Snap7Helper Framework – Technical Reference Manual

IMPORTANT WARNING & DISCLAIMER

THIS LIBRARY IS PROVIDED FOR TESTING, EDUCATIONAL, AND NON-PRODUCTION USE ONLY.

Snap7Helper is NOT intended for real-life or production industrial applications. It must never be used for safety, emergency stop, or protection systems.

The author assumes **NO responsibility or liability** for any damages, injuries, losses, or legal consequences arising from its use.

1. Overview

Snap7Helper is a C++ middleware wrapper around the Snap7 library. It abstracts raw byte manipulation, endianness conversion, and offset calculation, allowing developers to interact with Siemens PLC memory using human-readable addresses.

4.1 Data Logging to CSV

This example logs pressure data from the PLC every 100 ms into a CSV file.

```
#include "Snap7Helper.h"
#include <fstream>
#include <chrono>
#include <thread>

void RunLogger() {
    Snap7Helper plc;
    if (!plc.Connect("192.168.0.1", 0, 1)) return;

    std::ofstream log("process_data.csv", std::ios::app);
    log << "Timestamp,Pressure\n";

    for (int i = 0; i < 1000; i++) {
        float pressure = plc.GetReal("%MD10");
        log << i << "," << pressure << "\n";
        log.flush();
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
    }

    plc.Disconnect();
}
```

4.2 Recipe Management

```
struct Recipe {
    std::string name;
    int16_t motorSpeed;
    float targetTemp;
};

void LoadRecipe(Snap7Helper& plc, int index) {
    std::vector<Recipe> recipes = {
        {"Standard", 1200, 45.5f},
        {"HighTemp", 800, 80.0f}
    };

    plc.WriteInt("%MW20", recipes[index].motorSpeed);
    plc.WriteReal("%MD30", recipes[index].targetTemp);
}
```

4.3 Object-Oriented Device Wrapper

```
class Motor {
    Snap7Helper* plc;
    std::string startAddr;
    std::string feedbackAddr;

public:
    Motor(Snap7Helper* p, std::string s, std::string f)
        : plc(p), startAddr(s), feedbackAddr(f) {}

    void Start() { plc->WriteBool(startAddr, true); }
    void Stop()  { plc->WriteBool(startAddr, false); }
    bool IsRunning() { return plc->GetBool(feedbackAddr); }
```

```
};
```

4.4 Deterministic Cycle Timing

```
using namespace std::chrono;

void RunLoop(Snap7Helper& plc) {
    auto next = steady_clock::now();
    const auto cycle = milliseconds(20);

    while (true) {
        next += cycle;

        bool start = plc.GetBool("%I0.0");
        if (start) plc.WriteBool("%Q0.0", true);

        std::this_thread::sleep_until(next);
    }
}
```