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

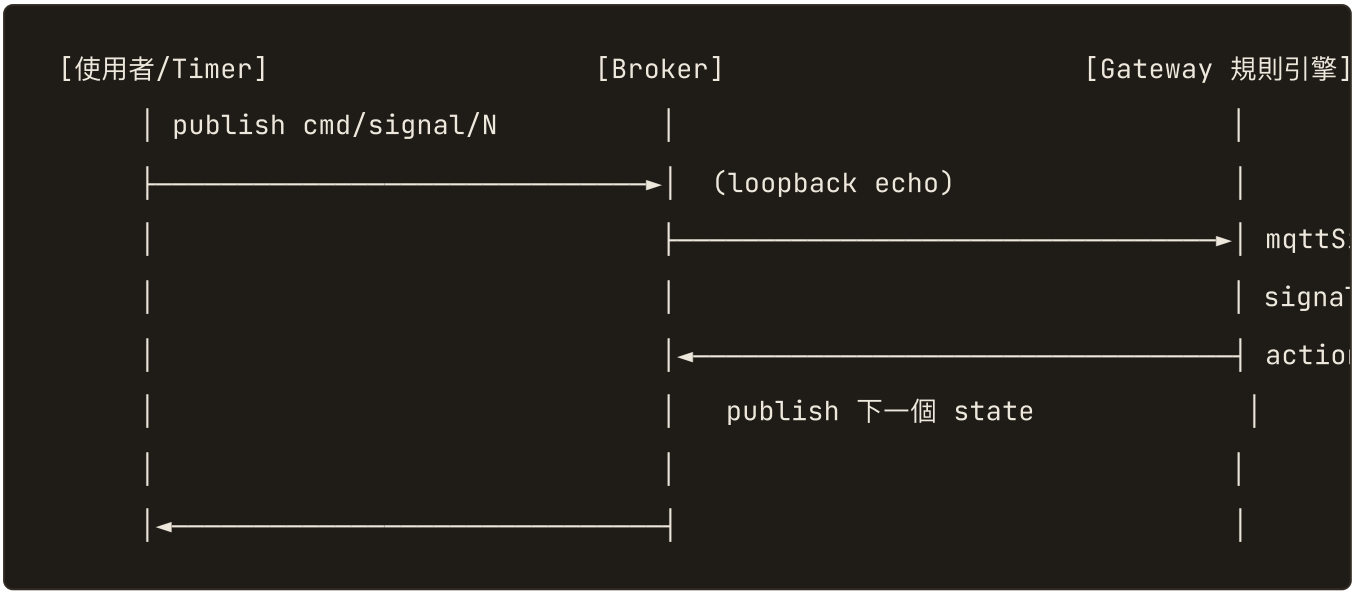
本文件說明 Opta（MES Gateway）以**純設定**（不改韌體）建構**有狀態、可組合工作流**的機制：以 MQTT topic 當「狀態匯流排」，透過規則引擎的 signal → rule → action 串接，再以 Converter timer 做定時驅動。TDA08B 秤重校正引導精靈即以此實作（見 `guide-calibration-wizard.md`）。

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。