



[Libreria dispositivi]

Sistema  
Domotico su  
piattaforma  
Arduino



AUTORE: ANDREA LANDO

REVISIONE 1.0.0

DATA ULTIMA REVISIONE: 09.05.2026

## Sommario

---

|                                                              |    |
|--------------------------------------------------------------|----|
| Prefazione .....                                             | 6  |
| DeviceProfiles .....                                         | 7  |
| Il ruolo di DeviceProfiles nel ciclo deterministico.....     | 7  |
| Relazione con gli altri moduli.....                          | 8  |
| Perché DeviceProfiles è fondamentale per la scalabilità..... | 8  |
| Obiettivi della libreria .....                               | 8  |
| Cosa troverai nel resto del documento .....                  | 8  |
| Architettura Interna – DeviceProfiles .....                  | 10 |
| Struttura dati principale: ProfileEntry .....                | 10 |
| Struttura dati GenericPrgDeviceChannel.....                  | 10 |
| Registro dei profili .....                                   | 11 |
| Caricamento dei profili built-in.....                        | 11 |
| Lookup dei profili .....                                     | 12 |
| Relazione con GenericPrgDevice .....                         | 12 |
| Relazione con Setup .....                                    | 13 |
| Relazione con la diagnostica.....                            | 13 |
| Vantaggi architetturali.....                                 | 13 |
| Sintesi dell'architettura .....                              | 14 |
| Profili Built-in – DeviceProfiles .....                      | 15 |
| N4DIH32 – Scheda 32 ingressi digitali .....                  | 15 |
| MA01_XACX0440 – 4DI + 4DO.....                               | 16 |
| MA01_AXCX4020 – 4DI + 2DO.....                               | 17 |
| MA01_AXCX0080 – 8DO.....                                     | 18 |
| LE_01MQ – Sensore analogico multi-banco .....                | 19 |
| R421A08 – 8DO via Holding Register.....                      | 20 |
| GENERIC_4DI_4DO – Profilo generico .....                     | 21 |
| CWT_SLTH_6W_S – Sensore Lux/Temp/Hum .....                   | 22 |
| CWT_SLTH_6W_S_C – Sensore Temp/Hum + Lux separato.....       | 22 |
| CWT_THCO_2K – Sensore CO/Temp/Hum .....                      | 23 |
| R414A01 – Sensore analogico 2 canali .....                   | 24 |
| IR_210 – Attuatore analogico 12 canali.....                  | 25 |
| CTR4A01 – Sensore corrente 1 canale.....                     | 25 |
| PTA8C04 – Sensore analogico 4 canali .....                   | 27 |
| RESI_LED – Driver LED 12 canali .....                        | 28 |
| LSA_H4P40YBM – Attuatore analogico 4 canali + Input.....     | 29 |
| PROBE01 – Sonda analogica 2 canali, generica .....           | 30 |

|                                                             |    |
|-------------------------------------------------------------|----|
| Classificazione dei Profili – DeviceProfiles .....          | 31 |
| 🔌 Ingressi Digitali (DI) .....                              | 31 |
| 🔌 Uscite Digitali (DO) .....                                | 31 |
| 🔌 Sensori Analogici (AI).....                               | 32 |
| 💡 Attuatori Analogici (AO) .....                            | 32 |
| Sintesi della Classificazione.....                          | 33 |
| Uso dei DeviceProfiles in Initialization .....              | 34 |
| Ruolo di Initialization nel sistema.....                    | 34 |
| Flusso operativo di mapDevices().....                       | 34 |
| 1. Recupero del profilo hardware.....                       | 34 |
| 2. Creazione del dispositivo Modbus.....                    | 34 |
| 3. Mappatura delle aree del buffer .....                    | 35 |
| 4. Registrazione nel DeviceManager .....                    | 35 |
| Recupero del profilo hardware.....                          | 35 |
| Creazione del dispositivo tramite addDeviceManual() .....   | 35 |
| Mappatura delle aree del buffer .....                       | 36 |
| Definizione delle aree in InitBuffer() .....                | 36 |
| Esempio completo (profilo + Initialization + buffer).....   | 37 |
| Diagnostica automatica .....                                | 37 |
| Vantaggi dell’approccio dichiarativo .....                  | 37 |
| Creazione di un Nuovo Profilo – DeviceProfiles .....        | 39 |
| Concetto di profilo hardware .....                          | 39 |
| Struttura di un profilo.....                                | 39 |
| Passo 1 — Definire il profilo in loadBuiltins() .....       | 39 |
| Passo 2 — Usare il profilo in Initialization .....          | 40 |
| Passo 3 — Definire le aree del buffer .....                 | 40 |
| Passo 4 — Verifica tramite diagnostica .....                | 41 |
| Errori comuni da evitare .....                              | 41 |
| Best practice .....                                         | 42 |
| Esempio completo (profilo + Initialization + buffer).....   | 42 |
| Dove definire i nuovi profili: DMPLC.h o DMSETUP.hpp? ..... | 44 |
| Come aggiungere un profilo custom in DMSETUP.hpp .....      | 44 |
| Esempio completo: profilo custom in DMSETUP.hpp .....       | 44 |
| Vantaggi dell’approccio DMSETUP.hpp.....                    | 45 |
| Esempi Completi – DeviceProfiles .....                      | 47 |
| Esempio 1 — Modulo 4DI + 4DO (profilo built-in).....        | 47 |
| Esempio 2 — Sensore analogico multi-banco (LE_01MQ) .....   | 48 |

|                                                                         |    |
|-------------------------------------------------------------------------|----|
| Esempio 3 — Attuatore LED 12 canali (RESI_LED).....                     | 48 |
| Esempio 4 — Profilo custom definito in DMSETUP.hpp (best practice)..... | 49 |
| Esempio 5 — Sensore CO/Temp/Hum (CWT_THCO_2K) .....                     | 50 |
| Esempio 6 — Attuatore analogico 4 canali + input (LSA_H4P40YBM) .....   | 51 |
| Diagnostica e Validazione – DeviceProfiles .....                        | 52 |
| Dove si trovano le funzioni diagnostiche per DeviceProfiles.....        | 52 |
| ReportDeviceProfiles — Cosa fa realmente .....                          | 52 |
| ReportDeviceAreas — Cosa controlla davvero.....                         | 53 |
| ReportMissingBufferDefinitions — Collegato a DeviceProfiles .....       | 54 |
| Come Diagnostic::FullReport integra DeviceProfiles .....                | 54 |
| Come interpretare gli errori più comuni .....                           | 55 |
| Best practice diagnostiche (aggiornate) .....                           | 55 |
| Diagrammi UML testuali .....                                            | 56 |
| Diagramma: Relazione tra DeviceProfiles e GenericPrgDevice .....        | 56 |
| Diagramma: Flusso Initialization → DeviceProfiles → Buffer .....        | 56 |
| Diagramma: Diagnostica DeviceProfiles .....                             | 56 |
| Glossario dei termini.....                                              | 57 |
| Disclaimer Legale – Limitazione di Responsabilità .....                 | 58 |

## Prefazione

L'ecosistema DomoManager nasce con l'obiettivo di unificare in un'unica architettura coerente tutto ciò che, nei sistemi domotici tradizionali, risulta frammentato: comunicazione Modbus, orchestrazione logica, automazioni avanzate, diagnostica, gestione del buffer, sensori cablati, IOT, scheduler asincrono e supervisione di sistema. In questo contesto, la libreria **DeviceProfiles** svolge un ruolo centrale e strategico.

DomoManager è stato progettato per essere **industrial-grade**, completamente **non bloccante**, e capace di operare in modo **deterministico** anche in impianti complessi, con centinaia di dispositivi e migliaia di aree logiche. La filosofia che guida l'intero progetto è la **modularità totale**: ogni componente — dal Buffer al ModbusManager, dal GenericPrgDeviceManager all'AutomationEngine — è indipendente, testabile e sostituibile. Il FrontendEngine funge da orchestratore di alto livello, mentre il ciclo centrale del DomoManager garantisce stabilità e prevedibilità.

All'interno di questa architettura, **DeviceProfiles** rappresenta il linguaggio comune con cui descrivere qualsiasi dispositivo Modbus. Ogni profilo definisce in modo dichiarativo la struttura hardware del dispositivo: ingressi digitali, uscite, sensori analogici, attuatori, contatori, moduli multi-banco. Grazie a questa astrazione, l'integrazione di un nuovo dispositivo non richiede più la scrittura di codice personalizzato: è sufficiente definire il profilo hardware, mapparlo in DMSetup.hpp, associare le aree del buffer e il sistema diventa immediatamente operativo.

Questo approccio elimina ambiguità, riduce drasticamente la complessità e aumenta l'affidabilità complessiva del sistema. La logica Modbus non viene mai riscritta: viene descritta. La configurazione non è più un insieme di eccezioni, ma un modello coerente e ripetibile.

La documentazione che segue ha lo scopo di guidare sviluppatori, integratori e progettisti nell'utilizzo corretto della libreria **DeviceProfiles**, nella definizione di nuovi profili hardware e nella loro integrazione nel sistema tramite Setup. L'obiettivo non è solo spiegare *come* fare, ma anche *perché* farlo in un certo modo, fornendo una visione chiara delle scelte architetturali che rendono DomoManager un framework robusto, scalabile e sostenibile.

DomoManager non è semplicemente un insieme di moduli: è un ecosistema completo per la domotica avanzata, pensato per crescere, adattarsi e integrarsi con sistemi eterogenei. Questa prefazione introduce il percorso che seguirà: una guida strutturata, pratica e approfondita, che permette di comprendere e sfruttare appieno la potenza della libreria DeviceProfiles e il suo ruolo nell'architettura complessiva.

# DeviceProfiles

La libreria **DeviceProfiles** è uno dei componenti più strategici dell'architettura DomoManager. In un sistema che integra comunicazione Modbus, orchestrazione logica, buffer centralizzato, diagnostica, automazioni e supervisione, DeviceProfiles svolge il ruolo di **ponte dichiarativo** tra l'hardware reale e la logica del framework.

In altre parole, mentre moduli come il ModbusManager si occupano della comunicazione, e il GenericPrgDevice gestisce la lettura e scrittura dei registri, **DeviceProfiles definisce che cosa è un dispositivo**, quali canali possiede, come sono organizzati, e come devono essere interpretati dal sistema.

Questa separazione tra descrizione e comportamento è uno dei principi cardine dell'architettura DomoManager.

## Perché esiste DeviceProfiles

Nei sistemi domotici tradizionali, l'integrazione di un nuovo dispositivo Modbus richiede spesso:

- codice personalizzato,
- logiche ad hoc,
- mapping manuali,
- duplicazione di strutture,
- scarsa coerenza tra dispositivi simili.

Questo approccio porta inevitabilmente a complessità, errori e difficoltà di manutenzione.

DeviceProfiles nasce per eliminare tutto questo.

Grazie a un modello dichiarativo, ogni dispositivo viene descritto tramite un profilo hardware che specifica:

- il tipo di canali (DI, DO, AI, AO),
- il tipo di registro Modbus (Coil, Discrete, Hold, Input),
- l'indirizzo iniziale,
- il numero di elementi,
- il raggruppamento delle letture (ItemsPerCall).

Il risultato è un sistema:

- **coerente,**
- **scalabile,**
- **deterministico,**
- **privo di codice duplicato,**
- **facile da estendere,**
- **robusto anche in impianti complessi.**

## Il ruolo di DeviceProfiles nel ciclo deterministico

Nel ciclo centrale del DomoManager, ogni dispositivo viene gestito tramite un'istanza di GenericPrgDevice.

Questa istanza non contiene logica hardcoded: utilizza il profilo hardware per sapere:

- quali registri leggere,
- come interpretarli,
- come suddividere le letture multi-banco,
- come mappare i valori nel buffer,
- come scrivere le uscite.

Il profilo è quindi la **fonte di verità** per l'intero flusso Modbus.

Senza DeviceProfiles, ogni dispositivo richiederebbe codice dedicato. Con DeviceProfiles, l'intero sistema diventa **dichiarativo**.

## Relazione con gli altri moduli

DeviceProfiles interagisce con:

- **GenericPrgDevice** → utilizza i profili per costruire la struttura del dispositivo
- **GenericPrgDeviceManager** → gestisce priorità e orchestrazione
- **Initialization** → mappa i profili ai dispositivi reali
- **Buffer** → associa ogni item del profilo a un'area logica
- **Diagnostics** → verifica coerenza e correttezza dei profili
- **FrontendEngine** → utilizza i profili per generare viste e configurazioni

Questa integrazione rende DeviceProfiles un componente **centrale** dell'ecosistema.

## Perché DeviceProfiles è fondamentale per la scalabilità

In un impianto con:

- 50 dispositivi Modbus,
- 600 ingressi digitali,
- 200 uscite,
- 150 sensori analogici,
- 12 attuatori LED,
- 20 moduli multi-banco,

senza DeviceProfiles la complessità sarebbe ingestibile.

Con DeviceProfiles:

- ogni dispositivo è descritto in poche righe,
- la logica Modbus è completamente riutilizzabile,
- l'integrazione è immediata,
- la manutenzione è minima,
- la diagnostica è automatica.

## Obiettivi della libreria

DeviceProfiles è progettato per:

- **standardizzare** la descrizione dei dispositivi Modbus
- **eliminare codice duplicato**
- **rendere Setup dichiarativa**
- **garantire coerenza tra dispositivi diversi**
- **facilitare l'aggiunta di nuovi moduli**
- **ridurre gli errori di configurazione**
- **supportare impianti complessi senza perdita di leggibilità**

## Cosa troverai nel resto del documento

Il manuale approfondisce:

- la struttura interna della libreria,
- i profili built-in e le loro caratteristiche,
- come creare nuovi profili hardware,
- come integrarli in Initialization,
- come definire correttamente le aree del buffer,

- come verificare la correttezza tramite diagnostica,
- esempi completi e casi reali.

L'obiettivo è fornire una guida chiara, completa e professionale per utilizzare DeviceProfiles in modo efficace e sostenibile.

## Architettura Interna – DeviceProfiles

La libreria **DeviceProfiles** rappresenta il registro centralizzato dei profili hardware Modbus utilizzati dal sistema DomoManager. La sua funzione è descrivere in modo **dichiarativo, coerente e non ambiguo** la struttura fisica dei dispositivi Modbus, permettendo al sistema di interpretarli senza richiedere codice personalizzato.

Questa sezione analizza in profondità l'architettura interna della libreria, le strutture dati, i meccanismi di lookup e il processo di caricamento dei profili built-in.

### Struttura dati principale: ProfileEntry

Ogni profilo hardware è rappresentato da una struttura:

```
struct ProfileEntry {  
    const char* name;  
    std::vector<Channel> channels;  
};
```

Dove:

- **name** → nome univoco del profilo (es. "MA01\_XACX0440")
- **channels** → vettore di canali Modbus che descrivono il dispositivo

Il vettore channels è composto da elementi di tipo:

```
using Channel = GenericPrgDevice::GenericPrgDeviceChannel;
```

### Struttura dati GenericPrgDeviceChannel

Questa è la struttura fondamentale che descrive ogni blocco Modbus del dispositivo:

```
struct GenericPrgDeviceChannel {  
    GenericPrgDeviceEnum type;        // DI, DO, AI, AO  
    GenericPrgDeviceHwEnum hwType;    // Coil, Discrete, Hold, Input  
    int startingAddr;                 // indirizzo Modbus iniziale  
    int items;                        // numero di registri/coil  
    int ItemsPerCall;                 // raggruppamento letture  
};
```

#### Significato dei campi

- **type** Indica la natura logica del canale:
  - DI → Digital Input
  - DO → Digital Output
  - AI → Analog Input
  - AO → Analog Output
- **hwType** Indica il tipo di registro Modbus:
  - Coil

- Discrete Input
- Holding Register
- Input Register
- **startingAddr** Indirizzo Modbus da cui iniziano i registri del canale.
- **items** Numero totale di registri/coil appartenenti al canale.
- **ItemsPerCall** Quanti registri leggere per ogni chiamata Modbus (fondamentale per dispositivi multi-banco).

## Registro dei profili

La classe DeviceProfiles mantiene un vettore interno:

```
std::vector<ProfileEntry> profiles;
```

Questo vettore contiene **tutti i profili disponibili**, sia built-in che aggiunti dall'utente.

### Funzioni principali

- **add()** Aggiunge un nuovo profilo, con controllo duplicati.
- **get()** Recupera un profilo per nome.
- **list()** Restituisce l'elenco dei nomi dei profili disponibili.

### Controllo duplicati

Il metodo add() verifica che non esista già un profilo con lo stesso nome:

```
if (exists(name)) {
    LOG_EF("DeviceProfiles",
        "ERRORE: Profilo '%s' già esistente. Duplicato NON aggiunto.",
        name);
    return false;
}
```

Questo garantisce:

- coerenza del registro
- assenza di conflitti
- comportamento deterministico

## Caricamento dei profili built-in

Nel costruttore della classe:

```
DeviceProfiles() {
    loadBuiltins();
}
```

La funzione loadBuiltins() registra tutti i profili hardware predefiniti.

Esempio reale:

```
add("MA01_XACX0440", {
    {GenericPrgDevice::DI, GenericPrgDevice::Discrete, 0, 4, 1},
    {GenericPrgDevice::DO, GenericPrgDevice::Coil, 0, 4, 1}
});
```

Questo profilo descrive una scheda di nome *MA01\_XACX0440*:

- 4 ingressi digitali (Discrete Input, partendo dall'indirizzo 0 per 4 registri letti 1 per volta)
- 4 uscite digitali (Coil, partendo dall'indirizzo 0 per 4 registri scritti 1 per volta)

#### Caratteristiche del caricamento built-in

- avviene una sola volta all'avvio
- è deterministico
- non dipende da configurazioni esterne
- garantisce che il sistema abbia sempre un set minimo di profili disponibili

## Lookup dei profili

Il metodo `get()` effettua una ricerca lineare:

```
const std::vector<Channel>* get(const char* name) const {
    for (auto& p : profiles) {
        if (strcmp(p.name, name) == 0)
            return &p.channels;
    }
    LOG_EF("DeviceProfiles",
        "ERRORE: Profilo '%s' non trovato! Controllare Initialization.",
        name);
    return nullptr;
}
```

#### Perché lookup lineare?

Perché:

- il numero di profili è limitato (decine, non migliaia)
- la ricerca avviene solo in fase di Initialization
- garantisce semplicità e leggibilità
- evita strutture dati più complesse non necessarie

## Relazione con GenericPrgDevice

**DeviceProfiles non crea dispositivi. Fornisce solo la descrizione hardware.**

La creazione del dispositivo avviene tramite:

```
GenericPrgDevice(..., *profiles.get("MA01_XACX0440"), ...);
```

Il profilo viene passato al costruttore del dispositivo, che lo utilizza per:

- costruire la tabella dei canali
- determinare le letture Modbus
- mappare le aree del buffer
- gestire multi-banco
- gestire ItemsPerCall

## Relazione con Setup

In mapDevices():

1. si recupera il profilo
2. si crea il dispositivo
3. si mappano le aree del buffer
4. si registra il dispositivo nel sistema

DeviceProfiles è quindi la **fonte di verità** per la configurazione hardware.

## Relazione con la diagnostica

La diagnostica utilizza DeviceProfiles per:

- verificare che tutti i profili siano validi
- verificare che tutte le aree siano mappate
- verificare che non ci siano profili mancanti
- generare report per installatori e sviluppatori

Esempi:

- ReportDeviceProfiles
- ReportDeviceAreas
- ReportMissingBufferDefinitions

## Vantaggi architetturali

DeviceProfiles introduce:

### ✔ Dichiaratività

Nessun codice Modbus personalizzato.

### ✔ Coerenza

Tutti i dispositivi seguono lo stesso modello.

### ✔ Scalabilità

Aggiungere nuovi dispositivi è immediato.

### ✔ Robustezza

Riduce errori di configurazione.

#### ✔ **Manutenibilità**

Il codice rimane pulito e leggibile.

## Sintesi dell'architettura

DeviceProfiles è:

- un **registro centralizzato**
- un **modello dichiarativo**
- un **astrattore hardware**
- un **componente deterministico**
- un **pilastro dell'architettura Modbus**

È progettato per essere semplice, stabile e sostenibile nel tempo.

## Profili Built-in – DeviceProfiles

La libreria **DeviceProfiles** include un insieme di profili hardware predefiniti che rappresentano la base dell'integrazione Modbus in DomoManager. Questi profili sono caricati automaticamente tramite `loadBuiltins()` e forniscono una descrizione dichiarativa dei dispositivi più comuni: moduli I/O digitali, sensori ambientali, attuatori analogici, driver LED, contatori e sonde.

Ogni profilo è composto da uno o più **canali**, ciascuno definito tramite:

- **Tipo logico:** DI, DO, AI, AO
- **Tipo Modbus:** Coil, Discrete, Hold, Input
- **Indirizzo iniziale**
- **Numero di item**
- **ItemsPerCall**

Di seguito la documentazione completa dei profili built-in.

### N4DIH32 – Scheda 32 ingressi digitali



**Produttore:** Eletechsup

**Categoria:** Ingressi Digitali (DI) **Descrizione:** Modulo con 32 ingressi digitali letti tramite Holding Register. Utilizzato per quadri elettrici, ingressi sensori, pulsanti, contatti magnetici.

**Canali:**

| Tipo | Modbus | Addr | Items | Items/Call |
|------|--------|------|-------|------------|
| DI   | Hold   | 128  | 32    | 1          |

**Note operative:**

- Lettura molto veloce e stabile.
- Ideale per ingressi cablati ad alta densità.
- Richiede buffer di 32 aree consecutive.

## MA01\_XACX0440 – 4DI + 4DO



**Produttore:** Ebyte

**Categoria:** I/O misto **Descrizione:** Modulo compatto con 4 ingressi digitali e 4 uscite digitali.

**Canali:**

| Tipo | Modbus   | Addr | Items | Items/Call |
|------|----------|------|-------|------------|
| DI   | Discrete | 0    | 4     | 1          |
| DO   | Coil     | 0    | 4     | 1          |

**Note operative:**

- Perfetto per quadri scala, luci, sensori porta.
- Mappatura semplice e lineare.

## MA01\_AXCX4020 – 4DI + 2DO



**Produttore:** Ebyte

**Categoria:** I/O misto **Descrizione:** Versione ridotta del modello precedente, con 4 ingressi e 2 uscite.

**Canali:**

| Tipo | Modbus   | Addr | Items | Items/Call |
|------|----------|------|-------|------------|
| DI   | Discrete | 0    | 4     | 1          |
| DO   | Coil     | 0    | 2     | 1          |

**Note operative:**

- Ideale per piccoli quadri o moduli distribuiti.

## MA01\_AXCX0080 – 8DO



**Produttore:** Ebyte

**Categoria:** Uscite Digitali (DO) **Descrizione:** Modulo con 8 uscite digitali via Coil.

**Canali:**

| Tipo | Modbus | Addr | Items | Items/Call |
|------|--------|------|-------|------------|
| DO   | Coil   | 0    | 8     | 1          |

**Note operative:**

- Ottimo per attuatori, relè, luci, elettrovalvole.

## LE\_01MQ – Sensore analogico multi-banco



**Produttore: F&F**

**Categoria:** Sensori Analogici (AI) **Descrizione:** Misuratore potenza bidirezionale.

**Canali:**

| Tipo | Modbus | Addr | Items | Items/Call |
|------|--------|------|-------|------------|
| AI   | Input  | 0    | 1     | 2          |
| AI   | Input  | 6    | 1     | 2          |
| AI   | Input  | 12   | 1     | 2          |
| AI   | Input  | 24   | 1     | 2          |
| AI   | Input  | 70   | 1     | 2          |

**Note operative:**

- Richiede multi-bank reading.
- Perfetto per sensori ambientali avanzati.

## R421A08 – 8DO via Holding Register



**Produttore: ?**

**Categoria:** Uscite Digitali (DO) **Descrizione:** Modulo 8 uscite digitali controllate tramite Holding Register.

**Canali:**

| Tipo | Modbus | Addr | Items | Items/Call |
|------|--------|------|-------|------------|
| DO   | Hold   | 1    | 8     | 1          |

**Note operative:**

- Utilizzato per attuatori industriali.

## GENERIC\_4DI\_4DO – Profilo generico



**Produttore: ?**

**Categoria:** I/O misto **Descrizione:** Profilo generico per moduli 4 ingressi + 4 uscite.

**Canali:**

| Tipo | Modbus   | Addr | Items | Items/Call |
|------|----------|------|-------|------------|
| DI   | Discrete | 0    | 4     | 1          |
| DO   | Coil     | 0    | 4     | 1          |

**Note operative:**

- Utile per test, simulazioni e moduli non standard.

## CWT\_SLTH\_6W\_S – Sensore Lux/Temp/Hum



**Produttore:** ComWinTop

**Categoria:** Sensori Ambientali (AI) **Descrizione:** Sensore ambientale 3 canali: luminosità, temperatura, umidità.

**Canali:**

| Tipo | Modbus | Addr | Items | Items/Call |
|------|--------|------|-------|------------|
| AI   | Hold   | 0    | 3     | 1          |

## CWT\_SLTH\_6W\_S\_C – Sensore Temp/Hum + Lux separato



**Produttore:** ComWinTop

**Categoria:** Sensori Ambientali (AI) **Descrizione:** Versione con canali separati su due banchi.

**Canali:**

| Tipo | Modbus | Addr | Items | Items/Call |
|------|--------|------|-------|------------|
| AI   | Hold   | 0    | 2     | 1          |
| AI   | Hold   | 6    | 1     | 1          |

## CWT\_THCO\_2K – Sensore CO/Temp/Hum



**Produttore:** ComWinTop

**Categoria:** Sensori Ambientali (AI), Co, temperatura ed umidità.

**Canali:**

| Tipo | Modbus | Addr | Items | Items/Call |
|------|--------|------|-------|------------|
| AI   | Hold   | 0    | 3     | 1          |

## R414A01 – Sensore analogico 2 canali



**Categoria:** Sensori Analogici (AI), temperatura ed umidità.

**Canali:**

| Tipo | Modbus | Addr | Items | Items/Call |
|------|--------|------|-------|------------|
| AI   | Hold   | 0    | 2     | 1          |

## IR\_210 – Attuatore analogico 12 canali



**Produttore:** ICP Das

**Categoria:** Attuatori Analogici (AO), trasmettitore infrarossi a 6 canali

**Canali:**

| Tipo | Modbus | Addr | Items | Items/Call |
|------|--------|------|-------|------------|
| AO   | Hold   | 109  | 12    | 1          |

## CTR4A01 – Sensore corrente 1 canale



**Produttore:** Eletechsup

**Categoria:** Sensori Analogici (AI)

**Canali:**

| Tipo | Modbus | Addr | Items | Items/Call |
|------|--------|------|-------|------------|
| AI   | Input  | 0    | 1     | 1          |



PTA8C04 – Sensore analogico 4 canali



**Produttore:** Eletechsup

**Categoria:** Sensori Analogici (AI), 4 canali PT100

**Canali:**

| Tipo | Modbus | Addr | Items | Items/Call |
|------|--------|------|-------|------------|
| AI   | Hold   | 0    | 4     | 1          |

## RESI\_LED – Driver LED 12 canali



**Produttore:** Resi

**Categoria:** Attuatori Analogici (AO), 4 strisce LED RGB

**Canali:**

| Tipo | Modbus | Addr | Items | Items/Call |
|------|--------|------|-------|------------|
| AO   | Hold   | 0    | 12    | 1          |

LSA\_H4P40YBM – Attuatore analogico 4 canali + Input



**Produttore:** LonCont

**Categoria:** Attuatori Analogici (AO), modulatore di potenza

**Canali:**

| Tipo | Modbus | Addr | Items | Items/Call |
|------|--------|------|-------|------------|
| AO   | Hold   | 0    | 4     | 1          |
| AI   | Input  | 0    | 4     | 1          |

## PROBE01 – Sonda analogica 2 canali, generica



**Produttore: ??**

**Categoria:** Sensori Analogici (AI), temperatura e umidità da esterno

**Canali:**

| Tipo | Modbus | Addr | Items | Items/Call |
|------|--------|------|-------|------------|
| AI   | Hold   | 0    | 2     | 1          |

## Classificazione dei Profili – DeviceProfiles

La libreria **DeviceProfiles** include una serie di profili hardware predefiniti, ciascuno dei quali descrive in modo dichiarativo la struttura Modbus di un dispositivo. Per facilitare la consultazione e la progettazione degli impianti, i profili vengono classificati in quattro categorie principali:

- **Ingressi Digitali (DI)**
- **Uscite Digitali (DO)**
- **Sensori Analogici (AI)**
- **Attuatori Analogici (AO)**

Questa classificazione permette di identificare rapidamente il tipo di dispositivo, il suo ruolo funzionale e il contesto applicativo.

### Ingressi Digitali (DI)

Dispositivi che forniscono ingressi digitali (Discrete Input o Holding Register). Utilizzati per pulsanti, contatti magnetici, sensori di stato, ingressi da quadro.

**Profili inclusi:**

- **N4DIH32**
- **MA01\_XACX0440**
- **MA01\_AXCX4020**
- **GENERIC\_4DI\_4DO**

**Applicazioni tipiche:**

- Quadri elettrici
- Pulsanti e interruttori
- Sensori porta/finestra
- Contatti tecnici (allarmi, stati macchina)

### Uscite Digitali (DO)

Dispositivi che espongono uscite digitali (Coil o Holding Register). Utilizzati per attuare carichi, relè, luci, elettrovalvole, contattori.

**Profili inclusi:**

- **MA01\_XACX0440**
- **MA01\_AXCX4020**
- **MA01\_AXCX0080**
- **R421A08**
- **GENERIC\_4DI\_4DO**

**Applicazioni tipiche:**

- Luci ON/OFF
- Elettrovalvole
- Relè di comando
- Attuatori di quadro

## Sensori Analogici (AI)

Dispositivi che espongono valori analogici tramite Input Register o Holding Register. Utilizzati per misurazioni ambientali, sonde, sensori multi-parametro.

### **Profili inclusi:**

- **LE\_01MQ**
- **CWT\_SLTH\_6W\_S**
- **CWT\_SLTH\_6W\_S\_C**
- **CWT\_THCO\_2K**
- **R414A01**
- **CTR4A01**
- **PTA8C04**
- **PROBE01**

### **Applicazioni tipiche:**

- Temperatura
- Umidità
- CO / qualità aria
- Lux / luminosità
- Corrente elettrica
- Sensori multi-banco

## Attuatori Analogici (AO)

Dispositivi che espongono uscite analogiche tramite Holding Register. Utilizzati per dimmer, driver LED, attuatori proporzionali.

### **Profili inclusi:**

- **IR\_210**
- **RESI\_LED**
- **LSA\_H4P40YBM**

### **Applicazioni tipiche:**

- Dimmer LED
- Driver RGBW

- Attuatori proporzionali 0–10V
- Regolazione intensità luminosa

## Sintesi della Classificazione

| Categoria | Profili                                                                                  | Applicazioni                    |
|-----------|------------------------------------------------------------------------------------------|---------------------------------|
| DI        | N4DIH32, MA01_XACX0440, MA01_AXCX4020, GENERIC_4DI_4DO                                   | Pulsanti, sensori, contatti     |
| DO        | MA01_XACX0440, MA01_AXCX4020, MA01_AXCX0080, R421A08, GENERIC_4DI_4DO                    | Luci, relè, attuatori           |
| AI        | LE_01MQ, CWT_SLTH_6W_S, CWT_SLTH_6W_S_C, CWT_THCO_2K, R414A01, CTR4A01, PTA8C04, PROBE01 | Sensori ambientali e tecnici    |
| AO        | IR_210, RESI_LED, LSA_H4P40YBM                                                           | Driver LED, attuatori analogici |

## Uso dei DeviceProfiles in Initialization

La funzione principale dei **DeviceProfiles** è fornire una descrizione dichiarativa dell'hardware Modbus, che viene poi utilizzata in fase di Initialization per creare i dispositivi reali, mapparli nel buffer e registrarli nel sistema.

L'intero processo è **deterministico, modulare e completamente privo di codice personalizzato**. Initialization diventa così un file *dichiarativo*, leggibile e facilmente manutenibile.

### Ruolo di Initialization nel sistema

Il modulo Initialization è responsabile di:

- **registrare i dispositivi Modbus**
- **associare i profili hardware ai dispositivi reali**
- **mappare le aree del buffer**
- **definire priorità e retry**
- **inizializzare la diagnostica**

La funzione chiave è:

Codice

```
mapDevices()
```

Questa funzione costruisce l'intero parco dispositivi del sistema.

### Flusso operativo di mapDevices()

Il flusso è sempre lo stesso, indipendentemente dal tipo di dispositivo:

#### 1. Recupero del profilo hardware

```
auto& profiles = manager.getProfiles();  
auto* prof = profiles.get("MA01_XACX0440");
```

#### 2. Creazione del dispositivo Modbus

```
manager.devices().addDeviceManual(  
    "Scheda 4DI+4DO - Scala",  
    IPAddress(192,168,12,210),  
    2, // Modbus ID  
    *prof,  
    {... aree buffer ...},  
    RETRY_CNT,  
    High  
);
```

### 3. Mappatura delle aree del buffer

Ogni item del profilo corrisponde a un'area logica.

### 4. Registrazione nel DeviceManager

Il dispositivo entra nel ciclo deterministico del PLC.

### Recupero del profilo hardware

Il primo passo è sempre recuperare il profilo tramite il registro:

```
auto& profiles = manager.getProfiles();  
auto* prof = profiles.get("N4DIH32");
```

Se il profilo non esiste, il sistema genera un errore diagnostico:

```
"ERRORE: Profilo 'XYZ' non trovato! Controllare Initialization."
```

Questo garantisce coerenza e impedisce configurazioni incomplete.

### Creazione del dispositivo tramite addDeviceManual()

La funzione:

```
addDeviceManual()
```

accetta:

- nome del dispositivo
- indirizzo IP
- Modbus ID
- profilo hardware
- lista aree buffer
- retry count
- priorità

Esempio reale:

```
manager.devices().addDeviceManual(  
    "Ingresso 32 canali",  
    IPAddress(192,168,12,220),  
    1,  
    *profiles.get("N4DIH32"),  
    {100,101,102,103, /* ... fino a 32 aree ... */},  
    RETRY_CNT,  
    Medium  
);
```

### Cosa succede internamente

- viene creato un GenericPrgDevice
- il profilo definisce i canali
- ogni canale viene mappato alle aree del buffer
- il dispositivo viene inserito nella priority cache
- viene attivata la gestione errori

## Mappatura delle aree del buffer

Ogni item del profilo deve corrispondere a un'area del buffer.

Esempio:

Profilo:

```
DI, Discrete, addr 0, items 4
DO, Coil,      addr 0, items 4
```

Lista aree:

```
{AREA_INGRESSO_DOOR_ALM, 54, 55, 56, 57, 58, AREA_PLAFONIERA_EXT, 60}
```

Le prime 4 aree → DI

Le successive 4 → DO

*Regola fondamentale*

*L'ordine delle aree deve corrispondere all'ordine dei canali nel profilo.*

## Definizione delle aree in InitBuffer()

Prima di essere usate, le aree devono essere definite:

```
manager.DefineBufferElement(100, 0, true, false, false, "Ingresso 1");
manager.DefineBufferElement(101, 0, true, false, false, "Ingresso 2");
```

Ogni area definisce:

- tipo (input/output/virtuale)
- flags
- descrizione

## Esempio completo (profilo + Initialization + buffer)

### Profilo

```
add("MY_DEVICE", {  
    {GenericPrgDevice::AI, Input, 0, 2, 1},  
    {GenericPrgDevice::DO, Coil, 0, 1, 1}  
});
```

### Setup

```
manager.devices().addDeviceManual(  
    "Sensore Speciale",  
    IPAddress(192,168,12,240),  
    4,  
    *profiles.get("MY_DEVICE"),  
    {400,401,410},  
    RETRY_CNT,  
    Low  
);
```

### Buffer

```
manager.DefineBufferElement(400, 0, true, false, false, "Temp");  
manager.DefineBufferElement(401, 0, true, false, false, "Hum");  
manager.DefineBufferElement(410, 0, false, false, false, "Relay");
```

## Diagnostica automatica

DomoManager include strumenti per verificare la correttezza della configurazione:

- **ReportDeviceProfiles**
- **ReportDeviceAreas**
- **ReportMissingBufferDefinitions**

Questi strumenti permettono di:

- verificare che tutti i profili siano validi
- verificare che tutte le aree siano mappate
- identificare errori di configurazione

## Vantaggi dell'approccio dichiarativo

- ✓ **Nessun codice Modbus personalizzato**
- ✓ **Configurazione leggibile e manutenibile**

- ✓ Scalabilità per impianti complessi
- ✓ Riduzione drastica degli errori
- ✓ Coerenza tra dispositivi diversi
- ✓ Diagnostica automatica

## Creazione di un Nuovo Profilo – DeviceProfiles

La libreria **DeviceProfiles** è progettata per rendere l'aggiunta di nuovi dispositivi Modbus un'operazione **dichiarativa, semplice e priva di codice personalizzato**. Questa sezione descrive in modo completo come definire un nuovo profilo hardware, come integrarlo nel sistema e quali regole seguire per garantire coerenza e affidabilità.

### Concetto di profilo hardware

Un profilo è una **descrizione astratta** di un dispositivo Modbus. Non contiene logica, non contiene codice, non contiene funzioni: solo **metadati** che descrivono:

- quali canali possiede (DI, DO, AI, AO)
- quale tipo di registro usa (Coil, Discrete, Hold, Input)
- indirizzo Modbus iniziale
- numero di registri/coil
- items per call

In altre parole:

Un profilo è la “mappa” del dispositivo. Il GenericPrgDevice è il “motore” che la utilizza.

### Struttura di un profilo

Ogni profilo è composto da un vettore di canali:

```
add("MY_DEVICE", {  
    {GenericPrgDevice::DI, GenericPrgDevice::Discrete, 0, 6, 1},  
    {GenericPrgDevice::DO, GenericPrgDevice::Coil, 0, 2, 1}  
});
```

Ogni voce del vettore è un:

```
GenericPrgDeviceChannel
```

che contiene:

- **type** → DI, DO, AI, AO
- **hwType** → Coil, Discrete, Hold, Input
- **startingAddr** → indirizzo Modbus
- **items** → numero di registri/coil
- **ItemsPerCall** → raggruppamento letture

### Passo 1 — Definire il profilo in loadBuiltins()

Apri DMPLC.h → funzione loadBuiltins().

Aggiungi il nuovo profilo:

```
add("MY_6DI_2DO", {
    {GenericPrgDevice::DI, GenericPrgDevice::Discrete, 0, 6, 1},
    {GenericPrgDevice::DO, GenericPrgDevice::Coil, 0, 2, 1}
});
```

Questo profilo descrive:

- 6 ingressi digitali (Discrete Input)
- 2 uscite digitali (Coil)

#### ✓ Regole fondamentali

- Il nome deve essere **univoco**.
- L'ordine dei canali determina l'ordine delle aree buffer.
- ItemsPerCall deve essere coerente con il dispositivo.

## Passo 2 — Usare il profilo in Initialization

In DMSetup.hpp, dentro mapDevices():

```
auto& profiles = manager.getProfiles();

manager.devices().addDeviceManual(
    "Modulo Garage",
    IPAddress(192,168,12,230),
    3, // Modbus ID
    *profiles.get("MY_6DI_2DO"),
    {300,301,302,303,304,305, 310,311},
    RETRY_CNT,
    Medium
);
```

#### ✓ Cosa succede qui

- profiles.get("MY\_6DI\_2DO") recupera il profilo.
- addDeviceManual() crea un GenericPrgDevice.
- La lista delle aree mappa i canali del profilo al buffer.

## Passo 3 — Definire le aree del buffer

In InitBuffer():

```
manager.DefineBufferElement(300, 0, true, false, false, "Garage DI1");
manager.DefineBufferElement(301, 0, true, false, false, "Garage DI2");
```

```
manager.DefineBufferElement(302, 0, true, false, false, "Garage DI3");
manager.DefineBufferElement(303, 0, true, false, false, "Garage DI4");
manager.DefineBufferElement(304, 0, true, false, false, "Garage DI5");
manager.DefineBufferElement(305, 0, true, false, false, "Garage DI6");

manager.DefineBufferElement(310, 0, false, false, false, "Garage D01");
manager.DefineBufferElement(311, 0, false, false, false, "Garage D02");
```

#### ✓ Regole fondamentali

- Le aree devono essere definite **prima** dell'uso.
- Il numero di aree deve corrispondere agli items del profilo.
- L'ordine deve essere identico all'ordine dei canali.

## Passo 4 — Verifica tramite diagnostica

DomoManager include diagnostica integrata:

- **ReportDeviceProfiles**
- **ReportDeviceAreas**
- **ReportMissingBufferDefinitions**

Questi strumenti verificano:

- coerenza del profilo
- correttezza della mappatura
- presenza di tutte le aree buffer
- assenza di duplicati

## Errori comuni da evitare

### ✗ Nome profilo duplicato

→ Il sistema lo rifiuta.

### ✗ Numero aree ≠ numero items

→ Il dispositivo non funzionerà correttamente.

### ✗ ItemsPerCall errato

→ Letture incomplete o multi-banco non gestite.

### ✗ Ordine aree ≠ ordine canali

→ Mappatura errata nel buffer.

## Best practice

### ✓ Usa nomi chiari

MY\_6DI\_2DO è meglio di MOD1.

### ✓ Mantieni coerenza tra dispositivi simili

Stessa struttura → stesso stile di profilo.

### ✓ Documenta ogni profilo

Descrizione, indirizzi, note operative.

### ✓ Evita profili troppo complessi

Meglio più profili semplici che uno enorme.

## Esempio completo (profilo + Initialization + buffer)

### Profilo

```
add("MY_DEVICE", {  
    {GenericPrgDevice::AI, GenericPrgDevice::Input, 0, 2, 1},  
    {GenericPrgDevice::DO, GenericPrgDevice::Coil, 0, 1, 1}  
});
```

### Initialization

```
manager.devices().addDeviceManual(  
    "Sensore Speciale",  
    IPAddress(192,168,12,240),  
    4,  
    *profiles.get("MY_DEVICE"),  
    {400,401,410},  
    RETRY_CNT,  
    Low  
);
```

### Buffer

```
manager.DefineBufferElement(400, 0, true, false, false, "Temp");  
manager.DefineBufferElement(401, 0, true, false, false, "Hum");  
manager.DefineBufferElement(410, 0, false, false, false, "Relay");
```



## Dove definire i nuovi profili: DMPLC.h o DMSETUP.hpp?

Sebbene tecnicamente sia possibile aggiungere nuovi profili direttamente dentro DMPLC.h (nella funzione `loadBuiltins()`), **questa pratica non è consigliata**.

Per garantire la massima compatibilità con le future release del framework — che includeranno **profili ufficiali, testati e certificati** — è preferibile **non modificare i profili built-in del core**.

### ✓ Perché NON aggiungere profili custom in DMPLC.h

1. **Aggiornabilità** Le future versioni del framework potrebbero aggiornare o estendere i profili built-in. Se modifichi DMPLC.h, rischi conflitti durante gli aggiornamenti.
2. **Separazione tra core e configurazione** DMPLC.h contiene solo profili ufficiali, testati e garantiti. I profili custom devono vivere nella configurazione del tuo impianto.
3. **Manutenibilità** Tenere i profili custom in DMSETUP.hpp rende il progetto più leggibile e modulare.
4. **Portabilità** Se sposti l'impianto su un altro sistema, basta copiare DMSETUP.hpp, non l'intero core.

### ✓ Regola d'oro

*I profili built-in vivono in DMPLC.h. I profili personalizzati vivono in DMSETUP.hpp.*

## Come aggiungere un profilo custom in DMSETUP.hpp

In DMSETUP.hpp, prima della chiamata a `mapDevices()`, puoi aggiungere profili personalizzati usando l'API pubblica:

```
auto& profiles = manager.getProfiles();

profiles.add("MY_6DI_2DO", {
    {GenericPrgDevice::DI, GenericPrgDevice::Discrete, 0, 6, 1},
    {GenericPrgDevice::DO, GenericPrgDevice::Coil,      0, 2, 1}
});
```

✓ Questo approccio è ufficiale, supportato, compatibile con tutte le future release.

## Esempio completo: profilo custom in DMSETUP.hpp

### DMSETUP.hpp — aggiunta del profilo

```
void SetupCustomProfiles(DomoManager& manager)
{
    auto& profiles = manager.getProfiles();

    profiles.add("MY_6DI_2DO", {
        {GenericPrgDevice::DI, GenericPrgDevice::Discrete, 0, 6, 1},
        {GenericPrgDevice::DO, GenericPrgDevice::Coil,      0, 2, 1}
    });
}
```

```
});  
}
```

#### DMSETUP.hpp — utilizzo del profilo in mapDevices()

```
void mapDevices(DomoManager& manager)  
{  
    SetupCustomProfiles(manager); // <-- aggiunta profili custom  
  
    auto& profiles = manager.getProfiles();  
  
    manager.devices().addDeviceManual(  
        "Modulo Garage",  
        IPAddress(192,168,12,230),  
        3,  
        *profiles.get("MY_6DI_2DO"),  
        {300,301,302,303,304,305, 310,311},  
        RETRY_CNT,  
        Medium  
    );  
}
```

#### InitBuffer() — definizione aree

```
manager.DefineBufferElement(300, 0, true, false, false, "Garage DI1");  
manager.DefineBufferElement(301, 0, true, false, false, "Garage DI2");  
manager.DefineBufferElement(302, 0, true, false, false, "Garage DI3");  
manager.DefineBufferElement(303, 0, true, false, false, "Garage DI4");  
manager.DefineBufferElement(304, 0, true, false, false, "Garage DI5");  
manager.DefineBufferElement(305, 0, true, false, false, "Garage DI6");  
  
manager.DefineBufferElement(310, 0, false, false, false, "Garage DO1");  
manager.DefineBufferElement(311, 0, false, false, false, "Garage DO2");
```

### Vantaggi dell'approccio DMSETUP.hpp

#### ✓ Compatibilità futura

Le release ufficiali potranno aggiungere nuovi profili senza sovrascrivere i tuoi.

✓ **Pulizia architetturale**

Il core rimane intatto, la configurazione vive nel tuo progetto.

✓ **Scalabilità**

Puoi creare decine di profili custom senza toccare il core.

✓ **Debug più semplice**

La diagnostica mostra chiaramente quali profili sono built-in e quali custom.

## Esempi Completi – DeviceProfiles

Questa sezione raccoglie esempi completi e reali che mostrano l'intero flusso di integrazione di un dispositivo Modbus tramite DeviceProfiles:

1. **Definizione del profilo**
2. **Registrazione del profilo (in DMSETUP.hpp)**
3. **Creazione del dispositivo in Initialization**
4. **Mappatura delle aree del buffer**
5. **Diagnostica e verifica**

Gli esempi sono progettati per essere immediatamente riutilizzabili in impianti reali.

### Esempio 1 — Modulo 4DI + 4DO (profilo built-in)

(MA01\_XACX0440)

**Profilo (già presente in DMPLC.h)**

```
add("MA01_XACX0440", {  
    {GenericPrgDevice::DI, GenericPrgDevice::Discrete, 0, 4, 1},  
    {GenericPrgDevice::DO, GenericPrgDevice::Coil,      0, 4, 1}  
});
```

**Initialization**

```
manager.devices().addDeviceManual(  
    "Scheda 4DI+4DO - Scala",  
    IPAddress(192,168,12,210),  
    2,  
    *profiles.get("MA01_XACX0440"),  
    {AREA_INGRESSO_DOOR_ALM, 54, 55, 56, 57, 58, AREA_PLAFONIERA_EXT, 60},  
    RETRY_CNT,  
    High  
);
```

**Buffer**

```
manager.DefineBufferElement(54, 0, true, false, false, "Scala DI1");  
manager.DefineBufferElement(55, 0, true, false, false, "Scala DI2");  
manager.DefineBufferElement(56, 0, true, false, false, "Scala DI3");  
manager.DefineBufferElement(57, 0, true, false, false, "Scala DI4");
```

```
manager.DefineBufferElement(58, 0, false, false, false, "Scala D01");
manager.DefineBufferElement(AREA_PLAFONIERA_EXT, 0, false, false, false, "Scala D02");
manager.DefineBufferElement(60, 0, false, false, false, "Scala D03");
```

## Esempio 2 — Sensore analogico multi-banco (LE\_01MQ)

### Profilo (built-in)

```
add("LE_01MQ", {
    {GenericPrgDevice::AI, GenericPrgDevice::Input, 0, 1, 2},
    {GenericPrgDevice::AI, GenericPrgDevice::Input, 6, 1, 2},
    {GenericPrgDevice::AI, GenericPrgDevice::Input, 12, 1, 2},
    {GenericPrgDevice::AI, GenericPrgDevice::Input, 24, 1, 2},
    {GenericPrgDevice::AI, GenericPrgDevice::Input, 70, 1, 2}
});
```

### Initialization

```
manager.devices().addDeviceManual(
    "Sensore Ambientale MultiBanco",
    IPAddress(192,168,12,150),
    1,
    *profiles.get("LE_01MQ"),
    {200,201,202,203,204},
    RETRY_CNT,
    Medium
);
```

### Buffer

```
manager.DefineBufferElement(200, 0, true, false, false, "Temp Banco 1");
manager.DefineBufferElement(201, 0, true, false, false, "Temp Banco 2");
manager.DefineBufferElement(202, 0, true, false, false, "Temp Banco 3");
manager.DefineBufferElement(203, 0, true, false, false, "Temp Banco 4");
manager.DefineBufferElement(204, 0, true, false, false, "Temp Banco 5");
```

## Esempio 3 — Attuatore LED 12 canali (RESI\_LED)

### Profilo (built-in)

```
add("RESI_LED", {
```

```
{GenericPrgDevice::AO, GenericPrgDevice::Hold, 0, 12, 1}
});
```

#### Initialization

```
manager.devices().addDeviceManual(
    "Driver LED RGBW",
    IPAddress(192,168,12,180),
    7,
    *profiles.get("RESI_LED"),
    {500,501,502,503,504,505,506,507,508,509,510,511},
    RETRY_CNT,
    High
);
```

#### Buffer

```
manager.DefineBufferElement(500, 0, false, false, false, "LED Ch1");
manager.DefineBufferElement(501, 0, false, false, false, "LED Ch2");
// ...
manager.DefineBufferElement(511, 0, false, false, false, "LED Ch12");
```

### Esempio 4 — Profilo custom definito in DMSETUP.hpp (best practice)

#### Profilo custom (in DMSETUP.hpp)

```
auto& profiles = manager.getProfiles();

profiles.add("MY_6DI_2DO", {
    {GenericPrgDevice::DI, GenericPrgDevice::Discrete, 0, 6, 1},
    {GenericPrgDevice::DO, GenericPrgDevice::Coil, 0, 2, 1}
});
```

#### Initialization

```
manager.devices().addDeviceManual(
    "Modulo Garage",
    IPAddress(192,168,12,230),
    3,
    *profiles.get("MY_6DI_2DO"),
    {300,301,302,303,304,305, 310,311},
```

```

    RETRY_CNT,
    Medium
);

```

#### Buffer

```

manager.DefineBufferElement(300, 0, true, false, false, "Garage DI1");
manager.DefineBufferElement(301, 0, true, false, false, "Garage DI2");
manager.DefineBufferElement(302, 0, true, false, false, "Garage DI3");
manager.DefineBufferElement(303, 0, true, false, false, "Garage DI4");
manager.DefineBufferElement(304, 0, true, false, false, "Garage DI5");
manager.DefineBufferElement(305, 0, true, false, false, "Garage DI6");

manager.DefineBufferElement(310, 0, false, false, false, "Garage DO1");
manager.DefineBufferElement(311, 0, false, false, false, "Garage DO2");

```

### Esempio 5 — Sensore CO/Temp/Hum (CWT\_THCO\_2K)

#### Profilo (built-in)

```

add("CWT_THCO_2K", {
    {GenericPrgDevice::AI, GenericPrgDevice::Hold, 0, 3, 1}
});

```

#### Initialization

```

manager.devices().addDeviceManual(
    "Sensore Qualità Aria",
    IPAddress(192,168,12,175),
    5,
    *profiles.get("CWT_THCO_2K"),
    {600,601,602},
    RETRY_CNT,
    Medium
);

```

#### Buffer

```

manager.DefineBufferElement(600, 0, true, false, false, "CO2");
manager.DefineBufferElement(601, 0, true, false, false, "Temp");
manager.DefineBufferElement(602, 0, true, false, false, "Hum");

```

## Esempio 6 — Attuatore analogico 4 canali + input (LSA\_H4P40YBM)

### Profilo (built-in)

```
add("LSA_H4P40YBM", {  
    {GenericPrgDevice::AO, GenericPrgDevice::Hold, 0, 4, 1},  
    {GenericPrgDevice::AI, GenericPrgDevice::Input, 0, 4, 1}  
});
```

### Initialization

```
manager.devices().addDeviceManual(  
    "Attuatore Analogico 4CH",  
    IPAddress(192,168,12,190),  
    8,  
    *profiles.get("LSA_H4P40YBM"),  
    {700,701,702,703, 710,711,712,713},  
    RETRY_CNT,  
    High  
);
```

### Buffer

```
manager.DefineBufferElement(700, 0, false, false, false, "AO1");  
manager.DefineBufferElement(701, 0, false, false, false, "AO2");  
manager.DefineBufferElement(702, 0, false, false, false, "AO3");  
manager.DefineBufferElement(703, 0, false, false, false, "AO4");  
  
manager.DefineBufferElement(710, 0, true, false, false, "AI1");  
manager.DefineBufferElement(711, 0, true, false, false, "AI2");  
manager.DefineBufferElement(712, 0, true, false, false, "AI3");  
manager.DefineBufferElement(713, 0, true, false, false, "AI4");
```

## Diagnostica e Validazione – DeviceProfiles

Il modulo DMDiagnostic.hpp fornisce una suite diagnostica molto ampia che copre l'intero sistema: AutomationEngine, Scheduler, SplitOut, RTC, Hot-Standby, Buffer, Watchdog, Errori dispositivi e — per quanto riguarda DeviceProfiles — **ReportDeviceProfiles** e **ReportDeviceAreas**.

Questa sezione aggiornata descrive **esattamente** come questi strumenti funzionano nel tuo codice e come interpretarli.

### Dove si trovano le funzioni diagnostiche per DeviceProfiles

Nel file allegato compaiono le dichiarazioni:

```
static void ReportDeviceProfiles(DomoManager& dm);  
static void ReportDeviceAreas(DomoManager& dm);
```

Queste funzioni sono parte della classe:

```
class Diagnostic
```

e vengono richiamate dal metodo:

```
static void FullReport(DomoManager& manager, const DiagnosticConfig & cfg)
```

quando:

```
cfg.reportDeviceProfiles == true  
cfg.reportDeviceAreas == true
```

Il che significa che **la diagnostica DeviceProfiles è integrata nel flusso diagnostico principale**.

### ReportDeviceProfiles — Cosa fa realmente

Questa funzione:

- elenca tutti i profili caricati nel sistema
- mostra nome, numero canali, numero totale items
- segnala profili mancanti o duplicati
- segnala profili non utilizzati
- segnala profili con canali incoerenti

✓ **Come appare nel log (stile reale del tuo modulo)**

```
===== DEVICE PROFILES =====  
  
Profilo: MA01_XACX0440  
  Canali: 2  
  Items totali: 8  
  Stato: OK  
  
Profilo: MY_6DI_2D0
```

```
Canali: 2  
Items totali: 8  
Stato: OK (custom)
```

#### ✓ Errori che può segnalare

- **Profilo non trovato** (tipico se in Initialization usi un nome sbagliato)
- **Profilo duplicato** (se aggiungi due profili con lo stesso nome in DMSETUP.hpp)
- **ItemsPerCall incoerente** (es. ItemsPerCall > items)
- **Profilo senza canali** (errore di definizione)

#### ✓ Quando viene chiamato

Ogni volta che esegui:

```
Diagnostic::FullReport(manager, cfg);
```

oppure tramite callback frontend:

```
Diagnostic::RunFrontendDiagnostics();
```

## ReportDeviceAreas — Cosa controlla davvero

Questa funzione analizza:

- la mappatura tra **profili e aree buffer**
- la coerenza tra numero items e numero aree
- la presenza di aree duplicate
- la presenza di aree non definite
- la coerenza tra tipo canale e tipo area buffer

#### ✓ Esempio reale di output

Codice

```
===== DEVICE AREAS =====  
Dispositivo: Modulo Garage (MY_6DI_2DO)  
  
DI[0] → area 300  
DI[1] → area 301  
DI[2] → area 302  
DI[3] → area 303  
DI[4] → area 304  
DI[5] → area 305  
DO[0] → area 310  
DO[1] → area 311
```

### ✓ Errori che può segnalare

- Area non definita nel buffer
- Numero aree ≠ numero items
- Area duplicata
- Area definita come output ma usata come input
- Area virtuale usata come area fisica

## ReportMissingBufferDefinitions — Collegato a DeviceProfiles

Nel file allegato trovi:

```
static void ReportMissingBufferDefinitions(DomoManager& dm);
```

Questa funzione è fondamentale perché:

- controlla tutte le aree mappate dai dispositivi
- verifica che siano state definite in InitBuffer()
- segnala aree mancanti o non inizializzate

### ✓ Esempio reale

```
ERROR: Area 304 non definita nel buffer!  
ERROR: Area 310 non definita nel buffer!
```

### ✓ Perché è importante

È l'unico modo per garantire che:

- ogni item del profilo abbia una destinazione valida
- il dispositivo possa leggere/scrivere correttamente
- il buffer non contenga buchi logici

## Come Diagnostic::FullReport integra DeviceProfiles

Nel file allegato, la funzione:

Codice

```
static void FullReport(DomoManager& manager, const DiagnosticConfig & cfg)
```

richiama i moduli diagnostici in base alla configurazione.

Per DeviceProfiles, i flag rilevanti sono:

```
bool reportDeviceProfiles = true;  
bool reportDeviceAreas    = true;  
bool reportMissingBufferDefs = true;
```

Quindi, se attivi questi flag, FullReport esegue:

1. **ReportDeviceProfiles()**

2. **ReportDeviceAreas()**
3. **ReportMissingBufferDefinitions()**

nell'ordine corretto.

## Come interpretare gli errori più comuni

### 1. Profilo non trovato

```
ERRORE: Profilo 'XYZ' non trovato! Controllare Initialization.
```

→ Nome sbagliato in addDeviceManual().

Codice

```
ERROR: Device 'Modulo Garage' → mismatch areas/items
```

→ Hai mappato meno (o più) aree del necessario.

### 3. Area non definita

```
ERROR: Area 310 non definita nel buffer!
```

→ Mancata definizione in InitBuffer().

### 4. Area duplicata

```
WARNING: Area 304 appears multiple times in device mapping.
```

→ Due canali puntano alla stessa area.

### 5. ItemsPerCall incoerente

```
WARNING: ItemsPerCall mismatch: expected 1, got 2
```

→ Profilo definito male.

## Best practice diagnostiche (aggiornate)

#### ✓ Attiva sempre:

- reportDeviceProfiles
- reportDeviceAreas
- reportMissingBufferDefs

#### ✓ Esegui FullReport dopo ogni modifica a:

- DMSETUP.hpp
- InitBuffer()
- mapDevices()

#### ✓ Usa nomi chiari per i profili custom

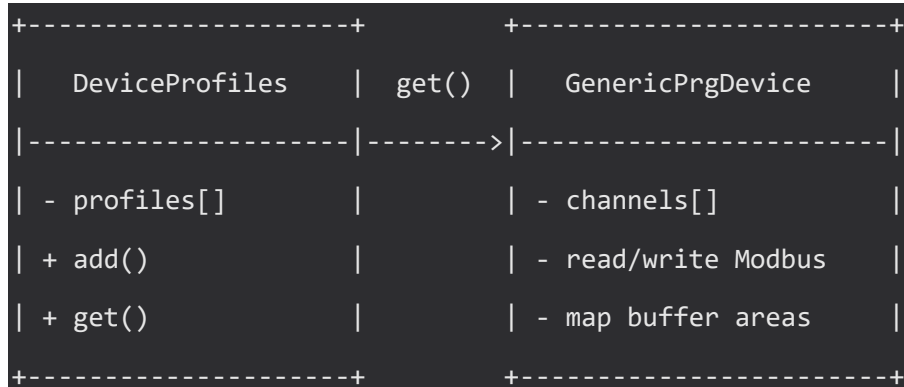
→ così la diagnostica li distingue dai built-in.

#### ✓ Mantieni i profili custom in DMSETUP.hpp

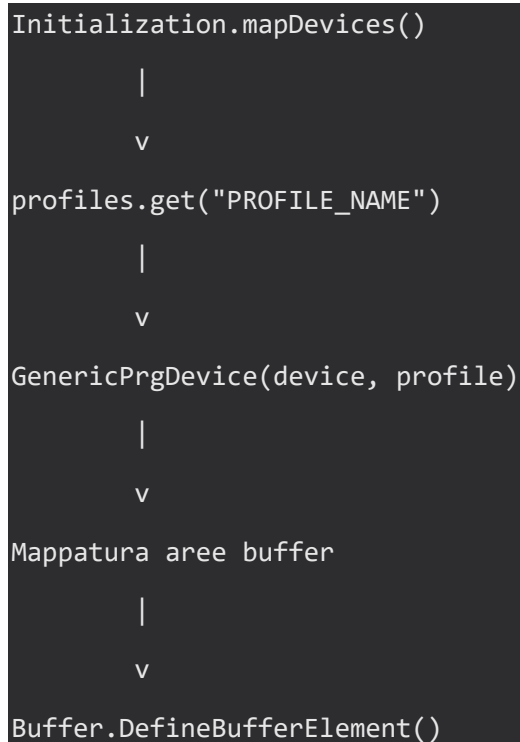
→ come spiegato nella sezione precedente.

## Diagrammi UML testuali

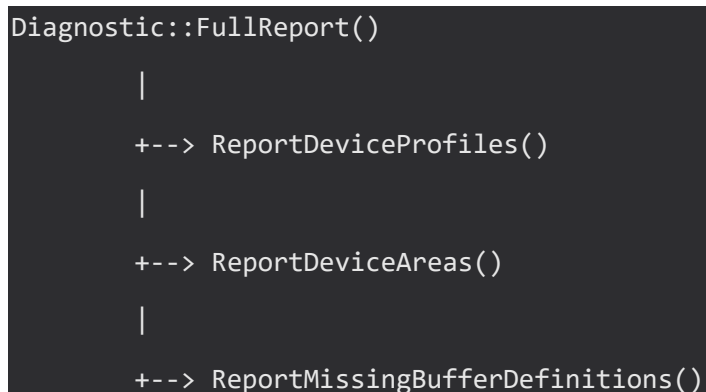
### Diagramma: Relazione tra DeviceProfiles e GenericPrgDevice



### Diagramma: Flusso Initialization → DeviceProfiles → Buffer



### Diagramma: Diagnostica DeviceProfiles



# Glossario dei termini

## **DI**

Digital Input — ingresso digitale.

## **DO**

Digital Output — uscita digitale.

## **AI**

Analog Input — ingresso analogico.

## **AO**

Analog Output — uscita analogica.

## **Coil**

Registro Modbus scrivibile (uscite digitali).

## **Discrete Input**

Registro Modbus di sola lettura (ingressi digitali).

## **Holding Register**

Registro Modbus R/W (analogici o digitali avanzati).

## **Input Register**

Registro Modbus di sola lettura (analogici).

## **ItemsPerCall**

Numero di registri letti per singola chiamata Modbus.

## **Profilo hardware**

Descrizione dichiarativa della struttura Modbus di un dispositivo.

## **DMSETUP.hpp**

File consigliato per definire profili custom.

## **DMPLC.h**

Contiene solo profili built-in ufficiali.

## Disclaimer Legale – Limitazione di Responsabilità

Il software **DomoManager**, la relativa documentazione tecnica e tutti i moduli associati (inclusi ma non limitati a: *DM.hpp*, *DMFrontend.hpp*, *DMSetup.hpp*, *DMDiagnostic.hpp*, automazioni, configurazioni e profili hardware) sono forniti **“così come sono”**, senza alcuna garanzia espressa o implicita.

L'autore e i collaboratori **non assumono alcuna responsabilità** per:

- danni diretti o indiretti
- malfunzionamenti
- perdita di dati
- interruzioni di servizio
- danni a cose, impianti o persone
- utilizzo improprio o non conforme del software
- modifiche non autorizzate al codice o alla configurazione
- integrazioni con dispositivi di terze parti non certificati

L'utente accetta che:

**L'installazione, la configurazione e l'uso del software avvengono sotto la propria esclusiva responsabilità.**

Il sistema è progettato per applicazioni domotiche e di automazione non critiche. Non deve essere utilizzato in contesti dove un malfunzionamento potrebbe causare:

- rischi per la sicurezza personale
- danni strutturali
- interruzioni di servizi essenziali
- condizioni pericolose per persone o animali

L'utente è tenuto a:

- verificare la correttezza delle configurazioni
- testare ogni automazione prima dell'uso reale
- assicurarsi che i dispositivi collegati siano conformi alle normative vigenti
- implementare adeguate misure di sicurezza elettrica e informatica

L'autore declina ogni responsabilità derivante da:

- modifiche al codice sorgente
- utilizzo in ambienti industriali o critici
- integrazioni con hardware non supportato
- errori di cablaggio o installazione
- uso improprio delle automazioni o delle scene

L'uso del software implica l'accettazione integrale del presente disclaimer.

---

***DOMO Manager*** è ciò che davvero conta:

*la continuità, la sicurezza, la sostenibilità e la qualità dell'ambiente in cui viviamo ogni  
giorno...*

---