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/js
  -d '{"slaveId":4,"functionCode":3,"regAddress":0,"count":2}' $DUT/api/modbus/p
```

> 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。