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