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。