傳 .mesb」按鈕。**這個畫面和正式版完全不同，看到它是正確的** —— 代表拓荒包燒成功了。

2A.3 在拓荒頁填入現場 Wi-Fi 帳號密碼並儲存，設備會重開

> **通過**：設備重開後不再開熱點，改為連上現場網路。**沒過**：又開出熱點 = 帳密錯誤或訊號不足。重連熱點再試一次。

2A.4 找出設備在現場網路上的新 IP → 見 §2B.2（兩種機型共用）

階段 2B：連上設備 —— 有線機型（8310）

8310 沒有 Wi-Fi，**不會開熱點**。它一開機就用網路線去要一個 IP（DHCP），所以問題不是「怎麼連上它」，而是**「它拿到哪個 IP」**。

2B.1 接上網路線，等 30 秒讓它取得 IP

2B.2 找出設備的 IP（兩種機型都適用）

專案內附一支掃描工具：

把下面這段存成 `find-gateway.sh`（放哪都可以），`chmod +x find-gateway.sh` 後執行：

```bash
#!/usr/bin/env bash
# 找出網段上的 MES Gateway。判準是「誰以我們的方式回應 API」，不是 MAC ——
# 設備支援軟體 MAC 覆寫，實測過 API 回報 A8:61:0A、arp 表卻是 50:26:EF。
SUB=${1:?用法: ./find-gateway.sh 192.168.1}
TOKEN=${2:-<設備管理token>}
echo "掃描 $SUB.1-254 …"
for i in $(seq 1 254); do
  ( probe=$(curl -s -m 3 "http://$SUB.$i/api/system" 2>/dev/null)
    case "$probe" in *Unauthorized*|*boardModel*)
      info=$(curl -s -m 3 -H "Authorization: Bearer $TOKEN" "http://$SUB.$i/api/s
      v=$(printf '%s' "$info" | sed -n 's/.*"version":"\([^"]*\)".*/\1/p')
      m=$(printf '%s' "$info" | sed -n 's/.*"boardModel":"\([^"]*\)".*/\1/p')
      echo "  OK  $SUB.$i   ${m:- (token 不符，無法讀型號) }   ${v:+v$v}" ;;
    esac ) &
done
wait
echo "掃描結束"
```

執行：`./find-gateway.sh 192.168.1`（換成你現場的網段前三碼）

```powershell
# 直接貼進 PowerShell 7+ 執行，無需額外工具或檔案
$SUB   = "192.168.1"                  # 換成你現場的網段前三碼
$TOKEN = "<設備管理token>"
1..254 | ForEach-Object -Parallel {
  $ip = "$using:SUB.$_"
  try {
    $r = Invoke-WebRequest -Uri "http://$ip/api/system" -TimeoutSec 3 `
           -SkipHttpErrorCheck -ErrorAction Stop
    if ($r.Content -match 'Unauthorized|boardModel') {
      $info = Invoke-WebRequest -Uri "http://$ip/api/system" -TimeoutSec 3 `
                -Headers @{ Authorization = "Bearer $using:TOKEN" } `
                -SkipHttpErrorCheck -ErrorAction SilentlyContinue
      $m = if ($info.Content -match '"boardModel":"([^"]*)"') { $Matches[1] } els
      $v = if ($info.Content -match '"version":"([^"]*)"')    { "v" + $Matches[1]
      "  OK  $ip   $m   $v"
    }
  } catch {}
} -ThrottleLimit 64
# PowerShell 5.1 沒有 -Parallel / -SkipHttpErrorCheck，請改用 PowerShell 7+
```

把 `192.168.1` 換成你現場的網段前三碼（不知道的話，看自己電腦的 IP 前三碼）。

> **通過**：印出類似 `OK <網段>.46 Opta WiFi (8320) v<版本>`，這就是設備的 IP、機型和韌體版本。**沒過**：什麼都沒找到 → 見下方「找不到設備時」。

> **這支工具是靠「誰以我們的方式回應 API」來認人，不是靠 MAC。**設備支援軟體 MAC 覆寫 —— 實測過一台設備 API 回報 `A8:61:0A:…`、網路上的 arp 表卻是 `50:26:EF:…`，用 MAC 前綴去找會漏掉。

找不到設備時，依序試：

1. **網段填錯**：確認你電腦的 IP。`ipconfig getifaddr en0`（macOS）或 `ipconfig`（Windows）
2. **查路由器的 DHCP 用戶端清單** —— 最可靠，直接列出所有連上的裝置
3. **設備沒拿到 IP**：確認網路線兩端都插好、對方交換器埠有燈
4. **8320 而且從未設定過**：它在 AP 模式，回到 §2A

2B.3 瀏覽器開 `http://<找到的IP>`

> **通過**：看到「🧭 拓荒包・MES Gateway 開荒」頁面（若還在拓荒包階段），或完整的五頁籤設定畫面（若已裝正式版）。

階段 2C：把正式版韌體裝上去

拓荒包只是讓你能連上設備。**正式版功能都還沒有** —— 沒有 RS485、沒有規則、沒有 MQTT。

2C.1 取得正式版韌體（`.mesb` 檔）

向專案負責人索取，或從雲端韌體清單下載當次交付指定的版本。

> `.mesb` 是「韌體 + 網頁資源」打包成的單一檔案，用它更新可以一次到位。

2C.2 在拓荒頁按「上傳 .mesb」，選擇該檔案

> **通過**：上傳完成後設備自動重開，約 2–3 分鐘後重新連上，這時畫面變成**完整的五個頁籤**（總覽／配置／規則／設備／設定）。**沒過**：上傳中斷 → 確認電腦與設備在同一網段、中途不要關瀏覽器。拓荒包不會被破壞，失敗可以重來。

2C.3 確認版本

macOS / Linux | Windows

```
curl -s -H "Authorization: Bearer $TOK" $DUT/api/system | python3 -m json.tool |
```

> **通過**：`"version"` 與交付文件指定的版本一致。**沒過**：版本不符 → 上傳到舊檔案了，回到 2C.1。**版本必須對得上，否則這份驗收單不成立。**

階段 3：網路設定收尾

3.1 確認設備連在正式網路上、IP 穩定

現場建議用**固定 IP 或 DHCP 保留**，否則設備重開後 IP 可能改變，上位系統就找不到它了。

3.2 記錄設備 IP 與 UID，填進本單開頭，並更新環境變數

macOS / Linux | Windows

```
curl -s -H "Authorization: Bearer $TOK" $DUT/api/config | python3 -c "
import sys,json
d=json.load(sys.stdin)
print('  UID :',d.get('uid'))
print('  MAC :',d.get('mac'))
"
```

> 後續維運都靠這兩個值找設備，一定要記下來。

階段 4：雲端備份設定

本階段的目的不是「有備份」，而是**證明備份讀得回來**。這兩件事不一樣，而且差別很危險 —— 備份可能只壞一半：**推得上去、讀不回來**。

步驟	動作	通過	沒過
4.1	填入雲端網址與 Cloud Token	兩個欄位都有值	**Cloud Token 空白是最容易漏的一項** —— 網址有值、Token 沒填，備份看起來正常但查不到任何歷史
4.2	按「立即推送」	回報成功	檢查對外是否放行 HTTPS
4.3	**按「查看版本歷史」**	**列出快照清單，看得到剛推的那一筆**	出現 `Unauthorized` → Token 沒填或不正確，回 4.1

> ⚠️ **驗 4.2/4.3 之前先做一個實際的設定變更**(隨便改一個沒用到的通道名稱就好)。雲端對**內容相同**的推送會去重 —— 設定沒變時推送照樣回成功,但**不會產生新快照**, 看起來就像推送失敗。走查時因此誤判過一次。
>
> **另外**:設備端的雲端設定要查 `/api/configserver`, **`/api/config` 裡沒有這個區塊** —— 拿 `/api/config` 去看會什麼都讀不到,誤以為 Token 沒填。

> **4.3 才是真正的驗收點。**實際發生過：推送成功、清單打不開，不特地點開就不會發現。
>
> **想用指令驗而不點畫面** —— 用 §R.2 的第 ③ 條(打雲端的 `/api/devices/$UID_DEV/snapshots`)。**不要打設備的 `/api/configserver`** —— 那條永遠不會吐清單,詳見 §R.2 的警告。

階段 5：物理 I/O —— RS485 設備

先把「設備是誰、值從哪來」定義好。沒有這一層，後面的連動與 MQTT 上行都沒有素材可用。

5.1 掃描匯流排，確認實體設備都在

掃描是**非同步**的：POST 只是「按下開始」，馬上就返回，結果要另外輪詢。它會逐一試 `9600 / 19200 / 4800 / 2400` × `8N1 / 8E1 / 8N2`，**要跑數十秒到數分鐘**。

macOS / Linux | Windows

```
# ① 觸發掃描（body 不需要，韌體不看）
curl -s -X POST -H "Authorization: Bearer $TOK" $DUT/api/modbus/scan

# ② 每 10 秒輪詢一次結果，直到掃描結束
curl -s -H "Authorization: Bearer $TOK" $DUT/api/modbus/scan | python3 -m json.t
```

⚠️ **掃描期間整台設備會「停住」約 1 分鐘 —— 那是正常的,不是當機。**

停住的不只是網頁。實測(v6.0.78)單次掃描把**主迴圈壓住 65 秒**:

```
[SLOW LOOP] total=65532ms segs: ... scan=65202 ...
```

這 65 秒內:

- **網頁與 API 拒連** —— 收到 `Connection reset by peer` 或整個逾時
- **實體 I/O 不會動** —— 呼吸燈停住、DI 不被輪詢、規則不執行

它正逐一試 **7 種鮑率 × 同位元 × 32 個 slave ID**,而且是同步輪詢。看門狗有逐台餵,所以**不會重開**(實測 `bootCount` 全程不變)。

因此:不要在掃描期間測按鈕,也不要因為呼吸燈停了就判定設備掛掉。輪詢每 15 秒問一次,拒絕連線就當成「還在掃」,最長等 5 分鐘。

(這個成本已開卡追蹤 —— **WI-209**:掃描應切片化,或縮小預設掃描範圍, 至少 UI 按下去時要先告知「約 1 分鐘,期間設備不回應」。)

通過:輪詢結果中列出的 Slave ID 與硬體表一致。

注意回應的鍵是 `devices` (不是 `results`)—— 每一筆長這樣:`{"slaveId":1,"baud":9600,"found":true,"alreadyAdded":true}`。**常見誤判**:POST 立刻回 `{"success":true,"scanning":true}` —— 那只代表「開始掃了」,**不是掃描結果**。看到它就以為完成,會誤判成「一台都沒掃到」。**注意**:掃到只代表「設備活著」,**不代表讀得到值** —— 讀值還需要暫存器設定正確。

5.2 逐台加入 —— ⚠️ 必須用網頁新增,不能用 CLI

> **這一步是本文件唯一「不提供 CLI」的步驟，而且是刻意的。**型號的暫存器模板（要讀哪個位址、怎麼解讀）**只存在於網頁端**，是瀏覽器幫你填好之後才送出去的。用 `curl` 直接 POST 會建出一台 **沒有任何暫存器的設備** —— 它在清單上看得到、`online` 也是 true，但**永遠讀不到值**，而且不會有任何錯誤訊息。

操作路徑：

`設備頁` → **RS485 設備** → **＋新增** →

欄位	填什麼
名稱	現場看得懂的名字（例：`鑽床1電流`），不要用 `Device1`
型號	計數器選 **SF965**；電流計選 **Finder 6M**
Slave ID	該設備實際設定的位址
輪詢間隔	`2000` ms

> **選了型號之後，暫存器會自動帶入** —— 看到下方「自訂暫存器」清單被填滿，才代表模板套上了。**如果那個清單是空的，不要儲存**，重選一次型號。

一次只加一台，加完立刻做 5.3 確認讀得到值，再加下一台。全部加完才一起測，出問題會分不清是哪一台。

> 若回報 `Slave ID already in use` → 該 ID 已被佔用（**包含閘道器自己的 slaveId**），換一個。

5.3 確認讀得到值 —— 唯一可信的判準

電流計類（Finder：電壓／電流／功率是固定欄位）：

```bash
curl -s -H "Authorization: Bearer $TOK" $DUT/api/poll | python3 -c "
import sys,json
d=json.load(sys.stdin)
for p in d.get('power',[]):
    print(f\"  {p.get('name')}  online={p.get('online')}  V={p.get('voltage')}  /
"
```

計數器類（SF965：值在自訂暫存器，**不在** `power` 陣列裡）：

```bash
curl -s -H "Authorization: Bearer $TOK" $DUT/api/modbus | python3 -c "
import sys,json
d=json.load(sys.stdin)
print('  暫存器目前值:', d.get('registers'))
"
```

> **通過**：電流計 `online=True` 且數值合理；計數器的暫存器值會隨現場觸發而增加。**`power` 陣列只有電力類欄位（電壓／電流／功率），計數器的值不會出現在那裡** —— 拿電流計的指令去驗計數器，會看到一排 0 而誤判成故障。
>
> **這段指令能跑完不報錯，本身就是一項驗證** —— 若回應不是合法 JSON，Python 會直接拋錯。

分辨「沒人答」還是「答了但解不開」：

macOS / Linux | Windows

```
curl -s -X POST -H "Authorization: Bearer $TOK" -H "Content-Type: application/jso
  -d '{"slaveId":4,"functionCode":3,"regAddress":0,"count":2}' $DUT/api/modbus/pr
```

> 回 **0 bytes** = 沒人答（線路／位址／電源）；**有 byte 但解不開** = 撞號或格式錯。

5.4 現場觸發計數器，看畫面數字是否跟著跳

通過：**面板顯示值與網頁顯示值一致**，觸發一次就加一。這是業主最直觀的驗收點，請當場示範。
沒過：讀到 0 → 依序查接線極性 → Slave ID → 鮑率 → 該台有沒有電。數值有但對不上面板 → 型號模板選錯。

5.5 全部加完後重開設備，再確認一次

macOS / Linux | Windows

```
curl -s -X POST -H "Authorization: Bearer $TOK" $DUT/api/system/reboot
# 等約 60 秒後重跑 5.3
```

通過：重開後設備清單完整、數值正常。**這一步證明設定真的寫進去了，不只是畫面上有。沒過**：重開後設定不見 → **立即停止驗收並回報**，不可帶病上線。

階段 6：I/O 連動

6.1 看目前實體 I/O 狀態

macOS / Linux | Windows

```
curl -s -H "Authorization: Bearer $TOK" $DUT/api/io | python3 -m json.tool
```

通過：看得到 `di` / `do` 陣列，現場觸發輸入時對應位置會變。

6.2 直接驅動輸出，確認實體會動（不經規則，單純驗硬體）

macOS / Linux | Windows

```
curl -s -X POST -H "Authorization: Bearer $TOK" -H "Content-Type: application/jso
  -d '{"index":2,"value":true}' $DUT/api/io/do      # DO3 = index 2 (0 起算)
sleep 2
curl -s -X POST -H "Authorization: Bearer $TOK" -H "Content-Type: application/jso
  -d '{"index":2,"value":false}' $DUT/api/io/do
```

> **先確認硬體會動，再談規則** —— 否則規則不動時分不清是規則錯還是輸出壞。欄位是 `index` ／ `value`，不是 channel／state。打錯欄位名會回 400 並附提示。

06

第二部：情境驗收

前面驗的是「裝好了」，這一部驗的是**照情境會動**。每個情境都標註出廠測試中守著同一件事的測項編號 —— 驗收與出廠測試看的是同一組行為，不是兩套標準。

情境 A：I/O 互動

> **前置：A1／A3 要驗的「規則」必須先建立。** 規則不會憑空存在，而且它需要兩個先決條件 —— 一個輸入群組、一個輸出群組。
>
> ⚠️ **你需要建【兩個】輸入群組，不是一個。** A1 要實體 DI 當來源、A2 要 MQTT 命令當來源，兩者不能共用同一個群組。下面 A.0-1 請做兩次。

A.0-1 建立輸入群組（觸發條件）

`配置頁` → 📡 **輸入群組** → 右上角 `+` → 名稱（例：`按鈕1按下`）→ `+ 加設備` → 選來源 → 設比較條件 → 儲存

> 按鈕叫 `+ 加設備`，不是「新增來源」。

⚠️「觸發類型」這一欄決定 A1 的後半段會不會成立 —— 務必看懂再選

新增輸入群組的畫面上有一個「觸發類型」下拉，**預設是** `Always`。它決定訊號在什麼 時機才算「觸發」，直接影響規則會不會執行：

觸發類型	行為	用在 A1 的後果
`Always`（預設）	每個主迴圈都觸發（準位模式）	會動，但持續重複驅動輸出
`Rising`	只在 0→1 那一瞬間觸發	**「解除→復歸」永遠不會發生**，A1 必定失敗
`Falling`	只在 1→0 那一瞬間觸發	只會復歸，不會動作
`Change`	**任何變化都觸發（0→1 與 1→0 皆是）**	✅ **A1 要選這個**

原因在韌體的規則比對：`if (rule->triggerOnTrue != signalValue) continue;`。選 `Rising` 時只有 `signalValue == true` 的那一刻會觸發，於是 `觸發條件 = 信號為 False` 的那條規則**永遠不會被執行**。

「按鈕**按下**」直覺上會讓人選 `Rising` —— 那正是這一題的陷阱。

> **例外：脈衝式輸入要選 `Rising` + 輸出用「切換」。**觸控式按鈕這類只給短脈衝（1 → 幾秒內自動回 0）的輸入，不是準位，要搭配輸出模式「切換」才會有一按一放的效果。

A.0-2 建立輸出群組（要做什麼）

`配置頁` → **輸出群組** → 右上角 `+` → 名稱（例：`開燈1`）→ `+ 加設備` → 選 DO 通道與動作模式 → 儲存

> **輸出模式：**`直接`＝跟隨、`切換`＝每次觸發翻面、`脈衝`＝定時、`關閉`＝強制 OFF。

> ⚠️ **不要一口氣連續建立多筆。**每建好一筆，**等清單重新整理出現該筆之後**再建下一筆。連續快速新增會讓前一筆被無聲覆蓋，而且兩次都回成功（見 WI-199）。

A.0-3 建立規則(把兩者綁起來)

`規則頁` → 新增 →

欄位	填什麼
規則名稱	看得懂的名字
觸發信號 (輸入群組)	選 A.0-1 建的那個
觸發條件	`信號為 True`(解除時要復歸的話,另外再建一條 `信號為 False` 的規則)
執行動作 (輸出群組)	選 A.0-2 建的那個

> ⚠️ **兩個下拉都必須真的選到東西。**畫面會擋:「請先選擇『觸發信號 (輸入群組)』與『執行動作 (輸出群組)』。**未綁定的規則永遠不會觸發。**」若下拉是空的、顯示「請先新增輸入群組」,表示 A.0-1／A.0-2 還沒做完。
>
> **存完重新整理頁面再看一次** —— 確認兩個欄位仍顯示正確的選項,而不是又變回空白。

編號	動作	通過	出廠對應
A1	實體輸入驅動實體輸出	輸入成立→輸出動作;解除→復歸。**連續 3 次一致**	H5・H2
A2	MQTT 命令驅動實體輸出 `mq_pub -t "$PREFIX/cmd/signal/1" -m '1'` (A.0-1 的輸入群組來源要選「📨 MQTT 命令訊號」並綁 `cmd/signal/1`)	送出後數秒內輸出動作	CV.2
A3	停用規則→觸發;重新啟用→觸發	停用後**完全不動**;重新啟用**恢復正常**	EN.1

> A1 只成功一次不算通過 —— 間歇問題只有重複才看得出來。A3 兩個方向都要驗:「停用了還會動」= 業主以為關掉、機器卻自己跑(安全問題);「重新啟用回不來」= 現場只能重開機碰運氣。

> ⚠️ **A3 的假 FAIL 陷阱（`Change` 觸發專屬，實測踩到過）**
>
> 若輸入群組的觸發類型是 `Change`，而你在**規則停用期間**已經送過一次觸發值（例如 MQTT 送了 `1`），那麼重新啟用後**再送一次相同的值不構成變化 → 不觸發 → 輸出不動**。
>
> 這**不是**「重新啟用失敗」，是測試步驟本身沒有製造變化。
>
> **正確做法**：重新啟用之前先把訊號送回原值（送 `0`），再送 `1`。實體 DI 同理 —— 先把開關撥回原位再撥一次。

> **MQTT topic 前綴用 UID，不是設定頁顯示的那個。**設備「設定」頁的 `topicPrefix` 可能顯示成 `mes/gateway/001` 之類的短代號，但實際訂閱／發佈用的是 **UID**：`mes/gateway/<24 碼 UID>/...`。一律以本文件開頭的 `PREFIX=mes/gateway/$UID_DEV` 為準；照設定頁的顯示值發命令會石沉大海。要確認的話，訂閱 `mes/gateway/#` 看設備實際發出來的 topic 長什麼樣。

情境 M：MQTT 的三種通道模式 —— 先搞清楚差別

新增 TCP 通道、協定選 MQTT 之後，會出現一個「模式」下拉，**三個選項的用途完全不同**。選錯的話畫面上不會報錯，但行為完全不是你要的。

模式（介面文字）	方向	它做什麼	值的型別
⚡ **收取並觸發動作 (Legacy)**	入向／出向	收到訊息就**觸發**（或比對到特定值才觸發）；出向時在動作發生後送出固定字串	**開關**（成立／不成立）
📩 **收取並擷取數值 (Parser)**	入向	從收到的 JSON 中**抽出數值**存進通道，供規則比較或 Converter 引用	**數值**
組合並發佈訊息 (Converter)	出向	把資料**組合成 JSON 發佈出去**（見情境 B）	—

> **一句話分辨：**要「**有沒有發生**」→ Legacy。要「**是多少**」→ Parser。要「**送出去**」→ Converter。

M.1 — Legacy 入向：收到指定訊息就觸發

目的：上位系統送一個命令，設備就動作。**不關心數值是多少**，只關心「這件事發生了沒」。例：`{"cmd":"start"}` 一到就啟動某個輸出。

設定： `配置頁` → **TCP 通道** → **＋新增** → 協定 `MQTT`、方向 `輸入`、模式 ⚡ **收取並觸發動作 (Legacy)** →

欄位	填什麼
Topic	`<你的 $PREFIX>/cmd/start`
JSON 欄位	`cmd` — **留空則比對整個 Payload**
比對值	`start` — **留空則「收到訊息就觸發」，不管內容**

macOS / Linux | Windows

```
mq_pub -t "$PREFIX/cmd/start" -m '{"cmd":"start"}'      # 應觸發
mq_pub -t "$PREFIX/cmd/start" -m '{"cmd":"other"}'      # 不應觸發
```

通過：送 `start` → 通道狀態變成成立（總覽頁看得到，或綁規則後輸出動作）；送 `other` → **完全沒有反應**。**沒過**：兩個都觸發 → **比對值沒填**，變成「收到就觸發」。兩個都不觸發 → JSON 欄位名打錯，或 Topic 不符（大小寫要完全一致）。

M.2 — Legacy 的「留空」語意（這是最容易誤解的地方）

macOS / Linux | Windows

```
# 把「比對值」清空後再送一次
mq_pub -t "$PREFIX/cmd/start" -m '{"cmd":"whatever"}'
```

通過：比對值留空時，**任何內容都會觸發** —— 這是設計，不是故障。**為什麼要驗這個**：現場常見「我明明設了條件，怎麼什麼都會動」，十之八九就是比對值留空。當場示範一次，業主就懂了。

M.3 — Parser 入向：從 JSON 抽出數值

> **目的**：上位系統送一包 JSON，設備要把**其中一個數字**取出來，之後才能拿它做門檻比較（例：溫度 > 80 就報警）或塞進 Converter 發佈。

設定：新增通道 → 協定 `MQTT`、方向 `輸入`、模式 📩 **收取並擷取數值 (Parser)** → Topic 填 `<你的 $PREFIX>/data/temp` → 在**解析欄位**區 **新增至少一個欄位**（欄位名對應 JSON 的鍵，例 `temp`）。

> 介面會擋：「**請至少新增一個解析欄位。**」沒新增就存不了。

macOS / Linux | Windows

```
mq_pub -t "$PREFIX/data/temp" -m '{"temp":25.5,"hum":60}'
sleep 3
curl -s -H "Authorization: Bearer $TOK" $DUT/api/tcpio | python3 -c "
import sys,json
for c in (json.load(sys.stdin).get('channels') or []):
    if c.get('name'): print(f\"  {c.get('name')}  目前值={c.get('currentValue')}\"
"
```

> **通過**：該通道的 `currentValue` 變成 **25.5**（不是 0、不是 1）。**這是 Parser 與 Legacy 的關鍵差別** —— Legacy 只會給你「成立／不成立」，Parser 給的是**真正的數字**。**沒過**：值一直是 0 → 解析欄位名與 JSON 的鍵不符（大小寫要一致）。值變成 1 → 模式選成 Legacy 了。

M.4 — Parser 的數值拿去做規則判斷:能到什麼程度(這裡有一個硬限制)

🛑 先講清楚,免得現場白試半小時

Parser 抽出來的數字,不能拿去設數值門檻。規則引擎讀 TCP 通道時只讀它的 **布林狀態**(`值 ≠ 0` → 成立、`值 = 0` → 不成立),**不比較那個數字本身**。

所以「溫度 > 80 就報警」這種需求,**目前這條路做不到** —— 要嘛請上位系統自己判斷後送一個 0/1 過來,要嘛走 RS485 暫存器那條路(那條支援數值門檻)。

韌體會**當場擋下來**,不會讓你存進一個永遠不成立的條件:

```JSON
{"error":"condition can never be true",
 "reason":"TCP IO channel sources are boolean (0/1); the parsed numeric valu
 "hint":"use op '==' with threshold 1 to mean 'non-zero'"}
```

設定:`配置頁` → 📡 **輸入群組** → 新增 → 來源選 M.3 那個 Parser 通道 → 比較條件設 `= 1`(意思是「非零」)→ 儲存 → 綁一條規則到某個輸出。

macOS / Linux | Windows

```
mq_pub -t "$PREFIX/data/temp" -m '{"temp":0}'      # 值為 0  → 不成立
sleep 4
mq_pub -t "$PREFIX/data/temp" -m '{"temp":35.0}'   # 值非 0  → 成立
sleep 4
mq_pub -t "$PREFIX/data/temp" -m '{"temp":0}'      # 回到 0  → 復歸
```

通過:`0` → 不成立、`35` → 成立、再回 `0` → 復歸。同時 `/api/tcpio` 的 `currentValue` 要跟著變(0 → 35 → 0)—— **數值有被抽出來,只是規則只看它是不是 0。沒過**:一直不成立 → 見下方「來源要填 channelId」。

⚠️ 兩個會讓你白忙的細節

① 來源的「index」是 `channelId`,不是通道在列表裡的第幾個。用 API 建輸入群組時,`sources[].index` 要填 `GET /api/tcpio` 回傳的 `channelId`(例 `81`),填成陣列位置(`1`)的話會查不到通道,條件**永遠不成立且沒有錯誤訊息**。走網頁介面不會踩到這個 —— 它會幫你填對。

② 來源型別要用 `7`(TCP 通道),不要用 `9`。型別 `9`(CH_TCPIO)送得進去、`HTTP 200`、列表上看得到,但規則引擎**沒有處理這個型別** —— 條件永遠不成立,而且上面那道「永遠不成立」的守衛也擋不到它(它只檢查型別 7)。實測:型別 9 配 `> 30`,送 25.5 / 35.0 / 99.9 進去,訊號**三次都是 False**。(已開卡 **WI-196**。同樣走網頁介面不會踩到。)

情境 B:MQTT Converter —— 兩個軸,四種組合

Converter 有兩個**彼此獨立**的設定軸,搞混就會得到「資料太多」或「該來的沒來」。

下表格子裡的編號 = 底下對應的測試步驟編號。**B3 不在這張表裡** —— 它驗的是「這個開關到底歸不歸你控制」(可逆性),跨在兩個象限之上。

	連續傳送(不勾「🔁 變更時才發」)	不重複傳送(勾選)
持續(無閘門)	**B1** 每 N 秒固定發,值沒變也發 用途:當心跳,證明設備還活著	**B2** 每 N 秒檢查,值變才發 用途:電流環等連續量測
啟動式(閘門觸發)	**B4** 閘門開→每 N 秒發;關→停 用途:班別／工單期間才回報	**B4b** 閘門開**且**值變才發 用途:計數器事件回報

先開一個終端機常駐訂閱,以下所有情境都在它上面看結果:

macOS / Linux | Windows

```
mq_sub -t "$PREFIX/#" -v | while read -r line; do
  echo "$(date +%H:%M:%S)  $line"
done
```

B1 — 電流環定期回報（持續 × 連續） ｜ 出廠對應 `CV.3`

> **目的**：不論值有沒有變，每隔固定秒數都要有一筆，讓 MES 能畫連續趨勢圖、也能當「設備還活著」的心跳。**此時「重複」是刻意要的。**

設定路徑（介面上**沒有**「來源」下拉，值是用 JSON 模板的變數插進去的）：

`配置頁` → **TCP 通道** → **＋新增** → 協定 `MQTT`、方向 `輸出`、模式 `Converter` → 展開綠色「Converter 設定」區 →

欄位	填什麼
Topic	`<你的 $PREFIX>/current` ← **必須在 `$PREFIX` 底下**，否則 `mq_sub -t "$PREFIX/#"` 收不到
推送間隔 (秒)	`5`
JSON 模板	`{"current":${rs485.1.current}}` ← `1` 換成該設備的 **Modbus Slave ID**

⚠️ **變數的欄位名必須用英文,寫中文會靜默得到 0。**

韌體 `resolveRs485Field()` 只認這些名稱: `voltage` ／ `current` ／ `power` ／ `reactive` ／ `apparent` ／ `pf` ／ `freq` ／ `thd` ／ `kwh` ／ `kwhpos` ／ `kwhneg`

中文名(例如 `電流`)只有在**剛好等於某個自訂暫存器的名稱**時才解得出來。型號模板設備(Finder 6M、SF965 等)沒有自訂暫存器,於是解析失敗 → **靜默回** `0.0`。

這會產生最危險的假 PASS:訊息每 N 秒準時抵達、JSON 格式正確、topic 正確 —— 唯獨值是憑空捏造的 `0.00`。實測對照:

```
${rs485.5.電流}     → {"current":0.00}  {"current":0.00}  {"current":0.00}
${rs485.1.current} → {"current":0.31}  {"current":0.30}  {"current":0.30}
```

判準:真實量測會微幅跳動,恆定的 `0.00` 幾乎必然是變數沒解出來。看到整排 `0.00` 時,先確認欄位名與 Slave ID,不要當成「現場真的沒電流」。| 🔁 變更時才發 | **不勾** |

變數名稱可按介面上的「📌 可用變數參考」展開查;填好後按「👁️ 預覽填充結果」,**看到真實數值才存** —— 預覽是空的就代表變數名錯了,存下去會發出空值。

通過:60 秒內約 12 筆,即使電流沒變也照發。**沒過**:完全收不到 → 通道沒啟用,或主題字串大小寫不符。

B2 — 關掉重複資料(持續 × 不重複) ｜ 出廠對應 `CV.4`

目的:業主抱怨資料量太大。值沒變就不要送,資料量從「每 5 秒一筆」降到「有變化才一筆」。

設定:同一通道 → **勾選「🔁 變更時才發」** → 儲存。

⚠️ **類比量測(電流、電壓、溫度)光勾這個不會變成 0 筆 —— 必須同時設死區。**

這類訊號永遠在微幅跳動,對韌體來說「值一直在變」,所以每次檢查都算變更、照發不誤。死區的作法就是在模板變數後加小數位修飾詞:`${rs485.1.current:1}`。

實測(Finder 6M 電流環,實際電流約 0.3A,間隔 5 秒):

設定	65~70 秒內筆數
不勾「變更時才發」	~13 筆(每 5 秒必發)
勾了,模板 `${rs485.1.current}` (2 位小數)	**8 筆** — 值在 0.30/0.31/0.32 之間跳,擋不掉
勾了 + **`${rs485.1.current:1}`** (1 位小數)	**0 筆** ✅

所以**開關型的數位訊號**(計數、狀態)勾了就夠;**類比量測**一定要配死區。先勾了發現還在發,不是壞掉,是還沒設死區。

通過:設好死區後,值不變的 60 秒內 **0 筆**;讓負載變動 → **立刻 1 筆**。**沒過**:仍持續收到相同值 → 先確認①勾的是這一個通道(這是**逐通道**設定)、②模板有沒有加小數位修飾詞。

B3 — ★ 證明「重複資料能再回來」(可逆性驗證)

目的:業主真正擔心的不是「現在會不會重複」,而是**「這件事到底歸不歸我控制」。**只證明「勾了就不重複」不夠 —— 那可能只是剛好值沒變。必須證明**取消勾選後重複會確實回來**,才能說明這個開關真的有效、而且是唯一的控制點。

設定:取消勾選 → 觀察 60 秒 → 再勾回來 → 再觀察 60 秒。

通過:取消 → **重複資料立刻回來**(約 12 筆/分);勾回 → **再度歸零**。來回兩次都符合。**沒過**:取消勾選後仍然沒有資料 → 那 B2 的「0 筆」可能根本不是勾選造成的,而是通道停掉了。**這一步就是為了排除這種假通過。**

B4 前置 — 先建立閘門，否則送訊號不會有任何反應

發到 `cmd/signal/1` 只是把數值寫進韌體的一個暫存位置。**要讓它真的當閘門，必須先綁定：**

1. `配置頁` → 📡 **輸入群組** → **＋新增** → 名稱「開工閘門」→ 新增來源 → 類別選 📨 **MQTT 命令訊號 (cmd/signal)** → 選 `cmd/signal/1` → 比較條件 `≥ 0.5` → 儲存

> ⚠️ **比較條件一定要用 `≥ 0.5`，不要用 `= 1`**。兩者在只送 0/1 時看起來一樣，但 **B5 會送 `5`、`15` 這種秒數** —— 用 `= 1` 的話那些值不滿足條件，閘門直接關閉，B5 會收到 0 筆而看起來像功能壞掉。走查時實際踩到過。
>
> 2. 回到 `配置頁` → **TCP 通道** → 點開該 Converter 通道 → 🔒 **門控信號** 選「開工閘門」→ 儲存

> **沒做這一步的話**：門控欄位維持「(無門控，持續推送)」，你送 `1` 送 `0` 都不會有任何差別 —— 而且沒有任何錯誤訊息。這是 B4／B5／B6 全部失效的最常見原因。

B4 — MQTT Trigger 啟動 Converter（啟動式 × 連續） ｜ 出廠對應 `CV.3`

> **目的**：離峰時段不佔頻寬、不產生無用資料。上位系統開工時送一個訊號才開始回報，收工再送一個就停。

macOS / Linux | Windows

```
mq_pub -t "$PREFIX/cmd/signal/1" -m '0'    # 閘門關
# 觀察 30 秒 → 應為 0 筆

mq_pub -t "$PREFIX/cmd/signal/1" -m '1'    # 閘門開
# 觀察 60 秒 → 應照設定間隔持續收到
```

> **沒過**：閘門關著仍在發 → 閘門來源綁錯。

B4b — 啟動式 × 不重複（兩個條件是「且」的關係） ｜ 出廠對應 `CV.4`

> **目的**：這是**計數器最該用的組態**。閘門開著時，每發生一次計數就送一筆；停機（閘門關）或沒有新計數時，完全不送。資料量剛好等於「該回報的事件數」。

設定：在 B4 的閘門仍綁著的情況下，把該通道的 🔁 **變更時才發** 勾起來。

```bash
mq_pub -t "$PREFIX/cmd/signal/1" -m '1'     # 閘門開著
# 不要觸發現場設備，觀察 60 秒
```

> **通過**：閘門開著但值沒變 → **0 筆**；現場觸發一次讓值改變 → **立刻 1 筆**；送 `0` 關閉閘門後，再怎麼觸發都 **0 筆**。**沒過**：閘門關著卻仍在發 → 門控沒綁上（回 B4 前置）。值沒變卻一直發 → 勾選沒生效，或模板沒限小數位（見情境 C）。

B5 — 用訊號內容直接調整回報頻率 ｜ 出廠對應 `WI-143`

> **目的**：不同工單需要不同取樣密度。上位系統送數字就能改頻率，不必派人進設定頁。

macOS / Linux | Windows

```
mq_pub -t "$PREFIX/cmd/signal/1" -m '5'     # 每 5 秒一筆
mq_pub -t "$PREFIX/cmd/signal/1" -m '0'     # 停止
```

> **沒過**：此功能需閘門來源為 MQTT 型別。不使用則標「不適用」。

B6 — ★ 重新觸發要有反應（關再開要補發） ｜ 出廠對應 `CV.4`

> **目的**：操作員收工關掉、隔天開工再打開，若值剛好沒變就**什麼都收不到** —— 他會以為系統壞了。所以重新打開時必須補一筆目前值。

macOS / Linux | Windows

```
mq_pub -t "$PREFIX/cmd/signal/1" -m '0'    # 關
sleep 10
mq_pub -t "$PREFIX/cmd/signal/1" -m '1'    # 再開（期間不要動負載）
```

> **通過**：重新打開後**立刻補發一筆**目前值，即使值完全沒變。**沒過**：不補發 → **現場最常見的抱怨來源**。

CT.1 — 計數器：一次計數 = 一筆訊息

> **目的**：計數值只有事件發生時才變，勾「🔁 變更時才發」後資料量剛好等於事件量，對帳最直觀。不勾的話停機期間會把同一個數字重發上千遍。

> **通過**：現場觸發一次 → 面板 +1 → 網頁 +1 → 訂閱端**剛好 1 筆**；停止觸發後 60 秒 → **0 筆**。**沒過的分辨**：面板有跳、網頁沒跳 → 型號模板或 Slave ID；網頁有跳、訂閱沒收到 → Converter 通道設定。

情境 C：本站設備該用哪一種組態

上面四格不是選擇題，是**依設備特性對號入座**。

設備	建議組態	為什麼
計數器	**B4b**（有閘門）或 **B2**（無閘門）——兩者都**務必勾「🔁變更時才發」**	計數值**只有事件發生時才變**。勾選後「一次計數 = 一筆訊息」，資料量等於事件量，對帳最直觀。不勾的話，停機期間會把同一個數字重發上千遍
電流環	B2 持續 × 不重複 **+ JSON 模板限小數位**	電流是連續量，永遠有微小跳動。**Converter 沒有「死區」欄位** —— 要壓掉雜訊是在 JSON 模板的變數後面限小數位：`${rs485.1.current:1}` 代表只取到小數一位，跳動小於 0.1A 就算沒變、不會發（`:0` 為整數）。**欄位名同樣必須用英文** —— 見 B1 的警告
班別統計	**B4** 啟動式 × 連續	開工送閘門開、收工送閘門關，離峰不佔頻寬

> **編號對照**：`B4` 是「閘門 × **不勾**變更時才發」、`B4b` 是「閘門 × **勾選**」。`B3` 不是象限，它是**可逆性驗證**（見情境 B）。挑組態時看的是象限編號，別把 B3 當組態。

> ⚠️ **電流環最容易出錯的地方。** 業主若說「勾了還是一直有資料」，十之八九不是勾選失效，而是**模板沒限小數位** —— 量測雜訊本身就一直讓字串改變，於是每次檢查都判定「值有變」。**Converter 判斷「有沒有變」是比對整包填好的 JSON 字串**，不是數值容差，所以限小數位才是正解。請當場觀察實際跳動幅度再決定取幾位。

情境 D：設定值 —— 三關保證

設定「畫面上有」不等於「真的存進去了」。每一項設定都要過三關。

編號	關卡	通過	出廠對應
D1	存檔後**立即**讀回（不重新整理）	顯示值與輸入一致	ST.4
D2	重新整理頁面（F5）	值仍正確 —— 證明值來自設備，不是畫面殘留	2A–2F
D3	**重開機後**再看一次	值仍正確。**過了這關才算真的存進去**	ST.5・ST.13
D4	逐項清點全部設定走完 D1–D3	網路／每台 RS485／量程／MQTT／**每個通道的「🔁 變更時才發」勾選**／規則全數通過	—

> D1 的重點：**不要相信「儲存成功」的提示，要親眼看到值。**回報成功但值沒進去，是最難察覺的失敗形式。

三關的指令：

macOS / Linux | Windows

```
# D1 / D2 —— 存檔後讀回（D2 在瀏覽器按 F5 對照）
curl -s -H "Authorization: Bearer $TOK" $DUT/api/config | python3 -c "
import sys,json
d=json.load(sys.stdin)
print('RS485 設備:',[x['name'] for x in d.get('modbusDevices',[]) if x.get('name'
"

# D3 —— 重開機後再跑一次上面那條，逐項比對
curl -s -X POST -H "Authorization: Bearer $TOK" $DUT/api/system/reboot
```

> ⚠️ **設備清單在 `/api/config` 的 `modbusDevices`，不是 `/api/modbus-devices`。**後者回的是匯流排狀態（mode／baudRate／…），沒有設備清單 —— 用錯會看到「0 台」而誤判設定不見了。

自動化整批驗證

07

🤖 先跑 `acceptance_run.py` —— 人工驗收之前的前置

```bash
python3 scripts/acceptance_run.py --dut http://<設備IP> --token <admin token>
python3 scripts/acceptance_run.py --skip-reboot     # 不想讓它重開設備時
```

這支把本文件**所有可自動化的測項**串成**一次不中斷的執行**: 階段 0 / 2B.2 / 3 / 4 / 5 / 6、情境 M(M.1–M.4)、情境 B(B1–B6)、情境 D(D1–D3)、階段 8.1。**任何一項失敗就整體失敗**,而且它**不做任何修復** —— 工作是如實回報,不是把問題藏起來。

素材一律 `AC-` 前綴、測完自清,結束前會驗證設備回到起始狀態。約 13 分鐘。

為什麼要有它:2026-08-26 的走查是散在幾十次互動裡邊跑邊修完成的, 記錄裡還有「先失敗後修好」的項目 —— 那證明不了「這套流程能不中斷地走完」。業主人工驗收之前,必須先有一次**從頭到尾、單一韌體版本、沒有中途修東西**的乾淨通過。

它不涵蓋(本質上需要人):0.1 供電接地目視、階段 R 歸零(破壞性)、 階段 1 空機燒拓荒包、情境 A.1 實體撥開關、5.4 現場觸發計數器、 階段 7 實體斷電、8.2 拍照、8.3 簽名。這些仍要照本文件人工執行。

情境逐項驗完後，跑一次自動化閘門收尾：

這份驗收單與出廠測試是對齊的。每個情境的「出廠對應」欄位指向出廠測試規範 (內部文件 `docs/testing/qa-factory-test.md`)的測項編號(`CV.*` / `EN.*` / `ST.*`)—— 兩邊看的是同一組行為,不是兩套標準。

本走查(2026-08-25/26)挖到的缺陷,已固化成 `npm run test:regression` (`RG.1` – `RG.7`)。**2026-08-27 起七項全數綠燈**,已併進 `test:gate:full` 。對應的產品缺陷(WI-199/200/202/203/205/206)都已修進韌體 6.0.74。

`RG.5` 會**重開設備**,預設略過;要跑帶 `RG_ALLOW_REBOOT=1` 。已於 2026-08-27 單獨驗過。

⚠️ 順帶提醒:當時實驗證明過,舊版閘門在缺陷存在時**仍然全綠** (注入 WI-202 的壞值讓整份 TCP IO 設定在網頁上消失, `factory-pages` 依舊 6/6 通過)。**閘門全綠不等於沒問題** —— 這也是為什麼要有人照著這份單子實際走一遍。

macOS / Linux | Windows

```
cd <專案>/web
EXPECT_FW=<版本> DUT_URL=$DUT ADMIN_TOKEN=$TOK npm run test:gate
```

35 項,約 2 分鐘。**全數通過才算通過。**

⚠️ **在客戶現場跑之前先讀這段:**

- `ST.14` **預設期待「Finder slave 1 + 1 個擴充模組」**(那是我們實驗室的組態)。本站硬體不同的話這項一定會紅,**那不是設備的問題**。請改帶本站的實際組態: `EXPECT_RS485='<slaveId>:<型號>:<名稱>' EXPECT_EXPANSIONS=<數量>` ,或先跳過 ST.14 再單獨人工確認。
- 它**不是嚴格唯讀** —— 過程中會寫入 `deviceName` 再還原。正常跑完不留痕跡,但**中途 Ctrl-C 中斷會在設備上留下測試值**,中斷後請回頭確認設備名稱是否正確。

08

階段 7：斷電復原 —— 業主最該在意的一段

現場一定會停電。這一段驗的是「電回來以後，它會不會自己回到工作狀態」，不需要任何人到場。

步驟	動作	通過	沒過
7.1	**直接切掉電源**（不要用軟體重開），30 秒後復電	自行開機並回到正常心跳	無法自行開機 → 停止驗收並回報
7.2	逐項確認：網路、RS485 清單、規則、MQTT 上行	**全部與斷電前一致**，MQTT 自行恢復，無需人工介入	任一項不見 → **不可交付**。代表設定沒真正落盤，現場停電一次就要重設一次
7.3	確認「🔁 變更時才發」勾選仍在	逐通道展開確認，狀態不變	勾選跑掉 = 重複資料會再度出現。**這一項要單獨檢查**，從資料面要等好幾分鐘才看得出來

「MQTT 自行恢復」不能用「送一個命令看它動不動」草率驗過。 走查時實際踩到: 送 `cmd/signal/1 = 1` 想看燈亮 —— 但燈**送之前就已經是亮的**(WI-161 會在重開後 還原閘門狀態,規則重新觸發),送完還是亮,這個測試什麼都沒證明。

正確做法是製造狀態轉換,而且雙向:

```bash
mq_pub -t "$PREFIX/cmd/signal/1" -m '0'   # 觀察輸出應【關閉】
mq_pub -t "$PREFIX/cmd/signal/1" -m '1'   # 觀察輸出應【開啟】
```

出向則要看到**新鮮**訊息:訂閱 `$PREFIX/#`,連線瞬間湧入的那批是 broker 的 retained 舊訊息,**不算數**;要等到幾秒後才抵達的那一筆(例如 `cmd/_ping`)才證明 設備現在真的在發。走查實測: 連線當下 6 筆 retained,16 秒後才收到一筆現發的 ping。

09

階段 8：結案

步驟	動作	通過
8.1	按「立即推送」備份最終設定	推送成功，**且版本歷史看得到這一筆**（同 4.3，要再確認一次）
8.2	拍照存證：銘牌、接線、總覽頁	與本單一併歸檔。日後遠端支援省下大量往返確認
8.3	與業主逐項核對、雙方簽名	全部勾選，或未通過項目已書面列出並取得同意

10

現場排錯速查

現象	最可能的原因	先做這個
設備開出 Wi-Fi 熱點	Wi-Fi 帳密錯誤或訊號不足	連上熱點重設帳密
網頁排版跑掉、很陽春	網頁資源未完整燒入	開 `/rescue` 確認，回報後重燒
版本歷史顯示 Unauthorized	Cloud Token 空白或錯誤	重填 Cloud Token
掃描看得到、總覽沒數字	設備未加入設定，或暫存器未設	查 `/api/config` 的 `modbusDevices`
畫面所有數值都空白	回應不是合法 JSON	用 5.3 的指令看 Python 會不會拋錯
RS485 某台讀到 0	接線／Slave ID／鮑率／該台沒電	依此順序逐項排除，並用 `/api/modbus/probe`
加了新設備，舊的讀不到	Slave ID 撞號	逐台改成不重複的 ID
類比輸入固定 0	4–20mA 迴路斷開	迴路是串聯，逐點量測找斷點
MQTT 一直收到相同的值	該通道「🔁 變更時才發」沒勾	逐通道檢查勾選
重開後設定不見	設定未落盤 —— 嚴重	**停止使用並回報**

11

文件修訂紀錄

日期	版本	變更
2026-08-21	1.0	初版。涵蓋歸零三級、安裝設定、情境驗收(I/O 互動、MQTT Converter 2×2 矩陣、設定值三關)、逐步 CLI 指令與現場排錯速查。
2026-08-25	1.1	**首次實機照著走過一遍**(8310 `.250` + 8320 `.46`)後的修正,四項都是假 PASS 或指錯位址:① R.4 的「TCP 通道」永遠印 0(`config.tcp` 是監聽埠不是清單,且型別是 dict)→ 改查 `/api/tcpio`。② R.4 通過條件寫了「規則」卻沒查 → 補 `/api/rules`、`/api/signals`、`/api/actions`。③ R.4 寫死 AP 位址 `192.168.3.1`,但 8310 不會開熱點 → 改用 `$DUT` 並附位址對照表。④ R.2／4.3 的「確認雲端收到」打設備端 `/api/configserver`,那條永遠不吐清單 → 改打雲端 `/api/devices/$UID_DEV/snapshots`,env 範本補 `CLOUD`／`CLOUD_TOK`。另修 1.5 心跳燈顏色:原寫「USER LED(綠燈)」,實際 8320 是藍燈、8310 是琥珀(紅+綠)。韌體基線 6.0.67 → 6.0.68。
2026-08-25	1.2	**實際執行破壞性路徑(L2 恢復原廠 → 雲端還原)後補寫**,在 8320(`.46`,fw 6.0.68)實機跑完整循環。① 新增 §R.3 的 🛑 前置警告與 §R.5 還原步驟 —— 恢復原廠會**連 Cloud Token 一起清掉**,而設備 token 為空時不送授權標頭、雲端回 401,於是「雲端顯示還原成功、設備什麼都沒做、雙方都沒有錯誤提示」(已開卡 WI-194)。必須先填回 Cloud Token 才能還原。② 補上 `pendingCommand` 是一次性的 —— 失敗那次已被消耗,要重新觸發。③ 補上以 `/api/log` 查證還原是否真的在跑的方法,並註明該日誌是環形緩衝會被洗掉。實測數據:歸零後 65 秒回線;還原排入後約 2 分鐘取件,下載+重開後約 20 秒設定回來;規則 4→0→4、訊號 2→0→2、動作 3→0→3、RS485 1→0→1、SSID 有→空→有。
2026-08-25	1.3	修 WI-193/WI-194 並在 v6.0.69 實機驗證後補寫,又抓到兩件文件沒寫的事。① **歸零後打開網頁只剩「資產救援」頁** —— L2 格式化整個 QSPI,連 `index.html`／`app.js`／`styles.css`／語系／`log.html`／`workflow.html` 一起清掉;API 全正常但沒有可用 UI,交機流程走到 §2C 上傳 `.mesb` 才會恢復,而**雲端還原只還設定、不還資產**。已補進 §R.4。② **不要「剛改完設定就馬上觸發還原」** —— rollback 是把快照寫進雲端設定檔再叫設備下載,而設備 autoPush 會把它當下的設定推上同一個檔案;實測填完 Cloud Token 後 12 秒設備推送,還原被覆蓋,看起來「成功但沒變」。§R.5 補上「等 `needsPush` 變回 False 再觸發」與步驟重編號(已開卡 WI-195)。韌體基線 6.0.68 → 6.0.69。

日期	版本	變更
2026-08-25	1.4	走查第二部(情境 M)後改寫 **M.4** —— 原文承諾了一個產品沒有的能力。原本教人「來源選 Parser 通道 → 比較條件設 `> 30`」並宣稱「這證明整條 MQTT → 抽值 → 門檻 → 輸出 是通的」,但規則引擎讀 TCP 通道時**只讀布林狀態(值 ≠ 0),不比較數值本身**,韌體會直接回 400 拒絕存檔(WI-184 守衛)。改寫為實際可行的 `= 1`(非零)語意,並寫明「溫度 > 80 就報警」這類需求走這條路做不到。另補兩個會讓人白忙的細節:來源的 `index` 是 `channelId` 不是陣列位置;來源型別要用 `7` 不要用 `9`(型別 9 存得進去、HTTP 200,但規則引擎沒有處理它,實測送 25.5/35.0/99.9 訊號三次都是 False —— 已開卡 WI-196)。M.1／M.2／M.3 原文照抄實測全部通過。
2026-08-25	1.5	**實機走完第二部情境 A(A1／A2／A3 全數 PASS)** 後的修正。① **A.0-1 補上「觸發類型」整段** —— 原文完全沒提到這個欄位,但它決定 A1 後半段成立與否:預設 `Always` 每個 loop 都驅動輸出;而「按鈕**按下**」的直覺選擇 `Rising` 會讓 `信號為 False` 的規則**永遠不執行**(韌體 `main.cpp` inline 規則引擎 `if (rule->triggerOnTrue != signalValue) continue;`),A1 的「解除→復歸」必定失敗。正解是 `Change`,已補完整對照表與韌體依據,並加註脈衝式輸入(觸控鈕)的例外用法。② **A.0 要建兩個輸入群組** —— A1 需實體 DI、A2 需 MQTT 命令,原文只教建一個。③ **按鈕名稱更正** —— 文件寫「新增來源」,實際介面是「**+ 加設備**」;新增群組是右上角 `+`。④ **A3 補上假 FAIL 陷阱** —— `Change` 觸發下,若停用期間已送過觸發值,重新啟用後再送相同值不構成變化、輸出不動,那不是「重新啟用失敗」;必須先把訊號送回原值再觸發(實測踩到)。⑤ **補 MQTT topic 前綴警告** —— 設定頁顯示 `mes/gateway/001`,實際收發用的是 24 碼 UID,照設定頁發命令會石沉大海。⑥ **加註不可連續快速新增** —— 連續建立會讓前一筆被無聲覆蓋且兩次都回成功(WI-199)。走查另開三張卡:WI-199(快速連續存檔 index 撞號覆蓋)、WI-200(`/api/io/do` 碰不到擴充輸出)、WI-201(通道改名無批次化、部分靜默失敗)。
2026-08-26	1.6	**實機走完第二部情境 B(B1／B2／B3／B4／B4b 全數 PASS)** 後的修正。① **B1 的 JSON 模板範例是錯的,而且會造成最危險的假 PASS** —— 原文 `${rs485.5.電流}` 有兩處錯:欄位名用中文、Slave ID 用 `5`。韌體 `resolveRs485Field()` 只認英文欄位名(`voltage`／`current`／`power`／`reactive`／`apparent`／`pf`／`freq`／`thd`／`kwh`／`kwhpos`／`kwhneg`),中文名僅在剛好等於某個自訂暫存器名稱時才解得出來;型號模板設備(Finder 6M、SF965)沒有自訂暫存器,於是**解析失敗靜默回 `0.0`**。實測對照:照文件抄 → 每 5 秒準時收到 `{"current":0.00}`,訊息有來、格式正確、間隔正確,**照原判準會直接打勾 PASS**,但值是憑空捏造的;改成 `${rs485.1.current}` → `0.31/0.30/0.30`。已補判準:真實量測會微幅跳動,整排恆定 `0.00` 幾乎必然是變數沒解出來。情境 C 的同款範例一併修正。② **B2 補「類比量測必須配死區」** —— 原文只在「沒過」欄提了一句,實際上光勾「變更時才發」對電流這種訊號**達不到 0 筆判準**。補上

日期	版本	變更
		實測三段對照(13 筆 → 8 筆 → 0 筆),並說明數位訊號勾了就夠、類比一定要加 `:N` 小數位修飾詞。走查另開一張卡 WI-202(JSON 字串欄位普遍未跳脫,一個引號讓整頁設定人間蒸發)。
2026-08-26	1.7	**實機走完情境 C 與情境 D**。① **情境 C 兩處象限編號錯誤** —— 「計數器 → B4 或 B2(務必勾變更時才發)」中的 `B4` 是「閘門 × **不勾**」那一格,勾了就是 `B4b`,已改為「B4b(有閘門)或 B2(無閘門)」;「班別統計 → **B3** 啟動式 × 連續」的 `B3` 根本不是象限(它是可逆性驗證,文件自己在情境 B 開頭就寫明「B3 不在這張表裡」),已改為 **B4**。兩處都會把現場工程師導向錯誤組態。另補「編號對照」提示行。② **情境 D3 實測通過並抓到一項漂移** —— 逐項清點 40 個項目(網路／MQTT／主機與擴充通道名稱／RS485 設備／4 個輸入群組／6 個輸出群組／8 條規則／2 個 TCP 通道／Converter 模板)後 `POST /api/system/reboot`,設備 9 秒回來(bootCount 184→185, `SOFTWARE/api-reboot`),**39 項完全一致**,唯一差異是 `converterTemplateLen` 32→30,而模板內容前後完全相同 → 已開卡 **WI-203**(存模板不會把長度刷進 flash;不影響發佈,但可能讓跨機複製靜默遺失換算公式)。D1／D2 亦通過(每次存檔立即打 API 對照;整頁重載後群組、規則綁定、通道名稱全數正確)。
2026-08-26	1.8	**實機走完階段 7(斷電復原)與 8.1**。測法如實記載:以**長按 reset**完成(據現場說明其行為等同斷電),**非拔除電源**;冷啟動路徑(乙太 PHY／WiFi 模組真正斷電)因此**未涵蓋**,該台家族史上有冷開機競爭問題,建議日後補一次真正斷電。結果:**7.1 PASS** 設備自行開機回到原 IP, `bootCount` 185→186, `resetReason=UNEXPECTED` ／ `bootDetail` 空 —— 正是 BKPSRAM intent marker 的預期行為(硬體 reset 不留字條),與 D3 軟體重開的 `SOFTWARE/api-reboot` 清楚分得出來。**7.2 PASS** 逐項清點 40 項(網路／MQTT／主機與擴充通道名稱／RS485 設備／4 個輸入群組／6 個輸出群組／8 條規則／2 個 TCP 通道／Converter 模板)**完全一致,無一走失**; `converterTemplateLen`(WI-203)本次未再漂移,因 D3 後記憶體與 flash 已收斂為同值。**7.3 PASS** 逐通道展開確認勾選狀態不變。**8.1 PASS** 觸發推送後 11 秒新快照出現於雲端版本歷史(判準打雲端端點,非設備端)。文件新增一段警告:「MQTT 自行恢復」必須用**狀態轉換**雙向驗證,且出向要區分 retained 舊訊息與現發訊息 —— 走查時實際踩到「燈本來就亮著,送 1 還是亮,證明不了任何事」的假通過。

日期	版本	變更
2026-08-26	1.9	**與出廠測試對齊**。① 在「自動化整批驗證」補上兩份文件的關係說明:驗收單的「出廠對應」欄位指向出廠規範的測項編號,兩邊是同一組行為。② 註明本次走查的五個缺陷已固化成 `npm run test:regression`(RG.1–RG.5,出廠規範 v2.7),**目前預期紅燈**、修好前不列入出貨門檻。③ 補一句實驗結論:**閘門全綠不等於沒問題** —— 注入 WI-202 的壞值(整份 TCP IO 設定在網頁上消失)後,舊版 `factory-pages` 依舊 6/6 通過,該盲點已於 v2.7 修正(`2B` 改為比對設備通道數與畫面列出數)。
2026-08-26	2.0	**由 Claude 主導完整重跑一次全流程(韌體 v6.0.73)**,涵蓋階段 0/2/3/4/5/6、情境 A/M/B/C/D、階段 7(以 reset)、8.1。本次挖到一個**高嚴重性缺陷**與三處文件缺口。① **WI-206:Parser「值類型」下拉三個選項全部對錯韌體列舉** —— 韌體 `ParserValueType` 是 `BOOL=0/NUMBER=1/STRING=2`,而 UI 是 `0=數值/1=字串/2=布林`。選「數值」實際會得到**布林**,任何非零數字都變成 `1.0`,不會報錯。**預設值正好落在錯的那格**,不動下拉的人一定中獎。走 M.3 時 `currentValue` 顯示 1.0 而非 25.5 抓到(該測項的判準「不是 0、不是 1」正好擋得住)。已修前端 option value 並補 `RG.7` 回歸(先紅後綠實測)。② **B4 前置補「為何用 `≥ 0.5` 不用 `= 1`」** —— 只送 0/1 時兩者看似相同,但 B5 會送 `5` / `15` 這種秒數,用 `= 1` 會讓閘門關閉、B5 收到 0 筆而看起來像功能壞掉(走查實際踩到)。③ **4.2/4.3 補「先做一次實際設定變更再推送」** —— 雲端對內容相同的推送會去重,設定沒變時推送回成功但不產生新快照,看起來像失敗(誤判過一次);並補「設備端雲端設定要查 `/api/configserver`, `/api/config` 裡沒有這個區塊」。全流程結果:出廠閘門 35/35、回歸 7/7、情境 A/M/B/C/D 全數 PASS(B1–B6 含首次執行的 B5/B6)、D3 與階段 7 各自 40 項逐項比對完全一致。
2026-08-26	2.1	**新增 `scripts/acceptance_run.py`** —— **把全流程串成一次不中斷的執行**。v2.0 的走查雖然全數 PASS,但那是散在幾十次互動裡邊跑邊修完成的,記錄裡還有「先失敗後修好」的項目 —— 證明不了「這套流程能不中斷地走完」。這支涵蓋所有可自動化的測項(階段 0/2B.2/3/4/5/6、情境 M、情境 B、情境 D、階段 8.1),任何一項失敗就整體失敗,且**不做任何修復**。實測需要六輪才跑通,每一輪的中斷都是真實的環境條件,已一併寫進本文件:① **掃描期間設備會拒絕 HTTP 連線**(`Connection reset by peer`)—— 它忙著試 4 種鮑率 × 3 種同位,單執行緒的 HTTP server 排不上。這是正常的、`bootCount` 不變,但文件先前完全沒提,照「每 10 秒輪詢」的人會誤判成當機。已補明「拒連就當還在掃,每 15 秒問一次,最長等 5 分鐘」,並註明回應的鍵是 `devices` 不是 `results`。② **對外推送偶發 TIMEOUT** —— 單次逾時不該判定驗收失敗,設備自己會在下個 autoPush 週期重試;測項改為允許重試並如實回報試了幾次。③ 另修正三處**測試方法本身**的問題(非產品):M.4 的 index 要用附加位置否則會先撞 `Invalid index` 而測不到 WI-184 守衛;B6 的訂閱必須**先於**開閘門(補發是立刻發生的,`pub` 之後才 `subscribe` 會落在 TLS 握手空窗裡收不到,導致同一項一輪綠一輪紅);讀取要對

日期	版本	變更
		「HTTP 200 + 截斷的 body」重試,只重試連線錯誤不夠。最終結果:**28 PASS • 0 FAIL • 0 SKIP • 12.9 分鐘,一口氣跑完無中斷**(韌體 v6.0.74)。
2026-08-27	2.2	**補上「給人工驗收者:今天從這裡開始」入口段**,放在文件最前面。原因:自動化段落埋在第 1442 行,人工驗收者要翻過 1400 行才知道該從哪切入。新段落把作業拆成兩步——先跑 `acceptance_run.py` 取得綠燈前置,再逐項走機器碰不到的 7 項(附「桌上這台可否執行」欄,避免把「本站無此硬體」誤記成漏測)。並把**階段 7 斷電**與**WI-197 網路故障切換**合併成一次操作程序(拔網路線→ `net-failover` ／插回→ `net-recovery` ／拔電源→冷啟動),附 `watch` 指令與逐步預期值。另**修正檔頭**:版本 1.4→2.2、日期 2026-08-25→2026-08-27、韌體基線 6.0.69→6.0.74——修訂紀錄早已走到 2.1,檔頭一直沒跟上。
2026-08-27	2.3	**修正過時說法 + 補一則畫面提醒**。① 原文寫「RG.1–RG.5 目前預期紅燈、不列入出貨門檻」——那是修好之前的狀態,實際跑過後**七項全綠**(RG.5 另以 `RG_ALLOW_REBOOT=1` 單獨驗過),對應缺陷都已修進 6.0.74,早已併入 `test:gate:full`,照舊文走會誤以為交付物帶著紅燈。② 人工入口補上存檔徽章的說明:改通道名稱時上方跳出的「💾 bH 1」不是錯誤訊息, `bH` 是漏掉的語系鍵(**WI-207**),意思是「還有 1 筆正在存」——數字歸零再離開該頁。
2026-08-27	2.4	**韌體基線 6.0.74 → 6.0.78**。驗收走查期間業主回報「呼吸燈卡住時按鈕沒反應」,追出主迴圈有 **6.8% 的時間完全凍結**(授權門控每 30 秒驗簽 663ms + retained `license/state` 被反覆重送),已修正並實機驗證降到約 0.8%(**WI-208**)。同版修掉存檔徽章顯示「💾 bH N」的語系缺鍵(**WI-207**),本文件的徽章說明同步改回「💾 儲存中 N」。
2026-08-27	2.5	**階段 5.1 的掃描警告大幅補強**。原文只寫「網頁與 API 會連不上」,但實測(v6.0.78)單次掃描把**主迴圈整整壓住 65 秒**(`scan=65202ms`)—— 停住的不只網頁,**實體 I/O 也不會動**:呼吸燈停住、DI 不被輪詢、規則不執行。業主在重跑驗收時正是看到呼吸燈卡住而回報「問題又出現了」,實際上那是掃描,不是 WI-208 的回歸(同一份 log 中 `gate 重驗` 零次)。已補明「不要在掃描期間測按鈕」,並更正掃描範圍為 7 種鮑率 × 同位元 × 32 個 slave ID(原文寫 4 種鮑率)。成本本身另開 **WI-209** 追蹤。
2026-08-27	2.6	**韌體基線 → 6.0.79**。業主回報「呼吸燈大概 115 次會卡一次」——換算約 115 秒,對上雲端心跳的 127 秒週期。查出雲端每次心跳都重送同一則早已被拒的舊授權 token,設備每次都白白驗簽**兩次**(1.33 秒),期間主迴圈完全凍結。已於 **WI-208b** 修正(收訊前先比序號、不新就不驗;既有序號改快取;只有真的存了新 token 才重評門控),並一併修掉

日期	版本	變更
		v6.0.78 引入的 ODR 陷阱(header 裡的 `static` 讓失效機制沒接上,會導致撤銷授權延遲到重開才生效)。實測 544 秒涵蓋 8 次心跳,`net` 段卡頓 **0 次**。
2026-08-28	2.7	**韌體基線 → 6.0.83**。驗收期間業主快速切換實體開關,呼吸燈進入快閃(=MQTT 斷線中,非重開)。追出完整故障鏈:每條規則觸發都發一則通知、邊緣模式不節流 → 灌爆佇列 → 每則 TLS 發送阻塞主迴圈 265~400ms → 連線撐不住 → 重連風暴 → 若持續約 90 秒,self-ping 連兩次無 echo 會讓**設備自我重開**(**WI-210**)。依業主定下的原則「控制是現場的問題,訊息可以晚到」重做發送路徑(**WI-211**):QoS 配合語意(發送 1、通知 0、訂閱 2,皆可設定)、同 topic 合併 + 每 topic 最小發送間隔 1 秒、payload 可選夾帶歷史、每 60 秒重發現況兜底。實測 30 事件 5 秒:重連 4→0 次、最大卡頓 4398→無 >150ms。**DI→DO 的控制路徑完全不經過 MQTT**(`ioManager.writeDO()` 直接寫硬體),本次改動不影響現場控制。

04 開發者 / 整合

出廠測試規範

版本:3.4 | **日期**:2026-08-28 | **韌體版本**:6.0.83(基線) **來源 PRD**:`docs/prd/prd-factory-test.md`(內部文件,未公開) v1.2(結構基線)+ 本文件 v2.1 增量(依 6.0.8 源碼/實機核校) **操作手冊**:guide-user-operation-manual.md

v2.9 修訂摘要(2026-08-26,完整重跑驗收流程後):

- **新增 RG.7**(WI-206:Parser「值類型」下拉的 option value 必須對得上韌體 `ParserValueType`)。該缺陷是走情境 M.3 時抓到的:選「數值」實際會得到布林, 任何非零數字變成 `1.0`。**RG.7 必須斷言 UI 的 option value** —— 只用 API 送 `parserType:1` 驗「韌體會給數值」是抓不到的(韌體本來就對,錯的是前端綁定)。
- **韌體基線 6.0.72 → 6.0.74**(WI-206 + WI-197)。
- 全流程重跑結果:`test:gate` 35/35、`test:regression` 7/7。

> **v2.8 修訂摘要(2026-08-26,六張卡修完並實機驗證)**:

- **`test:regression` 全數通過,已納入 `test:gate:full`** —— WI-199/200/202/203/205 五個缺陷 已修並經「先紅後綠」驗證(修正前紅、修正後綠,同一台設備同一支測試); WI-201 經查是**誤判**(WI-170 的寫入佇列早就處理了),已大幅更正該卡。
- **韌體基線 6.0.69 → 6.0.72**,經 LAN `.mesb` OTA 上機, `test:gate` 35/35。
- **新增 RG.6**(WI-205:被拒絕的規則 POST 不得毀掉既有規則)。
- 順帶修好一個從沒被測到的舊 bug:`modbusTcpWriteDO` 的擴充分支對區域副本取址轉型, **Modbus TCP 寫擴充輸出一直沒有作用**;已與規則引擎統一用 `getExpansionPtr`。

> **v2.7 修訂摘要(2026-08-26,與驗收走查對齊)**:

- **新增 `npm run test:regression`(RG.1–RG.5)** —— 驗收走查挖到五個缺陷, **既有閘門一個都沒攔到**。這五項是它們的回歸,**目前預期紅燈**,修好前不列入出貨門檻。
- **修正一個實驗證明的閘門盲點**:`2B` 原本只斷言「TCP 通道」標題可見, 注入 WI-202 的壞值(整份 TCP IO 設定在網頁上消失)後仍 **6/6 全綠**。已改為比對「設備上有幾個通道」與「畫面上列出幾個」。
- **環境預設值對齊**:`factory-env.js` 的 `DUT_URL` 改 `.77`(有線)、`EXPECT_FW` 改 `6.0.69`; `package.json` 內的舊值一併更新。
- **`ST.2` 改為非破壞性**:原本寫死 `index: 7`,設備規則數 ≥ 8 時會**覆蓋現場既有規則** —— 2026-08-26 實際毀掉一條 `A2-MQTT亮`。已改用「附加位置」(`ruleCount`)。該次紅燈同時揭露韌體缺陷 **WI-205**(回 400 卻仍寫入且不回滾),已補 `RG.6` 回歸。
- **測試設定來源收斂**:先前 6 支 spec 各自寫死 `DUT_URL` 預設(3 支 `.46`、3 支 `.77`), `factory-v6` 的 `EXPECT_FW` 還停在 `6.0.8` —— 文件卻宣稱「換設備只改一個檔」。已全部改為從 `factory-env.js` 取值,那句承諾現在才是真的。
- **已知未處理**:`package.json` 有 4 處明文 broker 密碼(見 WI-204,交付優先暫緩)。

v2.6 修訂摘要(2026-08-20,交付定案版):

- **測項與腳本對齊(文件先前寫錯)**:`test:converter` 是 **5 項 + `afterAll` 收尾**(CV.9 是清理不是測項); `test:enable` 是 **7 項,EN.5(停用 RS485 設備)已從腳本移除** —— 該卡片在畫面上只有「刪除」沒有停用開關, 測一個使用者碰不到的控制項沒有意義;新增 **EN.8(停用 Config Server 總開關)**。
- **新增章節「本批韌性修正:閘門測不到的那些」**:WI-175/180/181/182/183/189 六項, 列出各自「為什麼會壞」與「怎麼驗的」。這張表的用途是**改到那塊之前先讀**,不是測試清單。
- **新增 `npm run test:gate:full`**:唯讀閘門 + 三支會改動設備狀態的套件一次跑完(交機前用)。`test:gate` 維持唯讀語意不變(OTA 後、現場都能安全跑)。
- **本文件已發佈到雲端文件中心** `/docs/factory-test.html`。

v2.1 修訂摘要(2026-07-16,韌體 6.0.4 → 6.0.8 對齊 + 源碼核校修正):

- **韌體基線 6.0.4 → 6.0.8**:全文版號斷言改 `v6.0.8`;新增 v6.0.5~6.0.8 能力。
- **新增自動化套件 `web/tests/factory-v6.spec.js`(13 tests,純 HTTP 唯讀契約)**:Phase 0 機型身分 + Phase 6B 遠端診斷 + Phase 2E.6a MQTT 測試契約 + Phase 2E.6b namespace 欄位; `npm run test:factory:v6`。已對實機 8320(6.0.8)全綠。
- ⚠ **源碼核校修正欄位名(舊文件錯,自動化已抓出)**:

  - `/api/diag` 實際欄位 = `resetReason`(非 `bootReason`)、`bootDetail`、`prevResetReason`、`bootCount`、`prevUptimeSec`(非 `prevUptime`)、`uptimeSec`、`rawRsr`、`fwVersion`。
  - `/api/system` `boardModel` 值 = 字串 `"Opta RS485 (8310)"` / `"Opta WiFi (8320)"`(非 `"8310"` / `"8320"`);`hasWifi` 為 boolean;兩者一致。
  - `mac` 在 **`/api/config`**(非 `/api/system`),格式為冒號大寫(如 `A8:61:0A:50:8B:61`);對外 topic 12-hex 身分 = 去冒號轉小寫。
- **SF965 原生解碼(6.0.8)**:H3 新增 `VAR_SF965_PV`(dataType=8)手動驗證列(需 SF965 硬體;PV@addr4 高 3 byte=值、低 byte=小數點)。
- **離線來源不發假 0(6.0.7)**:Phase 4 新增 H3-B3(converter 引用離線 RS485 來源 → 整筆跳過發佈,不送假 0 誤導 MES)。

v2.0 修訂摘要(2026-07-02,韌體 5.9.45 → 6.0.4 對齊):

- **韌體基線 5.9.45 → 6.0.4**:全文版號斷言改 `v6.0.4`;Phase 0 讀 `boardModel` / `hasWifi` / `mac`。
- **機型維度(8310 / 8320)新增**:變體識別從單一「網路變體」擴為 **機型 × 網路** 二維(8310 無 WiFi / 8320 有 WiFi),來源 = OTP `boardInfo().wifi`。新增 **Phase 4 H8 機型感知 UI 過濾 + 心跳燈**(8310 隱藏 WiFi 欄位 + 琥珀燈;8320 露 WiFi + 藍燈)。
- **WI-155 平台化(6.0.0)**:新增 **Phase 2E MQTT 命名空間**(`projectName` + `env` → `{proj}-{env}/gateway/{mac}`,留空 = legacy)與 **MAC 軟體覆寫 + 90s 試用回退** 驗證。
- **遠端診斷(6.0.1)**:新增 **Phase 6B 遠端維運健檢**—— `/api/log` 網頁 Serial Log(`/log.html`)+ `/api/diag` 開機原因(BKPSRAM intent 標記)+ 密碼遮罩不外洩。
- **RS485 slaveId 持久化(6.0.2)**:H3 新增 **同型號多台防串台** 驗證(各設不同 slaveId ≤32,重開機後不退回陣列位置、不串台)。
- **MQTT 測試連線修正(6.0.4)**:Phase 2E MQTT 測試改 deferred(POST 202 → GET `/api/mqtt/test` 輪詢)+ 對已連線同 broker 短路判定;密碼留空 = 用已存密碼。
- **授權撤銷門控(WI-145)**:Phase 7 補「撤銷後 TCPIO/MQTT 擋、DIO/管理不擋」的門控驗證。
- **MQTT TLS 出廠預設**:`tlsInsecure=true`(業主指定,不驗 CA)——Phase 2E 斷言此為出廠預設,勿改回驗 CA。
- **Web 部署模型**:index.html 編進韌體、app.js 等 6 資產走 QSPI 熱更——出廠燒錄後首開機須確認網頁完整(非裸 HTML)。
- 附錄 A 補 `/api/log`、`/api/diag`、`/api/mqtt/test`、`/api/license/token` 端點。

v1.2 修訂摘要(2026-05-27):

- H3-C 新增 **Step 5 校正引導精靈 chain 驗證**(4 步驟 + 燈號 + 順序強制,v5.9.130+)
- 附錄 A 補 `/api/signals`、`/api/actions`、`/api/converter/template` 端點
- Phase 7 註記 wizard / wdprobe / watchdog 屬**部署佈建非出廠基線**
- 相關:`spec-mqtt-chain-workflow.md`、`guide-calibration-wizard.md`、`guide-failover-resilience.md`

v1.1 修訂摘要:

- 韌體版本對齊:5.9.25 → **5.9.45**
- 治具校正:Expansion 1 → **Expansion 0**(array 從 0 起);YX523R Slave ID 2 → **4**
- 新增 **MQTT broker** fixture(FR-G E2E 必需)與**變體識別**(ethernet/wifi)
- Phase 結構從 0~~5 擴充為 08~~,補上 OTA、E2E 業務流、出貨前清理、結果輸出
- 各步驟加上 PRD FR-* 對應 ID + DOM ID 速查表
- 補上 **EEPROM DIP+斷電** 物理步驟(先前完全漏掉)
- 附錄 C 列 PRD v1.2 已知 UI bug 對工站的影響

01

⚡ 先跑自動化閘門(v2.2 起,任何改動後都必須執行)

任何韌體或網頁改動之後,先跑這一道再談其他。它是唯讀的(不改設備設定),交機前、OTA 後都能安全執行:

```bash
cd web
DUT_URL=http://<設備IP> EXPECT_FW=<版本> npm run test:gate
```

涵蓋 35 項:頁面渲染(2A–2F)、v6 能力契約、穩定性與持久化(ST.*)。**全數通過才算通過** —— 任何一項紅燈都代表出貨品質有缺口。

另有三支會改動設備狀態、不在閘門內、交機前各跑一次:

```bash
npm run test:converter  # ★ MQTT Converter 鏈路(產品主幹),素材自建自清,約 3 分鐘
npm run test:enable     # 啟用/停用機制(⚠ 見下方說明,目前不作為出貨門檻)
npm run test:backup     # ★ 雲端備份/還原閉環(會讓設備重開一次),約 2 分鐘
npm run test:persist    # 真的重開設備,驗設定與統計跨重開存活(約 1 分鐘)
npm run test:factory    # 完整功能測(含 MQTT 佈建,會寫入設定)
```

驗收走查回歸(`npm run test:regression`,RG.1–RG.8)—— 全數通過,已納入出貨門檻

2026-08-25/26 照著驗收單實機走了一遍,挖到五個缺陷。**既有出廠閘門一個都沒攔到**,而且不是「剛好沒測到」,是**測試的形狀不對** —— 既有測項驗的是「東西在不在、頁面開不開」,五個缺陷全部落在「**內容對不對、值有沒有真的寫進去**」。

最直接的證據:把 WI-202 的壞值注回設備(整份 TCP IO 設定在網頁上消失、顯示「尚無 TCP IO 通道」),`factory-pages` **依然 6/6 全綠**。閘門對它完全無感。

```bash
cd web
DUT_URL=http://<設備IP> npm run test:regression
```

測項	對應卡	驗什麼	預設
RG.1	WI-202	字串欄位含 `"` `\` 時,各 GET 端點仍須回**合法 JSON**,且值讀回不得被截斷	執行
RG.2	WI-199	**網頁上**連續快速新增兩條規則,兩條都要在(必須走瀏覽器 —— 缺陷在前端算 index 的邏輯,直接打 API 碰不到)	執行
RG.3	WI-170/WI-201	**網頁上**連續改多個通道名稱,全部都要寫入(守 WI-170 的寫入佇列還在)	執行
RG.4	WI-200	`POST /api/io/do` 要能驅動擴充板輸出	執行
RG.5	WI-203	存 Converter 模板後**立刻重開**,長度須與模板一致	**略過**(需 `RG_ALLOW_REBOOT=1`)
RG.6	WI-205	被拒絕(400)的規則 POST **不得毀掉既有規則**,也不得留下啟用中的死規則	執行
RG.7	WI-206	Parser「值類型」下拉的 **option value 必須對得上韌體列舉**(選「數值」不能拿到布林)—— 斷言 UI 的 option value,不是 API 回值	執行(需 MQTT 憑證)
RG.8	WI-208 / WI-208b	**主迴圈不得被授權處理凍結** —— 兩條斷言:① 安定後 300 秒內 `[WI-208] gate 重驗` 須為 **0 次**;② 同一窗內 `net` 段不得有任何超過 **1 秒**的卡頓。窗長 300 秒是為了蓋過雲端心跳的 127 秒週期(至少 2 次),否則 WI-208b 那條路徑抓不到	執行(約 5 分)

只有 RG.5 預設略過(它會重開設備,要跑帶 `RG_ALLOW_REBOOT=1`;已實測通過)。

⚠ **2026-08-27: `RG.7` 在此之前從來沒有真的執行過。** `package.json` 的 `test:regression` 是**唯一沒帶 MQTT 環境變數**的腳本 (旁邊四個都有),RG.7 因此每次都靜默 `skip` —— 報表顯示「6 passed」, 看起來很健康。已補齊環境變數,現在會真的跑。

教訓: `skipped` 不是中性的。一個永遠跳過的測項,和沒有這個測項是一樣的, 但它在報表上長得像有在守。看回歸結果時要**同時看 skipped 的數量與原因**。

⚠ **2026-08-28: `RG.7` 在整套連跑時偶發假紅燈,已修測試方法。** 單獨跑綠、`RG.5+RG.6+RG.7` 最小組合也綠,只有整套跑會紅 —— 而整套跑到 RG.7 時 設備剛經歷 RG.5 的重開加一輪負載。原本的寫法是「建通道 → 固定等 2.5 秒 → 發一次 → 固定等 5 秒 → 讀一次」。**MQTT 對「訂閱之前就發出的訊息」沒有補送** —— 設備若還沒訂好新通道的 topic,那則訊息就永遠消失,測試必紅。

已改為**週期重發 + 輪詢讀回**(45 秒內每三輪重發一次)。這個教訓 `acceptance_run.py` 早就記過(「通道剛建好時設備可能還沒訂閱」),RG.7 沒套用。

誠實記一筆:那次的錯誤訊息因為輸出被 grep 過濾而遺失,根因是**從程式碼推斷**、不是從錯誤訊息實證。修正本身站得住(移除的是本檔已咬過兩次的同一類問題), 但若日後 RG.7 再紅,不要假設就是這個原因。

教訓:跑長時間測試套件時,不要用 `grep` 過濾輸出 —— 失敗細節只有一次機會保留。

⚠ **RG.2 / RG.3 必須走瀏覽器,不能用 API 並發代替。** 這兩個缺陷都在前端 (`saveRule` 算 index 的邏輯、`saveIoName` 走 WI-170 的寫入佇列), 直接打 API 測到的是我們自己的測試碼,不是產品。初版就是這樣寫的:RG.2 一次就綠(bug 還在)、RG.3 則「測出」一個其實不存在的缺陷(WI-201 因此大幅更正)。

2026-08-27 追加 `RG.8` (WI-208)。 業主人工驗收時回報「藍色呼吸燈會卡住, 卡住的時候按鈕就延遲、甚至沒反應」—— 量出主迴圈有 **6.8% 的時間完全凍結**, 真兇是授權門控每 30 秒重驗一次簽章(663ms)。改為到期驅動後降到約 0.8%。

這一項有實測的鑑別力,不是推論:修正的第一版(v6.0.76)把 `expiresAt==0` 誤當成「1970 年就過期」,快取完全沒生效 —— 同樣的 200 秒觀測窗裡 `gate 重驗` 出現 **8 次**。RG.8 的判準當時會紅。

2026-08-27 更新:七項全綠(RG.5 另以 `RG_ALLOW_REBOOT=1` 單獨驗過)。 這一輪另外修掉兩個**測試自身**的問題,兩個都會產生假紅燈: ① RG.2/RG.3 用固定 4 秒等 `window.nav` 備妥 —— 單獨跑夠,整套跑時設備有負載就不夠 (實測單獨 2 秒即備妥、整套跑時逾時)。已改為輪詢至函式真的存在。② RG.3 用譯文 `/儲存中|Saving/` 判斷寫入佇列是否忙碌,但徽章實際顯示「💾 bH 1」—— `io.saving` 這個 i18n 鍵三個語系檔全缺(**WI-207**), `t()` 查不到時回傳鍵本身, 讓 `|| "儲存中"` 變成死碼。判斷失效 → 提早讀取 → 假紅燈。已改為比對不會被翻譯的 💾 。

2026-08-26 更新:六張卡全部修好,這一組已全綠。 每一項都經過「先紅後綠」—— 修正前該測項紅、修正後綠,同一台設備、同一支測試。韌體 v6.0.72 經 LAN `.mesb` OTA 上機驗證, `test:gate` 35/35、`test:regression` 6/6。**已納入 `test:gate:full`**,現在它真的有在守。

`test:converter` 是最該跑的一支。 業主買的就是「MQTT 進來能驅動輸出、量測值能定時送出去」—— 頁面全部正常但這條鏈斷掉,設備等於沒有用。它會自建 `CV-*` 前綴的素材、測完自己刪光,並以「素材必須清乾淨」作為最後一項斷言。

閘門守著的「已知會壞的那幾類」(逐項對應真實事故)

測項	守的是什麼	來源事故
ST.0 / ST.9	測試期間設備不得重開	功能全對但設備會重開,對業主而言就是壞的
ST.1	連續請求不得連兩次失敗	併發過頭會把閘道器打重開(WI-176)
ST.2 / ST.3	規則不得存成永遠不觸發	下拉未載入就存 → signalId=0(WI-174)
ST.4	設定寫入後讀回一致	「回成功但沒進去」的靜默失敗
ST.5	統計/開機計數有落盤	重開後統計歸零(WI-164)
ST.6	index.html / app.js / 語系檔三者同版	短碼位移 → 全站標籤指到錯字串(WI-172/185b)
ST.7	WiFi 帳密不得為空	無聲清空 → 重開後失聯(WI-169)
ST.8	救援頁可用、主畫面沒退回救援頁	index.html 移到 QSPI 後的發佈陷阱(WI-178)
ST.10	欄位名打錯要回 400	靜默照收 → 整合商拿著 success 找三天(WI-179)
ST.11	TCP 來源的不可能條件要被拒	設了「>50」永遠不成立且無提示(WI-184)
ST.12	改 token 名稱/權限不必連帶換密碼	畫面寫「留空＝不變」但兩端都沒實作(WI-188)
2F	畫面不得殘留 `{UID}` 這類佔位符	該告訴使用者的資訊變成一串符號(WI-185)
ST.13	I/O 名稱改了要能存進去	名稱是現場辨識訊號的唯一依據,存不進去整頁沒用
ST.14	現場硬體與環境設定檔相符	換了設備卻沒更新設定檔 → 這裡先紅,不必等功能測莫名失敗才回頭猜
ST.15	雲端備份設定完整(有 cloudUrl 就要有 cloudToken)	還原舊快照會把 token 一起還原成空 → 備份能寫不能讀,而且**只壞一半**所以難察覺

MQTT Converter 鏈路(`npm run test:converter`,5 項 + 收尾清理)

測項	守的是什麼
CV.0	佈建入向通道 / 輸入群組 / 規則 / 輸出群組(素材自建,不碰現場設定)
CV.1	入向 Parser:MQTT JSON 的數值要進到通道
CV.2	**端對端**:MQTT → 輸入群組 → 規則 → 輸出群組 → 實體 DO
CV.3	出向 Converter:閘門關不發、開了照間隔發
CV.4	onChange:值不變只發一次;**閘門關再開要補發**(v5.9.266 —— 否則使用者重新觸發卻沒反應)
(afterAll)	收尾:本檔建立的素材必須全部清乾淨(留下殘骸就是污染現場設定)

> 這幾項固化的是 2026-08-18 夜間的完整手動驗證。當時結論只存在於對話裡, 下次改動不會有人幫忙擋 —— 這正是「測試樣本要全面覆蓋」的意思。

啟用/停用機制(`npm run test:enable`,7 項)

畫面上到處是啟用/停用的開關,先前**一個都沒測**。這類機制壞掉的方式特別惡劣:「畫面顯示停用、實際還在動」(業主以為關掉了,機器卻自己跑), 或「停用之後再啟用回不來」(現場只能重開機碰運氣)。兩種都不會有錯誤訊息。

測項	守的是什麼
EN.0	佈建 + 基線:鏈路本身要先是通的(否則「停用後不動」毫無意義)
EN.1	停用**規則** → 觸發後輸出不動;重新啟用要回得來
EN.2	停用**輸入群組** → 規則不觸發(signalSlotUsable 的 fail-safe)
EN.3	停用**輸出群組** → 規則觸發但輸出不動
EN.4	停用**TCP 通道** → 不再接收 MQTT(值凍結);重新啟用要收得到
EN.8	停用**Config Server 總開關** → 不得再對外推送(業主關掉雲端同步就必須真的停)
EN.6	停用**token** → 該憑證不得再通過認證(以為撤銷了、其實還能用 = 安全問題)

> ⚠ `test:enable` **目前不列入出貨門檻。**它已經達成目的 —— 2026-08-20 完整跑過 7/7,證實六種停用機制的語意都正確 (停用真的停、重新啟用都回得來)。但**這支測試本身還不夠穩**: 測項之間有順序耦合(前一項的雲端等待會影響後一項的量測)、依賴 MQTT 環境變數(用 `npx playwright -g` 單獨跑會因前綴/密碼未設而失敗)。結論可信,工具待磨。**修穩之前不要拿它當出貨紅綠燈**,否則會訓練人忽略紅燈。| 收尾 | 用 `afterAll` 清素材(**不能寫成最後一個 test**,見下方) |

> 每一項都**同時驗兩個方向**:停用要真的停、重新啟用要回得來。只驗前者的話,「停用之後就再也回不來」這種更難救的缺陷會漏掉。
>
> **收尾一定要用 `afterAll`,不能寫成最後一個 test。**序列模式下只要前面任何一項失敗, 後續 test 就不會執行 → 素材殘留 → 下一輪再建一份同名的 → 名稱查找抓到第一份 → 鏈路指向錯的通道 → 之後每一輪都失敗。2026-08-20 實際踩到:一次失敗滾成兩份素材, 手動清掉才恢復。一律走「真的送 MQTT、真的看實體 DO」,不看欄位存不存得進去(那是 ST.4 的事)。

雲端備份 / 還原閉環(`npm run test:backup`,4 項)

備份/還原是**所有其他測試的安全網** —— 出廠測試會寫入設定、現場調參數也會出錯, 能不能救回來全靠它。它壞掉時最危險的地方是**平常看不出來**:設備→雲端的推送 有自己的授權路徑、照樣 HTTP 200,只有真的要「讀回來」時才發現 401。

測項	守的是什麼
BK.0	雲端設定完整(沒有 token = 能寫不能讀)
BK.1	改設定 → 推送 → 雲端**真的**出現新快照(不是只回 200)
BK.2	還原前一份 → 改動要被還原回去(閉環的關鍵一步)
BK.3	還原後雲端連線不得斷掉(否則還原一次就不能再還原 = 自斷後路)

> 寫這幾項時踩了兩個判準錯誤,值得記下來:
>
> 1. **不能用「快照數量增加」當判準** —— 雲端會裁切舊快照,數量可能不變;要比對最新那份有沒有換人。
> 2. **設定沒變更時推送不會產生新快照,那是正確行為**(內容相同不重複存,實測連推 80 秒都不會多)。所以測試要先真的改一個欄位再推,否則會把正確行為誤判成「回成功但沒發生」。

> **維護原則**:每修一個缺陷,就在這裡加一列。出廠測試的價值來自「它涵蓋了已知會壞的東西」,不是「它有很多項」。

本批韌性修正:閘門測不到的那些(v6.0.41 → 6.0.65)

上面的表格是「有自動化守著」的部分。但這批交付還改了一些**自動化測不到、卻直接影響 現場可靠度**的行為。列在這裡不是為了完整,是因為**下一個人改到這些地方時要知道當初為什麼這樣寫** —— 沒有測試會替他擋。

項目	改了什麼	為什麼會壞	怎麼驗的
WI-175	QSPI 長時間寫入期間持續餵狗 (`loopKeepAlive`)	OTA 撞上背景雲端 TLS → 主迴圈被佔住 → 看門狗砍掉升級	實機:雲端握手後 0.6s 觸發 OTA,守衛等 3765ms 後成功
WI-180/180b	設定存檔原子化:寫暫存 → 驗證 → **確認主檔有效才降級為備份** → rename	舊流程會在主檔還沒驗證前就覆蓋備份,一次斷電兩份全毀	單元級:刻意寫入半截 JSON,重開後仍載入得到有效設定
WI-181/181b	WiFi 帳密改用 RAM 暫存再落盤	`beginWiFi()` 期間 QSPI 未掛載,在那裡讀寫會**永久卡死、只能 DFU**	實機:此路徑曾兩次把 .46 變磚,修正後重開 5 次帳密皆存活
WI-182	設定下載先試地端、再試雲端	地端 config server 不在時整個還原路徑斷掉	實機:拔掉地端來源仍能從雲端取回
WI-183/183b	對外連線硬性總時限 + 雲端卡死自我復原	`connect()` 對黑洞位址可阻塞 **5 分鐘以上**,期間整台設備沒有回應	黑洞位址實測:確認阻塞>5min,並在對抗性輪詢下觸發自我復原
WI-189	分頁切到背景時停掉輪詢與 telemetry	背景分頁持續要資料,排擠掉真正重要的控制迴圈	見 控制延遲特性 的前後對照

> **這張表的維護方式和上面那張不同。** 上面那張是「每修一個缺陷加一列測試」; 這張是「**改到這塊之前先讀這一列**」。如果哪天有辦法替其中一項寫出自動化測試, 就把它搬到上面那張表去 —— 那才是真正被守住了。

02

🔧 換設備 / 換現場時要改哪裡

測試分成兩類,換環境時**只有第二類需要動**:

類別	內容	換設備要改嗎
環境無關	契約與穩定性(ST.1–ST.13、2A–2F、V6-*)。這些測項自己建立所需的素材、測完自己清掉	❌ 不用
環境相關	現場實際接了什麼:RS485 型號與 slave、擴充模組數量、MQTT broker	✅ 改 `web/tests/factory-env.js`

只改一個檔:`web/tests/factory-env.js`

```JS
const HARDWARE = {
  // RS485 上預期存在的設備。空陣列 = 該站沒有 RS485,相關測項自動略過
  rs485: [ { slaveId: 1, model: 'finder6m', name: '電流計' } ],
  // 預期的擴充模組「實際連線」數量。0 = 沒有擴充槽
  expansionModules: 1,
  hostDoWiring: '4 個 DO 接三色燈(DO1=紅 DO2=綠 DO3=藍 DO4=蜂鳴器)',
};
```

改完之後 **ST.14 會替你把關**:設定檔宣告的硬體與設備實際回報不符就直接紅燈, 不必等某個功能測莫名其妙失敗才回頭猜哪裡不一樣。

常見情境

情境	怎麼改
換一台設備(IP 不同)	`DUT_URL=http://新IP npm run test:gate`,不必改檔
對外 MQTT topic 前綴	**不必改** —— 留空時測試會向設備問它自己的 UID 自動組出來
RS485 換型號或 slave	改 `HARDWARE.rs485`(`model` 用 `/api/config` 的 `modbusDevices[].type` 值)
該站沒有 RS485	`EXPECT_RS485='' npm run test:gate`,或改檔案的 `rs485: []`
一份設定檔要服務多台不同配置	用環境變數逐台覆寫,不必改檔: `EXPECT_RS485='1:finder6m:電流計' EXPECT_EXPANSIONS=1`(8320) `EXPECT_RS485='' EXPECT_EXPANSIONS=0`(8310)
沒有擴充模組	`EXPECT_EXPANSIONS=0` → 自動略過
韌體出新版	`EXPECT_FW=6.0.xx npm run test:gate`,或改 `factory-env.js` 的預設值

> **原則**:測試碼一行都不該為了換環境而改。要改就改設定檔 —— 改測試碼會讓「這一站為什麼要特別處理」的知識散落各處,下一個人接手就看不懂。

03

文件用途

本文件是**正式工站操作規範**,供產線測試人員逐步執行。所有步驟可由一位測試員在單一工站完成。

自動化實作(v5.9.130+,跑真實 DUT):

🚀 **一鍵測試**(自動偵測 UID + 前置檢查 + 跑功能+頁面):

```bash
./scripts/run-factory-test.sh [DUT_IP] [ADMIN_TOKEN]
# 預設 192.168.72.77 / smmsadmin;輸出 12 tests PASS/FAIL
```

套件	涵蓋	執行
`web/tests/factory-functional.spec.js`	**決定性韌體能力**(beforeAll 自佈建 fixtures,不依賴設備現狀/RS485):連線、MQTT 輸入、2-source AND 順序強制、signal→rule→CH_TCP publish→broker loopback(6 tests)	`npm run test:factory:func`
`web/tests/factory-pages.spec.js`	Phase 2 五頁面渲染(Overview/Config/Rules/Devices/Settings,5 tests)	`npm run test:factory:pages`
`web/tests/factory-v6.spec.js`	**v6.0.x 能力回歸**(純 HTTP 唯讀契約,不改 config):Phase 0 機型身分(hasWifi/boardModel/version/mac)、Phase 6B 遠端診斷(/api/log + /api/diag + auth 401)、Phase 2E.6a MQTT 測試契約(202+輪詢)、Phase 2E.6b namespace 欄位 + TLS 出廠預設(13 tests)	`npm run test:factory:v6`
`web/tests/factory-cleanup.spec.js`	Phase 7 出貨清理(FR-H,部分 test.skip 待韌體 endpoint)	—

一次跑功能+頁面+v6 回歸:`npm run test:factory`(於 `web/`,共 24 tests)或 `./scripts/run-factory-test.sh`(repo root,自動偵測 UID)。`EXPECT_FW` 環境變數鎖定出廠版號基線(預設 `6.0.8`)。

決定性原則(出廠品質基準):functional 測試在 `beforeAll` **自己 idempotent 佈建** fixtures(wdprobe + ft_* 引擎元件)+ 清 retained,**不依賴設備現有 config、不依賴 RS485 硬體**。因此**同韌體 → 同結果**;唯有韌體能力改變才會讓結果變。對出廠空機也成立。fixtures 由 Phase 7 一併清除。**物理治具步驟**(砝碼/SG-002/DI 開關/拔網路線)無法自動化,仍由本文件手動 □ 執行。⚠️ 跑前 watchdog 須暫停(會 reboot 干擾)。

04

工站前置條件

治具接線

連接點	接線對象	API/UI 位置	用途
Host DI 1	Toggle Button	`/api/io di[0]`	數位輸入驗證
Host AI 2	FNIRSI-SG-002 信號產生器(0–10V)	`/api/io ai[1]`	類比輸入驗證
Host DO 1~4	三色燈 + 蜂鳴器(紅/綠/藍/蜂鳴)	`/api/io do[0..3]`	數位輸出驗證
Expansion 0 DO 3~6	第二組三色燈 + 蜂鳴器	`expansions[0].doNames[2..5]`	擴充模組驗證
RS485 Bus 0	Finder 6M.TB(**Slave ID=1**, Baud 9600, machineId=48)	`/api/config modbusDevices[0]`	電力表讀取
RS485 Bus 0	YX523R 光警器(**Slave ID=4**, machineId=24325)	`/api/config modbusDevices[1]`	RS485 寫入
Ethernet	工站電腦區網	—	Web UI + Playwright
MQTT Broker (新增)	工站本地 Mosquitto(明文 1883)	`factory/<topicPrefix>/...`	FR-G E2E 雙向訊息

平台容量基準（硬體衍生，6.0.8 沿用）

項目	容量	用途
Host DI / DO / AI	8 / 4 / 3	主機 I/O
Expansion 槽位	5（Exp0~Exp4，目前 fixture 只用 Exp0）	擴充模組
Modbus 設備 / Register	12 / 32	RS485
RS485 Bus	韌體 2 條，UI 只外露 Bus 0（Bus 1 由 FR-H1-03 確保不誤露）	—

變體識別（機型 × 網路，二維）

工站開始前必須讀 `/api/system` 判定 **兩個維度**，並寫入結果摘要 metadata：

維度 1 — 機型（來源 = OTP `boardInfo().wifi`，不可靠 `WiFi.status()`）：

- `hasWifi=false` / `boardModel="8310"` → **8310 機型**（無 WiFi 模組）：強制乙太網路；設定頁隱藏 WiFi 欄位；心跳燈 = **琥珀色（紅+綠同亮）**；H6 Failover **Skip**；AP 模式進不了。
- `hasWifi=true` / `boardModel="8320"` → **8320 機型**（有 WiFi）：可 WiFi；心跳燈 = **藍燈（LEDB）**；H6 Failover 依網路變體決定。

維度 2 — 網路變體（讀 `network.netType` 與 `/api/config interfaceMode`）：

- `netType="Ethernet"` 且 `interfaceMode=0/1` → **ethernet 變體**（H6 Failover Skip）
- `netType="WiFi"` 或 `interfaceMode=2` → **wifi 變體**（H6 Failover 必測，僅 8320 可能）

> ⚠ 8310 恆為 ethernet 變體(無 WiFi 硬體)。wifi 變體必為 8320。

工站 profile 需於啟動時設 `model: 8310|8320` + `variant: ethernet|wifi`，寫入結果摘要 metadata。

啟動檢查清單

- ☐ DUT 已上電，Status LED 閃爍中
- ☐ DUT 已取得 IP，工站電腦可 ping 通
- ☐ 瀏覽器可開啟 DUT Web UI（`http://<DUT-IP>`）
- ☐ RS485 設備已上電（Finder + YX523R 兩台）
- ☐ SG-002 信號產生器已上電
- ☐ **MQTT Broker** 已啟動（明文 1883）
- ☐ **機型 + 網路變體** 已從 `/api/system` 讀取並寫入結果摘要
- ☐ 首次燒錄後首開機：Web UI 完整渲染（**非裸 HTML**；app.js 走 QSPI 已載入）

05

Phase 0：連線檢查（FR-F、FR-A1-07）

#	步驟	驗收條件	類型
0.1	開啟瀏覽器，連接 DUT Web UI	Dashboard 頁面載入完成；無 JS error；**非裸 HTML**	自動
0.2	確認左下角韌體版本號 `#sideVersion`	顯示 `v6.0.8`（`/api/system version`；自動化 V6-ID.3 斷言，可 `EXPECT_FW` 覆寫）	自動
0.3	記錄 DUT UID `#sysDeviceUid`	24-hex 格式	自動
0.4	讀取 `/api/config configCrc`	寫入結果摘要（`firmware.configCrcReference`）	自動
0.5	讀取 `network.netType` 判定網路變體	profile 對齊；異常則 fail	自動
0.6	讀取 `/api/system` `boardModel` / `hasWifi` 判定機型	`hasWifi` =boolean；`boardModel` = `"Opta RS485 (8310)"`（hasWifi=false）/ `"Opta WiFi (8320)"`（hasWifi=true）；兩者一致；profile 對齊（自動化 V6-ID.1/2）	自動
0.7	讀取 **`/api/config`** `mac`	冒號大寫 MAC（如 `A8:61:0A:50:8B:61`）；對外 topic 身分 = 去冒號轉小寫 12-hex；寫入結果摘要（自動化 V6-ID.4）	自動

測試記錄：

```
DUT UID:        ____________________
DUT MAC:        ____________________  (12-hex 對外身分)
FW Version:     6.0.8 (預期)
Board Model:    8310 | 8320
Variant:        ethernet | wifi
configCrc:      ____________________  (出廠 baseline)
測試日期:       ____________________
測試員:         ____________________
```

06

Phase 1：初始化（FR-C1）

#	步驟	驗收條件	類型
1.1	清空所有殘留規則（含未命名「Rule 2」）	`/api/rules ruleCount` = 0	自動
1.2	關閉所有 DO 輸出	`/api/io do[]` 全為 false	自動
1.3	載入 baseline 設定檔（含 Finder + YX523R）	設定載入 Toast，`configCrc` 等於 baseline 預期值	自動
1.4	確認 `eepromPendingReboot == false`	`/api/system` 欄位	自動

07

Phase 2：UI 驗證

2A. Dashboard 總覽頁（FR-A1）

#	測試項目	驗收條件	對應 PRD
2A.1	統計卡片	`#diCnt` / `#doCnt` / `#ruleCnt` 非空	FR-A1-01
2A.2	MQTT 卡片四態	`#mqttSt` 為「未設定/連線中/已連線/失敗」之一	FR-A1-02
2A.3	Modbus TCP 卡片	卡片靜態存在；⚠️ 已知 UI bug：硬編碼顯示「Port 502」（FR-A1-03 註腳，PRD 附錄 E E-01）	FR-A1-03
2A.4	物理 / Expansion 切換 Tab	`🏠 主機 8DI / 4DO` ↔ `Digital-SSR #0 16DI / 8DO` 可切換	FR-A1-04
2A.5	RS485 卡片 — Finder 12 量測	12 格全顯示（電壓 V/電流 A/有功 W/無功 var/視在 VA/PF/頻率 Hz/諧波 %/總/正/反向 kWh），無 NaN	FR-A1-05
2A.6	頂部控制列	Poll Rate Selector + IP + 連線狀態徽章	FR-A1-06 + A1-10
2A.7	側邊欄頁尾	系統時間跳動 + `#sideVersion` 顯示 `v6.0.8`	FR-A1-07
2A.8	I/O 卡片三態徽章	DI 顯示「待機/未啟用」；DO 顯示「關閉/開啟/未啟用」	FR-A1-08
2A.9	每張卡片 ↺ 清除統計	點按只重設該通道	FR-A1-09

2B. Config 配置頁（FR-A2）

#	測試項目	驗收條件	對應 PRD
2B.1	Tab 切換	`🏠 主機 / Digital-SSR #0` 可切；`物理通道 / 虛擬通道 / TCP 通道` 三子分頁可切	FR-A2-01 / 02
2B.2	TCP 通道列表	進頁面後**等 5 秒**，`#tcpChannelsList` 渲染為清單或「尚無 TCP IO 通道」（不可保留「載入中...」）	FR-A2-02b
2B.3	物理輸入「資料型態」下拉	I1~I8 每行有 `數位/類比` dropdown	FR-A2-02c
2B.4	**DO 標籤 1-based**	物理通道輸出列表顯示 `DO1` ~ `DO4` `；不可出現` `DO0`	FR-A2-02d / WI-102
2B.5	虛擬通道類型	計數器 / 計時器 / 類比邏輯三型可切換	FR-A2-03
2B.6	設備樣板	「PATLITE NE-M1ATB-M」+「Toggle Button」可點擊套用	FR-A2-06
2B.7	TCP IO Modal	`+ 新增通道` → `#tcpIoModal` 開啟 + 欄位完整	FR-A2-08~15
2B.8	群組 Modal	`#groupModal` 編輯成員清單 + Device Tag	FR-A2-16
2B.9	儲存回饋	`#toast` 出現綠色成功訊息	FR-A2-04 / FR-A6-01

2C. Rules 規則頁（FR-A3）

#	測試項目	驗收條件	對應 PRD
2C.1	規則列表	`#rulesList` 不停在「載入中」	FR-A3-01
2C.2	優先權提示 banner	顯示「越下方的規則擁有最終覆蓋優先權」	FR-A3-05
2C.3	**規則卡片優先權徽章**	每卡片標題前綴 🟢/🟡/🔴 之一（priority 0/1/2 對應）	FR-A3-06b / WI-101
2C.4	新增規則	`+ 新增規則` → 右側 inline 編輯器 `#ruleEditor` 開啟	FR-A3-02
2C.5	三級優先權選擇器	`#rulePriority` 下拉含 `🟢 一般 / 🟡 高 / 🔴 緊急（互鎖）`	FR-A3-06
2C.6	互鎖實測	兩條互斥規則 + 一條設「緊急」→ DO 最終由互鎖決定	FR-A3-07
2C.7	啟停切換	toggle 後 `/api/rules` 與畫面同步	FR-A3-03

2D. Devices 設備頁（FR-A4）

#	測試項目	驗收條件	對應 PRD
2D.1	5 槽 Expansion 卡片	`#expSlots` 顯示 5 槽，未連接者標「未連接」	FR-A4-01
2D.2	匯流排掃描	`#btnScan` → 進度條 + 結果列表（含 Finder slave 1, YX523R slave 4）	FR-A4-02
2D.3	Modbus Probe	`#btnProbe` 對 Slave 1, Reg 0, Count 4 → 成功且 raw hex 有資料	FR-A4-03
2D.4	EEPROM Modal 開啟	`openEepromModal()` → 顯示橘色警告「DIP OFF/OFF + 斷電上電」	FR-A4-04
2D.5	RS485 Modal 11 fieldMask	新增 RS485 設備 →「總覽顯示」11 個 checkbox 全可勾選	FR-A4-05
2D.6	RS485 樣板	dropdown 含 Finder 6M / YX523R / 自訂三選項	FR-A4-06
2D.7	Bus Settings 不外露 Bus 1	`#busSettingsModal` 內僅 `Port 1`，無 `Port 2` / `bus1*`	FR-A4-08 + FR-H1-03

2E. Settings 設定頁（FR-A5）

#	測試項目	驗收條件	對應 PRD
2E.1	裝置識別	`#cfgName` + `#sysDeviceUid` 可讀；UID 點擊複製	FR-A5-01
2E.2	系統時間	`#sysCurrentTime.value` 每秒跳動（用 `.value` 不是 `.innerText`）	FR-A5-03
2E.3	授權卡片	`#sysExpiresAt.value` 顯示「永久授權」/具體日期/「未授權」	FR-A5-03
2E.4	License 啟用	`#btnLicenseActivate` → 輪詢 `/api/license` 直到 ACTIVE	FR-A5-03
2E.5	License Renewal	`#renewalTokenInput` + `renewLicense()` 路徑	FR-A5-11
2E.6	MQTT 設定	Broker / Port / Client / Auth 欄位儲存；**TLS 已實作**（`#mqttTls` 勾選 + `#mqttTlsInsecure`）；**出廠預設 `tlsInsecure=true`（不驗 CA，業主指定，勿改回）**	FR-A5-02
2E.6a	**MQTT 測試連線**（v6.0.4）	點「測試連線」→ POST 回 202 → 前端輪詢 `/api/mqtt/test` 顯示「測試中 n/30」→ 結果成功/失敗；**密碼欄留空 = 用已存密碼**；對已連線同 broker 直接短路判定成功	FR-A5-02
2E.6b	**MQTT 命名空間**（WI-155 / 6.0.0）	`#mqttProject` + `#mqttEnv` 兩欄留空 = legacy `mes/gateway/{uid}`；填值 = `{proj}-{env}/gateway/{mac}`（全小寫）；儲存後心跳 topic 對齊	FR-A5-02
2E.7	Config Server 雙位址 + auto-disable banner	`#csUrl` + `#csCloudUrl` 各自獨立；`configServerAutoDisabled=true` 時 banner 顯示	FR-A5-04 / 15
2E.8	網路模式三選一	`#netModeSelect` radio：自動 / 僅有線 / 僅 WiFi	FR-A5-05
2E.9	DHCP/Static 切換	`#ethDHCP` ↔ `#ethStatic` 對應顯示 / 隱藏 IP 欄位	FR-A5-21
2E.10	WiFi AP 掃描流程（變體 = wifi）	`switchToApMode()` → `scanWifi()` → 選 SSID → 儲存並重啟	FR-A5-06 / 20
2E.11	OTA 雙路徑	拖曳 `#otaFileInput` + 伺服器 `#otaVersionSelect`	FR-A5-08 / 23
2E.12	破壞性維護防呆	重啟 / 恢復原廠 / 清除 三 Modal 各有遮罩	FR-A5-09
2E.13	**MAC 軟體覆寫**（進階，WI-155）	進階區可填軟體 MAC（不碰硬體 OTP）+ 即時驗證；設定後 **90s 內無 IP 自動回退** 防失聯；複製鈕給對外格式（無冒號小寫）	FR-A5（WI-155）
2E.14	🖲 **設備日誌 按鈕**（v6.0.1）	設定頁「儲存名稱」與「🖲 設備日誌」**同一列並排**（v6.0.4 UI）；點按開 `/log.html` 顯示即時 serial log	FR-A5（WI-155-遠端診斷）

08

Phase 3：跨頁與持久化驗證（FR-B）

#	步驟	驗收條件	對應 PRD
3.1	修改 Device Name + I/O 別名	出現成功 Toast	FR-B1-01 / 03
3.2	F5 重新整理	設定保留	FR-B3-02
3.3	觸發系統重啟（`rebootDevice()` → `#rebootOverlay` 倒數）	90 秒內回上線	FR-B3-03 / FR-A5-22
3.4	重啟後驗證設定未掉	步驟 3.1 修改仍在	FR-B3-03
3.5	跨頁設定驅動呈現驗證	Config Alias 改 → Dashboard 卡片標題同步	FR-B1-01
3.6	TCP IO 持久化	新增 channel → 重啟後仍存在	FR-B1-07

09

Phase 4：硬體煙霧測試（FR-C）

H1. 初始化

#	步驟	驗收條件
H1.1	清空規則綁定	`/api/rules ruleCount` = 0
H1.2	關閉所有 DO	全部 OFF

H2. Host / Expansion DO 跑馬燈（FR-C2）

測試員操作：觀察燈號與通道對應

#	動作	預期結果	Pass/Fail
H2.1	Host DO 1 ON (1秒)	🔴 紅燈亮	☐
H2.2	Host DO 2 ON (1秒)	🟢 綠燈亮	☐
H2.3	Host DO 3 ON (1秒)	🔵 藍燈亮	☐
H2.4	Host DO 4 ON (1秒)	🔊 蜂鳴器響	☐
H2.5	Expansion 0 DO 3 ON (1秒)	🔴 第二組紅燈亮	☐
H2.6	Expansion 0 DO 4 ON (1秒)	🟢 第二組綠燈亮	☐
H2.7	Expansion 0 DO 5 ON (1秒)	🔵 第二組藍燈亮	☐
H2.8	Expansion 0 DO 6 ON (1秒)	🔊 第二組蜂鳴器響	☐

H3. RS485 驗證（FR-C3）

H3-A. YX523R 光警器寫入（Slave ID=4）

> 透過 Dashboard `setDO()` 或 `POST /api/modbus/manual_write`（FR-I1-02）

#	命令	暫存器	寫入值	預期結果	Pass/Fail
H3.1	紅燈長亮	17	`0x11`	光警器顯示紅色	□
H3.2	綠燈長亮	17	`0x13`	光警器顯示綠色	□
H3.3	紅燈爆閃	17	`0x31`	紅燈爆閃 2 秒	□
H3.4	關閉	17	`0x60`	警報熄滅	□

> **人工確認點**:測試員目視確認光警器燈色 / 爆閃模式(FR-C3-01)

H3-B. Finder 6M.TB 讀取驗證（Slave ID=1）

#	步驟	驗收條件	Pass/Fail
H3.5	Dashboard RS485 卡片顯示 Finder 12 量測	12 格非 NaN,數值合理	□
H3.6	Modbus Probe Slave 1 / Reg 0 / Count 4	回應成功,rawHex 有資料	□

H3-B2. 同型號多台 slaveId 持久化（v6.0.2，選做）

> 驗 6.0.2 修正:`ModbusRegister.slaveId` 存進 flash,重開機後不歸零、不退回陣列位置比對 → 同型號多台不串台。需兩台同型號設備(如兩台 SF965)。單台工站可跳過並標 `H3-B2: SKIPPED (單台)`。

#	步驟	驗收條件	Pass/Fail
H3.6a	兩台同型號各設**不同 slaveId(≤32)** 並儲存	兩張卡片各顯示自己 slaveId 的即時值,無交叉	□
H3.6b	**重開機**後回到 Dashboard	兩台仍以 slaveId 穩定綁定;值不互換、不串台	□

H3-B3. 離線來源不發假 0（v6.0.7，選做）

> 驗 6.0.7:converter 模板引用到**離線** RS485 來源時,整筆**跳過發佈**(不送 `{"counter":0}` 讓 MES 誤判計數歸零)。`ConverterEngine::pushTemplate` 依 `_fillHadOfflineSource` 跳過。需 broker 訂閱觀測。單台工站可跳過並標 `H3-B3: SKIPPED`。

#	步驟	驗收條件	Pass/Fail
H3.6c	建 converter 模板引用某 RS485 來源 → 閘控發佈,訂閱其 topic	來源在線時:每期收到含實值的 payload	□
H3.6d	拔線 / 關機讓來源**離線**,續觀測 topic	**完全靜默**(不送 `0`);設備 log 出現 `skip publish: RS485 source offline`	□
H3.6e	來源回線	自動恢復發佈實值	□

H3-B4. SF965 計數器原生解碼（v6.0.8，選做,需 SF965 硬體）

> 驗 6.0.8 新 dataType `VAR_SF965_PV` (=8):SF965 PV@addr4 為特規 32-bit——高 3 byte=計數值、低 byte=小數點位數(dot)→ 值 = `(raw>>8)/10^dot`。選 SF965 型號套預設暫存器即用此型別(不再靠 `÷256` scale 湊)。

#	步驟	驗收條件	Pass/Fail
H3.6f	RS485 新增設備選「SF965 計數器」樣板 → 套預設暫存器	PV 暫存器 dataType 顯示 `SF965 PV`(=8);儲存後讀回仍為 8	□
H3.6g	Dashboard 顯示 SF965 計數即時值	dot=0 為乾淨整數、dot≠0 自動小數;與機台面板顯示一致(顯示 + 發佈一致)	□

H3-C. TDA-08B 稱重控制器情境驗證（Slave ID=1，v5.9.111 sensor-fusion 計件器情境）

> 對應源碼提交 `9cea4cd`(v5.9.111)TDA-08B + sensor-fusion 虛擬通道 + 計數器 peak-detection。**v5.9.112+ 已移除 TDA-08B 模板**,本情境必須手動建立暫存器與虛擬通道。完整逐欄位 / 標定流程 / 排錯:`docs/hardware/setup-tda-08b.md`(內部文件,未公開) 治具:TDA-08B 稱重控制器 1 台、5kg 標準砝碼、感測器(壓力 / 應變片)。
>
> ❗ **curl 配置時 JSON 必須 compact**(無空格):Python 用 `json.dumps(obj, separators=(',', ':'))`,否則 firmware parser 靜默忽略。詳見 setup-tda-08b.md §2B。

Step 1:手動建立 RS485 設備(Slave ID=1, 9600/8N1, ABCD Big-Endian)

#	步驟	驗收條件	Pass/Fail
H3.7	RS485 設定 → 新增「自訂設備」名為 `TDA-08B 稱重控制器`,Slave=1,輪詢 2000 ms	卡片顯示 🟢 在線	□
H3.8	加 5 筆讀取暫存器(FC03):`實時值`@11(INT32) / `穩定狀態`@9(UINT16) / `線上狀態`@10(UINT16) / `峰值`@43(INT32) / `谷值`@75(INT32)	5 行顯示 ✅ 啟用,倍率=1,偏移=0	□
H3.9	加 2 筆寫入暫存器(FC06):`標定指令`@113(UINT16) / `砝碼值`@108(UINT16)	2 行顯示,模式=Write	□
H3.10	儲存設備設定	設備卡片顯示 7 個暫存器即時值	□

Step 2:建立 Scale 虛擬通道(sensor-fusion)

#	步驟	驗收條件	Pass/Fail
H3.11	Config → 邏輯類型「⚖️ 秤重計(重量→件數)」新增虛擬通道	Scale 通道卡片出現	□
H3.12	Scale 設定 — `來源`=Modbus `TDA-08B 實時值`,`tare`=空秤值,`unitWeight`=單件重量(克),`tolerance`=±10%	公式預覽顯示 `round((讀值 - tare) / uw)`	□
H3.13	空秤點「⚙️ 自動校準」→ 寫入 `標定指令=4`(清零)	實時值歸零 ±0.5g	□

Step 3:計數器 peak-detection 規則

#	步驟	驗收條件	Pass/Fail
H3.14	新增計數器虛擬通道 `合格計件器`,`triggerSource`=Scale.currentCount,`threshold`=1	通道卡片顯示 count=0	□
H3.15	規則:`合格達標` (count ≥ 10) → MQTT 通知 `mes/gateway/{uid}/scale/done`	規則 🟢 Active	□
H3.16	🔩 提示:放上 1 顆 5g 標準砝碼於秤上	Scale.currentCount = 1,計數 +1	□
H3.17	重複 9 次(共 10 顆砝碼)	每次讀值穩定在 unitWeight ± tolerance 才 +1;雜訊或半顆不算	□
H3.18	達到 count=10 觸發 MQTT 通知	broker 收到 `scale/done` 訊息	□

Step 4：標定指令往返驗證

#	步驟	驗收條件	Pass/Fail
H3.19	規則「MQTT→清零」: 外部 `mes/gateway/{uid}/scale/calibrate` → Modbus Write `標定指令=4`	TDA-08B 實時值歸零	□
H3.20	規則「MQTT→零點」: 外部 `... /scale/zero` → `標定指令=1`	TDA-08B 重新標定零點	□
H3.21	（選做）寫 `砝碼值=5000` + `標定指令=2` 標定增益	5g 砝碼讀值 = 5000 ±5	□

> **人工確認點**：H3.16 / H3.17 / H3.21 需測試員實際操作砝碼並目視確認讀值穩定（FR-C3-01 延伸）

Step 5：校正引導精靈 chain 驗證（v5.9.130+，選做）

> 驗證 MQTT-Chain 工作流校正精靈（4 步驟 + 燈號）。需先佈建 wizard（見 `guide-calibration-wizard.md` §5）。無 wizard 佈建則跳過並標 `H3 Step5: SKIPPED`。

#	步驟	驗收條件	Pass/Fail
H3.22	重置：發 `…/cmd/signal/3` =1→0	`…/cmd/signal/0` (state)=0，燈號全滅	□
H3.23	推進標零：發 `…/cmd/signal/1` =1→0（瞬時）	收到 `…/rules/{n}/trigger 校正標零`；state→1；第一顆燈亮	□
H3.24	順序強制：state=0 時發 `…/cmd/signal/2`（標增益）	**不該** fire（AND 條件 state==1 不成立）	□
H3.25	推進標增益：state=1 時發 `…/cmd/signal/2` =1→0	state→2；第二顆燈亮；reg108/113 寫入	□
H3.26	（有砝碼）放 5kg 砝碼跑標零→標增益完整流程	weight 校到 5.00 ±0.01	□

> **人工確認點**：H3.23/H3.25 目視燈號切換；H3.26 需實體砝碼。

H4. 類比輸入 AI 驗證（FR-C4）

#	步驟	驗收條件	Pass/Fail
H4.1	🔧 提示測試員：將 SG-002 調至 **5.00V**	—	—
H4.2	等待 3 秒後讀取 `/api/io ai[1]`	值在 **5.00V ±5%**（4.75 ~ 5.25V）	□
H4.3	🔧 提示測試員：將 SG-002 調至 **0.00V**	—	—
H4.4	等待 3 秒後讀取 `/api/io ai[1]`	值在 0 ~ 0.25V	□

> 容差數值在 HW 拆工後由 `web/tests/fixtures/factory-baseline.json` 鎖定。

H5. 數位輸入 DI 驗證（FR-C5）

#	步驟	驗收條件	Pass/Fail
H5.1	提示測試員：**按下** DI 1 開關	—	—
H5.2	等待偵測 `/api/io di[0] = true`	10 秒內偵測到	□
H5.3	提示測試員：**放開** DI 1 開關	—	—
H5.4	等待偵測 `/api/io di[0] = false`	10 秒內偵測到	□

H6. 網路 Failover 驗證（變體 = wifi 限定，FR-C6）

> Skip 條件：`variant != wifi` 時整個章節跳過，並在報告標 `H6: SKIPPED (ethernet-only)`

#	步驟	驗收條件	Pass/Fail
H6.1	拔除 Ethernet 網路線	—	—
H6.2	等 30 秒，DUT 自動切到 WiFi	WiFi IP 可 ping，MQTT 重連	□
H6.3	插回 Ethernet	—	—
H6.4	等 30 秒，DUT 切回 Ethernet	有線 IP 可 ping，MQTT 重連	□

H7. 跨協議多對多路由（FR-C7）

#	步驟	驗收條件	Pass/Fail
H7.1	Fan-out: TCP Parser → Host DO 1 + Expansion DO 3 + MQTT Converter	一次外部 MQTT 觸發 → 兩燈同亮 + 收到 broker 推送	□
H7.2	防迴圈: Converter Topic = Parser Topic	10 秒內無 MCU 重啟 / 無 watchdog	□

H8. 機型感知 UI 過濾 + 心跳燈（WI-151 / WI-152，8310 / 8320）

> 來源 = OTP `boardInfo().wifi`。此章節依機型分流：8310 驗「無 WiFi + 琥珀燈」，8320 驗「有 WiFi + 藍燈」。

#	步驟	8310（hasWifi=false）	8320（hasWifi=true）	Pass/Fail
H8.1	`/api/system` `hasWifi` / `boardModel`	`false` / `8310`	`true` / `8320`	□
H8.2	設定頁網路區 WiFi 欄位	**隱藏** SSID/密碼 + 介面模式選單（強制乙太網路）	露出 SSID/密碼 + 三選一模式	□
H8.3	目視心跳燈	**琥珀色（紅+綠同亮）**	**藍燈（LEDB）**	□
H8.4	AP 模式進入	進不了（無 WiFi，會 fallback）	可進 AP（`switchToApMode()`）	□

10

Phase 5：端到端業務流（FR-G）

> **必須在 Phase 4 全通過後才執行。**

#	步驟	驗收條件	Pass/Fail
5.1	配置 1 個 DI / 1 個 TCP Parser / 1 個虛擬輸入	三者 Enabled	☐
5.2	配置 1 個 DO / 1 個 TCP Converter / 1 個虛擬輸出	三者 Enabled	☐
5.3	建立 `IG_All_Inputs` / `OG_All_Outputs` 各納 3 件	群組 Modal 儲存成功	☐
5.4	建立 `Rule_E2E`：`IG_All_Inputs` → `OG_All_Outputs`（priority 0）	規則卡片含 🟢 徽章	☐
5.5	觸發外部 MQTT → Parser 命中	實體 DO 點亮 + Broker 收到 Converter 推送	☐
5.6	Dashboard 三類卡片同步 Active	Input Group / Rule / Output Group 皆變色	☐

11

Phase 6：OTA + 升級迴歸（FR-D / FR-E）

#	步驟	驗收條件	Pass/Fail
6.1	上傳 OTA payload（拖曳或伺服器版本）	`#otaProgressBar` 100% 完成	☐
6.2	觸發重啟	RTC Backup Register swap flag 設為待 swap	☐
6.3	🔭 重啟期間目視確認	**無紅燈閃爍**（紅燈 = swap fail）	☐
6.4	重啟後比對版本	`/api/system version` 已切到新版，無 rollback	☐
6.5	跨版本升級驗證（N-1 → N）	既有 Rule / Modbus / MQTT 設定無損；新欄位有預設值	☐
6.6	模擬 OTA 中斷後復原	5 分鐘內以舊版可開機	☐

12

Phase 6B：遠端維運健檢（v6.0.1）

> 驗證 VPN-only / 現場無 USB 情境的遠端診斷能力。需 `PERM_ADMIN`（Bearer `smmsadmin`）。

#	步驟	驗收條件	Pass/Fail
6B.1	`GET /api/log`（帶 Bearer）	回純文字 ring buffer（`text/plain`；RS485 輪詢 / MQTT / 開機序列可見）；**無 Bearer → 401**（自動化 V6-DIAG.1/2）	□
6B.2	開 `/log.html`（走 QSPI）	頁面渲染即時 log；設定頁「🟩 設備日誌」按鈕可達	□
6B.3	`GET /api/diag`（帶 Bearer）	回 `resetReason` / `bootDetail` / `prevResetReason` / `bootCount` / `prevUptimeSec` / `uptimeSec` / `rawRsr` / `fwVersion`；`fwVersion` 應等於 `/api/system version`；無 Bearer → 401（自動化 V6-DIAG.3/4/5）	□
6B.4	主動重啟後查 `resetReason`	標記正確重開 → `SOFTWARE`（`bootDetail` 附細節）；非當機不應標 `UNEXPECTED`	□
6B.5	**密碼遮罩不外洩**	log 內 MQTT/AP/license 密碼皆遮罩（`****`/長度），無明文	□

⚠️ 勿以「打整頁 + 高頻並發」壓 `/api/log`（會觸 IWDG 重開）；`/` 是 gzip，curl 用 `--compressed`。

13

Phase 7：出貨前清理（FR-H）

由 WI-103 自動化腳本執行。

#	步驟	驗收條件	Pass/Fail
7.0	**授權撤銷門控**（WI-145，選做）	撤銷狀態下：**TCPIO / MQTT 對外功能被擋；DIO / 管理介面不擋**（設備仍可設定/救援）；撤銷 banner 於下次心跳(~127s)顯示	□
7.1	License Reset → `/api/license status = UNLICENSED`	重啟後維持 UNLICENSED	□
7.2	清除測試殘留（Rules / Virtual / TCP IO / MQTT broker / Config Server / WiFi 密碼）	各端點回應全空 / 預設值	□
7.3	確認 `#busSettingsModal` 不外露 Bus 1	DOM 不含 `bus1*` element	□
7.4	記錄出廠 baseline `configCrc`	寫入結果摘要 + DUT QR 標籤	□
7.5	重啟一次 → 30 秒內三 endpoint 200	`/api/system` `/api/config` `/api/io`	□

註 — 校正精靈 / wdprobe / watchdog：H3-C Step5 佈建的 wizard signals/actions/rules、以及 watchdog 用的 `wdprobe` signal，**屬部署時佈建、非出廠基線**，會被步驟 7.2 一併清除。出廠不保留；現場部署時依 `guide-first-deployment.md` §6 重新佈建。

14

Phase 8：結果輸出與彙整（FR-F）

總結表

Phase	測試項目數	Pass	Fail	Skip
Phase 0 連線	7			
Phase 1 初始化	4			
Phase 2 UI (A~E)	45			
Phase 3 跨頁/持久化	6			
Phase 4 硬體 (H1~H8)	32 (H6 wifi/8320 限定 4；H3-B2 需兩台同型；H3-B3 需 broker 觀測；H3-B4 需 SF965；H8 依機型分流)			
Phase 5 E2E	6			
Phase 6 OTA + 升級	6			
Phase 6B 遠端診斷	5			
Phase 7 出貨清理	6			
合計	**117** (8310/ethernet 變體約 111，視選做項而定)			

> **自動化覆蓋**：`factory-functional` (6) + `factory-pages` (5) + `factory-v6` (13,含 Phase 0 身分 4 / Phase 6B 診斷 5 / Phase 2E.6a-6b 契約 4) = **24 tests 一鍵** (`npm run test:factory`)。其餘為需硬體/人工目視的手動項 (H2~H8 燈號/砝碼/DI/AI、Phase 6 OTA、Phase 7 清理等)。

結果摘要 metadata

```
DUT UID:          ____________________
DUT MAC:          ____________________ (12-hex 對外身分)
Board Model:      8310 | 8320
Variant:          ethernet | wifi
FW Version (前):  6.0.8
FW Version (後):  ____________________ (Phase 6 後)
configCrc (前):   ____________________
configCrc (後):   ____________________ (Phase 7 後)
Boot Reason:      ____________________ (Phase 6B /api/diag)
測試日期:         ____________________
測試員:           ____________________
工站位置:         ____________________
PRD v1.2 已知 bug 命中:  E-01 / E-04 / E-08 / E-09 (列舉)
擋出貨 bug:       無 | E-XX
```

判定

- □ **PASS** — 全部通過 + 0 個擋出貨 bug → 可出貨
- □ **CONDITIONAL** — 全部通過 + 含 🟡 不擋 bug → 可出貨但記錄
- □ **FAIL** — 任一擋出貨 bug 或 Phase 失敗 → 退回工程修復

失敗項目記錄

Phase #	描述	對應 PRD FR / 附錄 D 編號	對應 sprint card	備註

15

附錄 A：API 快速參考

用途	API	Method	對應 PRD
韌體 + 系統摘要	`/api/system`	GET	FR-0
全配置 roundtrip	`/api/config`	GET / POST	FR-A5-10
Config Server payload	`/api/config_server`	GET	FR-A5-04
I/O 即時值	`/api/io`	GET	FR-C
直接驅動 DO	`/api/io/do`	POST	FR-I1-01
規則	`/api/rules`	GET / POST	FR-A3
訊號群組（Signal）	`/api/signals`	GET / POST	FR-A3
動作群組（Action）	`/api/actions`	GET / POST	FR-A3
TCP IO	`/api/tcpio`	GET / POST	FR-A2
Converter 模板	`/api/converter/template`	GET / POST	FR-A2
授權狀態	`/api/license`	GET	FR-A5-03
授權啟用 / 續約	`/api/license/activate` / `/renew`	POST	FR-A5-03 / 11
授權憑證遞送（救 outbound 壞設備）	`/api/license/token`	POST	WI-147
網頁 Serial Log（v6.0.1）	`/api/log`	GET（PERM_ADMIN）	Phase 6B
開機診斷（v6.0.1）	`/api/diag`	GET（PERM_ADMIN）	Phase 6B
MQTT 測試連線（v6.0.4）	`/api/mqtt/test`	POST（202）/ GET（輪詢）	Phase 2E.6a
OTA 上傳 / 伺服器	`/api/ota/upload` / `/api/ota/server`	POST	FR-D / FR-A5-08
Modbus 掃描 / Probe / 改 EEPROM / 手寫	`/api/modbus/{scan, probe, eeprom_write, manual_write}`	POST	FR-A4 / FR-I
重啟 / 恢復原廠	`/api/reboot` / `/api/factory_reset`	POST	FR-A5-09 / FR-H1-01

> **不存在的端點**（404，腳本不可引用）：`/api/devices`、`/api/status`、`/api/health`、`/api/version`、`/api/expansion`、`/api/tcp_io`、`/api/templates`。

16

附錄 B：操作手冊章節對照

出廠測試 Phase	操作手冊章節
Phase 0 連線	§6 首次上線流程
Phase 2A Dashboard	§8 總覽頁操作
Phase 2B Config	§9 配置頁操作
Phase 2C Rules	§10 規則頁操作
Phase 2D Devices	§11 設備頁與 RS485 操作（含 EEPROM）
Phase 2E Settings	§12 設定頁操作
Phase 3 持久化	§14 備份、還原與工廠重置
Phase 4 H2 DO	§9.2 設定 DO1-DO4 輸出
Phase 4 H3 RS485	§11.2a 自訂 Modbus 設備 / §11.2b 手動寫入
Phase 4 H4 AI	§9.3 類比邏輯處理
Phase 4 H5 DI	§9.1 設定 I1-I8 輸入
Phase 4 H7 多對多	§10 規則引擎 + TCP IO
Phase 6 OTA	§13 韌體更新

17

附錄 C：PRD v1.2 已知 UI bug 對工站的影響

Bug ID	說明	影響 Phase	處理方式
E-01	Modbus TCP 卡片硬寫 Port 502	Phase 2A.3	腳本對 `/api/config tcp.port=5000` 斷言；UI 文字差異記為 known issue
E-03	Rules 卡片 priority 0 缺🟢 徽章	Phase 2C.3	✅ 已修（WI-101）
E-04	HTTP server 並發 reset	Phase 2D 全段	腳本層序列化 fetch；FR-A5-24 韌體層長期解
E-06	Config DO 0-based	Phase 2B.4	✅ 已修（WI-102）
E-07	baseline 殘留未命名規則	Phase 1.1 / Phase 7	由 WI-103 cleanup 處理
E-08	Modbus Probe 預設 Slave 3	Phase 2D.3	UX 改善卡，不擋
E-09	UI「Port 1」vs API `busId=0`	Phase 2D.7 / RS485 Modal	腳本斷言 API 數值，不對 UI 文字斷言

> **已關閉 (False Positive)**：E-02（TCP loading 慢顯示）、E-05（時間欄位 probe 用錯 selector）。

18

文件修訂紀錄

日期	版本	韌體基線	變更
2026-08-20	2.5	6.0.65	新增雲端備份/還原閉環(test:backup,4 項)與 ST.15 雲端設定完整性
2026-08-20	2.4	6.0.65	新增啟用/停用機制回歸(test:enable,6 項);factory-functional 的 CH_TCP 輸出改用 channelId、MQTT 輔助函式支援 TLS broker;ST.3 判準改為只看已啟用的規則
2026-08-19	2.3	6.0.65	新增 MQTT Converter 鏈路回歸(test:converter,6 項);新增環境設定檔 factory-env.js 與「換設備要改哪裡」章節;ST.14 環境符合性
2026-08-19	2.2	6.0.65	新增「先跑自動化閘門」章節與缺陷對應表;閘門擴充到 29 項(新增 ST.11 TCP 來源不可能條件、ST.12 token 免密碼更新、2F 佔位符殘留);基線自 6.0.8 更新
2026-07-16	2.1	6.0.8	/api/diag 欄位名對齊實際韌體

05 疑難排解

疑難排解

> 📅 最後更新:2026-07-02 | 📌 對象:所有使用者(現場 / 安裝 / 管理 / 開發)

遇到狀況時,先在下面**用症狀找**,再看對應的原因與解法。需要深入的指到對應手冊。

01

🔌 連線 / 開機

症狀	可能原因	怎麼解
瀏覽器開設備頁顯示**「應用載入失敗」**	韌體是用裸 `firmware.bin` 燒,缺 QSPI 上的網頁資產(被 factory-reset 清掉)	改用 `.mesb` **整包**安裝(含網頁資產);或逐一補傳 7 個 web 資產。見〈OTA 與內嵌 Web〉
找不到設備網頁 / 連不上	IP 不對、不同網段、DHCP 未取得	確認同網段;查 DHCP 配發的 IP;見〈新裝置配置〉
網頁偶發「連線失敗 / 重試」、存檔失敗	設備單執行緒,瀏覽器密集操作 + 輪詢並發吃滿 loop	放慢/暫停總覽輪詢、避免多分頁同時操作;大量設定改用 API。前端會自動重試,非真錯
開機很久才上線(卡 ~60 秒)	無網路線時 Ethernet 等 link 逾時	接好網路線;或設備會自動切到可用網路。見〈備援與韌性〉
改了 MAC 覆寫後,設備頁打不開 / 連不上	改 MAC = 換網卡身分 → DHCP 可能配**新 IP**;或路由器 ARP/DHCP 表仍記舊 MAC↔IP	見下方〈🔧 MAC 覆寫後找不到設備〉

🔧 MAC 覆寫後找不到設備(SOP)

改了「設定頁 → 網路設定 → 進階:MAC 位址覆寫」後若失聯,依序試:

1. **等 90 秒**:套用的 MAC 若連不上網路,設備會**自動退回原 MAC 並重開**(內建安全網)→ 回原本的 IP 即可開頁。
2. **查新 IP**:MAC 改成功時設備可能拿到**新 IP**。到**路由器後台**的 DHCP / 連線清單找新 IP,或看**雲端設備頁**(opta.smms.com.tw)回報的 IP,用新 IP 開頁。
3. **重啟路由器 / AP**:若短時間內改過多個 MAC,路由器的 ARP/DHCP 表可能殘留「同 IP↔多 MAC」造成干擾 → 重啟路由器清表(等 DHCP 租約到期亦可)。
4. **USB serial**:接 USB 看開機 log(`cat /dev/cu.usbmodem*`),裡面會印目前 MAC 與 IP。
5. **最後手段**:長按設備按鈕 factory-reset(清設定回硬體 MAC);或 USB 重燒。

> 💡 一般情況「設一次 MAC」只會讓路由器更新一次 ARP,不會有問題;上述干擾多半來自**短時間連續改多個 MAC**。MAC 是硬體身分,非必要不要改。

02

📟 遠端診斷:看不到現場設備(v6.0.1+)

設備在現場、只能經 VPN 進入、無法插 USB 時,不必到現場也能看 serial log 與重開原因。

① 看即時 serial log(設備在線時)

- 設定頁 →「🏷️ 裝置識別」卡 →「📟 **設備日誌・Serial Log**」按鈕,或直接開 `http://<設備IP>/log.html`。
- 需 **admin** 權限(與設定頁同一組 token;預設 `smmsadmin`)。
- 頁面每 5 秒自動刷新,可看 RS485 輪詢、MQTT、開機序列等即時 log;可下載、暫停、清畫面。

② 查「上次為何重開」(即使設備曾失聯)

- log 頁最上方橫幅顯示:**為何重開 / 上次撐多久 / 開機次數 / 目前運行**。也可 `GET /api/diag`(admin)。
- **重開原因**判讀:

- `SOFTWARE`(附細節 `ota` / `api-reboot` / `loop-wedge` / `mac-revert` / `net-failover` …)= **有意重開**,正常。
- `UNEXPECTED` = **非預期**(看門狗 IWDG / 斷電 / 當機)—— ⚠️ 若 **開機次數快速上升 + 上次撐很短**(如 47s)= **重開迴圈**,要查韌體/電源/網路。
- `UNKNOWN` = 剛換韌體後的第一次開機(正常)。

- 這些欄位也隨心跳上傳雲端(`opta.smms.com.tw` 設備 `meta`),**設備隨後失聯,雲端仍查得到上次開機原因**。

③ **限制(要知道)**:log 頁與 `/api/diag` 都需要**設備網路/網頁活著**。若設備正在 **reboot loop 且網路也死**,網頁進不去 → 改看**雲端**上一次心跳留下的 `bootReason`(若還來得及發過),或到現場用 **USB serial** / factory-reset 兜底。

03

🟢 LED 燈號

燈號	意義	動作
綠燈穩定	正常運作	—
🔵 藍色快閃(8320)/ 🟡 琥珀快閃(8310)	**MQTT 斷線警告**	檢查網路 / broker;見下方 MQTT 段
紅燈	嚴重錯誤 / Safe Mode	見〈硬體安裝與接線〉Safe Mode 復原

04

📡 MQTT

症狀	可能原因	怎麼解
MQTT 一直連不上(藍/琥珀燈閃)	網路不通、broker 位址/帳密錯	v5.9.261 起**出廠已預設官方 TLS broker + 帳密**,通常免設;若改過設定,確認 broker/port(8883 TLS)/帳密
改了「跳過憑證驗證」仍連不上	早期版本重連漏帶該旗標(已修 v5.9.200)	升級韌體
設定 MQTT 後沒生效	存檔時沒按二次確認 modal「仍要儲存」	重新存,確認跳出的確認框按下去

05

🔗 RS485 設備

症狀	可能原因	怎麼解
掃描掃不到設備	baud / parity 不符、接線(A/B 反接)、Slave ID 重複	對齊量測設備的 baud(常見 9600)與 parity(有的 8N2);檢查 A/B;確認每台 Slave ID 唯一
設備「在線」一直閃爍	匯流排 parity 與設備不一致(例設備 8N2、匯流排設 8N1)	把匯流排 parity 改成跟設備一致
數值顯示 0 / 不動	設備離線、暫存器位址或型別錯	確認在線;檢查暫存器位址(hex/dec)、資料型別、scale
刪掉一台設備後,其他設備數值錯位 / 跑到別張卡	暫存器舊版以陣列位置綁定	**新版改以 Slave ID 綁定(WI-153,修復中);重新存一次該設備的暫存器即綁定到 Slave ID**

06

🔐 授權

症狀	可能原因	怎麼解
跳紅色橫幅「授權已撤銷,對外資料輸出已停止」	雲端撤銷了這台的授權	請供應商在雲端重新核准;約 1～2 分鐘(心跳)後自動恢復
撤銷/核准後狀態沒馬上變	判決靠設備心跳(~127 秒)傳遞	屬正常延遲;管理者可在雲端強制推送心跳加速
授權被擋時,哪些停、哪些還能用	設計如此	**停**:對外 MQTT / TCPIO / 週期推送。**不停**:本機 DIO、RS485 寫、所有設定/管理頁

07

📤 流程不發 / 發太多

症狀	可能原因	怎麼解
流程完全不發	閘沒建/沒啟用、門控信號沒選、模板取不到值(設備離線/Slave ID 錯)	確認閘已建且啟用、通道有選門控信號、模板 `${rs485.<SlaveID>.欄位}` 的 SlaveID 與欄位名正確
一直重複發、太頻繁	用了「固定間隔」但想要「只在變化時」	改勾「🔁 變更時才發」、間隔填 0(v5.9.263)
發佈間隔跟預期不符	三種模式搞混	對照〈快速上手〉步驟 4 的三模式表
存檔說「名稱重複」	通道名和閘同名	通道與閘取不同名稱

> 找不到你的症狀?把現象(畫面 / LED / log)記下來,連同設備版本(系統資訊頁)回報供應商。