86LDmicro User Manual

86Duino IDE 502

Software: 86LDmicro 2.0.0

Platform: QEC-M-02 (Vortex86EX2)

IDE: 86Duino Coding 502 Preview (20261105)

Audience: engineers / technicians

Version: v2.0

Release date: November 2026

Revision history

Version	Date	Main changes
v2.0	2026-11	First release for 86LDmicro 2.0.0: IEC 61131-3 (LD, ST, SFC), EtherCAT, Motion86, 86HMI, online monitoring

Contents

Revision history ii
Contents iii
Chapter 1 Introduction 1
1.1 About this manual 1
1.2 About 86LDmicro 3
1.3 86Duino and QEC-M-02 6
1.4 Main differences from LDmicro 1.1 7
Chapter 2 Installation 9
2.1 System requirements 9
2.2 Installing 86Duino Coding 502 10
2.3 Installing the 86LDmicro tools 12
2.4 Hardware connection and COM port 13
2.5 Starting 86LDmicro 16
2.6 Options 22
Chapter 3 Interface tour 24
3.1 Main window and status bar 24
3.2 Main menu 26
3.3 Toolbar 35
3.4 Project tree 36
3.5 POU editors and the side panel 39
3.6 Tool windows 42
3.7 Tabs, side by side, zoom and find 45
Chapter 4 Projects and settings 48
4.1 Project file structure 48
4.2 Project settings 50
4.3 Tasks 52
4.4 Devices 55
4.5 Global variables and data types 58
4.6 I/O mapping 62
4.7 Adding POUs and user code 67
Chapter 5 Quick start: from a blank project to upload 70
5.1 Step 1: Open the 86Duino IDE, create a sketch, select the board and serial port 71
5.2 Step 2: Open 86LDmicro from the IDE and create a new project 72
5.3 Step 3: Get to know the new project: the project tree and the ladder diagram Main 73
5.4 Step 4: Declare the variables in the GVL 74
5.5 Step 5: Insert the NO contact Xstart 75
5.6 Step 6: Add the NC contact Xstop and the coil Rrun in series 76
5.7 Step 7: Add an Rrun contact in parallel to form a latch circuit 77
5.8 Step 8: Add network 2: a TON timer turns on the lamp after a 2-second delay 78
5.9 Step 9: I/O mapping: connect the variables to the physical I/O of the QEC-M-02 80
5.10 Step 10: Check (Shift+F7) and simulate (F5) 81
5.11 Step 11: Build, compile and upload (F8) 84
5.12 Step 12: Online view (Ctrl+F5) 86

Chapter 6 Examples 88
6.0 General principles (read this first) 88
6.1 Start / stop latch 93
6.2 Delayed light off (TOF) 95
6.3 Packing counter (CTU + compare) 97
6.4 Running lights (alternate blinking) 101
6.5 ST: Fill control 103
6.6 SFC: Sequence control 106
6.7 C/C++: Sending a message on Serial1 109
Chapter 7 Ladder diagram instruction reference 111
7.1 Editing 112
7.2 Contacts and coils 118
7.3 Edges and pulses 120
7.4 Timers 122
7.5 Counters 124
7.6 Comparison 125
7.7 Arithmetic and move 126
7.8 IEC function blocks (CALL) and the ST instruction 129
7.9 Program flow 131
7.10 I/O, serial port and others 132
7.11 Motion control instructions 134
Chapter 8 ST and SFC 136
8.1 The ST editor 136
8.2 ST syntax quick reference 141
8.3 Custom FUNCTIONs and FUNCTION_BLOCKs 144
8.4 The SFC editor 148
8.5 Steps, transitions and branches 151
8.6 SFC execution semantics 158
Chapter 9 Simulation and debugging 160
9.1 Simulation 160
9.2 Simulation I/O 164
9.3 Watch, write and force 166
9.4 Trace 169
9.5 Cross reference and search 170
9.6 Common error messages 173
Chapter 10 Building, uploading and the online view 176
10.1 Check, build, compile and upload 176
10.2 Generated files 180
10.3 Serial monitor 182
10.4 Online view 183
10.5 Logbook and watchdog 187
10.6 Uploading the 86HMI resources 190
Chapter 11 EtherCAT 191
11.1 Concepts 191
11.2 The complete workflow 192

11.3 EtherCAT Configurator ... 200
11.4 Reading the ENI and mapping variables ... 218
11.5 SubDevice page ... 222
11.6 PDO online values, SDO and CoE ... 225
11.7 Troubleshooting ... 228
Chapter 12 Motion86 motion control ... 229
12.1 Concepts ... 229
12.2 Creating a machine ... 230
12.3 Machine settings ... 232
12.4 Axis settings ... 234
12.5 Commissioning the machine ... 236
12.6 Commissioning a single axis ... 238
12.7 PLCopen MC function blocks ... 239
12.8 M86 machine function blocks ... 244
12.9 Safety mechanisms ... 248
Chapter 13 86HMI human-machine interface ... 250
13.1 Enabling 86HMI ... 250
13.2 Binding widgets ... 256
13.3 Widgets and how they interact ... 261
13.4 Text, trends, alarms and page switching ... 265
13.5 Simulation and online ... 268
13.6 Resource upload and languages ... 270
Chapter 14 C/C++ and 86Duino integration ... 272
14.1 User code files ... 272
14.2 Accessing PLC variables ... 274
14.3 Hook functions ... 277
14.4 Examples ... 278
14.5 The sketch's .ino ... 282
14.6 Points to note ... 284
Chapter 15 Import, AI, the command line and project protection ... 285
15.1 Importing an LDmicro 1.1 project ... 285
15.2 Importing PLCopen XML ... 288
15.3 AI assistant ... 290
15.4 MCP ... 294
15.5 Command-line tool ... 296
15.6 Project protection ... 298
Chapter 16 Frequently asked questions ... 300
Appendix A Comparison with LDmicro 1.1 ... 307
Appendix B Keyboard shortcuts ... 309
B.1 Main window (any tab) ... 309
B.2 Find bar and project tree ... 310
B.3 Ladder diagram editor ... 310
B.4 SFC editor ... 312
B.5 Code editors (ST, GVL, data types, C/C++, sketch) ... 312
B.6 AI assistant ... 312

Appendix C Variable naming and types 313
C.1 Naming conventions 313
C.2 Naming rules 313
C.3 Data types and ranges 314
C.4 Constants and how to write TIME 315
Contact us 316

Chapter 1 Introduction

1.1 About this manual

This is the complete user manual (version v2.0) for **86LDmicro 2.0.0**. It explains how to install and start 86LDmicro in 86Duino Coding 502 Preview (20261105), how to write PLC programs in the IEC 61131-3 ladder diagram (LD), Structured Text (ST) and Sequential Function Chart (SFC) languages and in C/C++, and how to simulate them on the PC, upload them to the QEC-M-02 controller and monitor them online. EtherCAT, Motion86 motion control and the 86HMI human-machine interface each have a chapter of their own.

This manual is written in English, and the screenshots show the English user interface of 86LDmicro 2.0.0; UI names are written exactly as they appear on the screen, for example Watch, Simulation I/O, Tasks and Data types. The chapters are organized as follows:

Chapter	Contents	For
Chapters 1–2	Introduction, installation (IDE, 86LDmicro, hardware connection, startup, options)	First-time users
Chapters 3–4	Interface tour (every menu, toolbar, project tree, tool windows), projects and settings (tasks, devices, variables, I/O mapping)	First-time users, reference
Chapter 5	Quick start: from a blank sketch to upload and online monitoring in 12 steps	First-time users
Chapter 6	Examples: latch, delayed lamp off, counting, running lights, ST, SFC, C/C++	Practice
Chapters 7–8	Ladder diagram instruction reference, ST and SFC	Programming
Chapters 9–10	Simulation and debugging, build, upload and online monitoring	Debugging, going online
Chapters 11–13	EtherCAT, Motion86 motion control, 86HMI human-machine interface	Advanced features
Chapters 14–15	C/C++ and 86Duino integration; import, AI, command line and project protection	Advanced features
Chapter 16, appendices	FAQ; LDmicro 1.1 comparison, shortcut keys, variable naming and data types	Reference

***Tip: You can try 86LDmicro even before you have a controller: after the installation in sections 2.2 and 2.3, go straight to Steps 1 to 10 of Chapter 5 (you can skip the serial port in Step 1 for now) to create, check and simulate your first program on the PC.**
When you are ready to connect a QEC controller, complete the hardware connection in section 2.4 and Steps 11 and 12 of Chapter 5 (build and upload, online monitoring).

1.1.1 How to read this manual

- **Bold**: buttons, menu items or field names on the screen, for example click **OK**.
- `Monospace`: text to type, variable names, file names or folders, for example `Xstart`, `data\86ld`.
- "Project > Add a PROGRAM (ST)...": click the menu items in this order. Shortcut keys are written as Ctrl+B, Shift+F7.
- `<IDE folder>`: the folder of the 86Duino IDE (the one that contains `86duino.exe`); `<sketch>`: the folder of the 86Duino sketch.
- **Tip**, **Note**, **Important**, **Safety** and **New** boxes: additional explanations, common pitfalls and safety information.

Example projects: the quick-start project `MyPlc` in Chapter 5, the examples in Chapter 6, and the comprehensive example `tools\example\ex30_dispense`.

*__Tip:__ For the latest documentation, see the online documentation site opened from the 86LDmicro "Help" menu, https://www.qec.tw/86duino/86ldmicro/ ; the offline documentation is in `<IDE folder>\tools\LDmicro\tool\docs`.

1.2 About 86LDmicro

86LDmicro 2.0 is the PLC development environment that ICOP built for the 86Duino QEC series controllers.

It keeps the ladder diagram workflow of LDmicro 1.1 and has been redesigned around the IEC 61131-3 project structure (POUs, tasks, global variables, data types), so engineers familiar with PLCs can start right away, and 86Duino users can also control machines the PLC way.

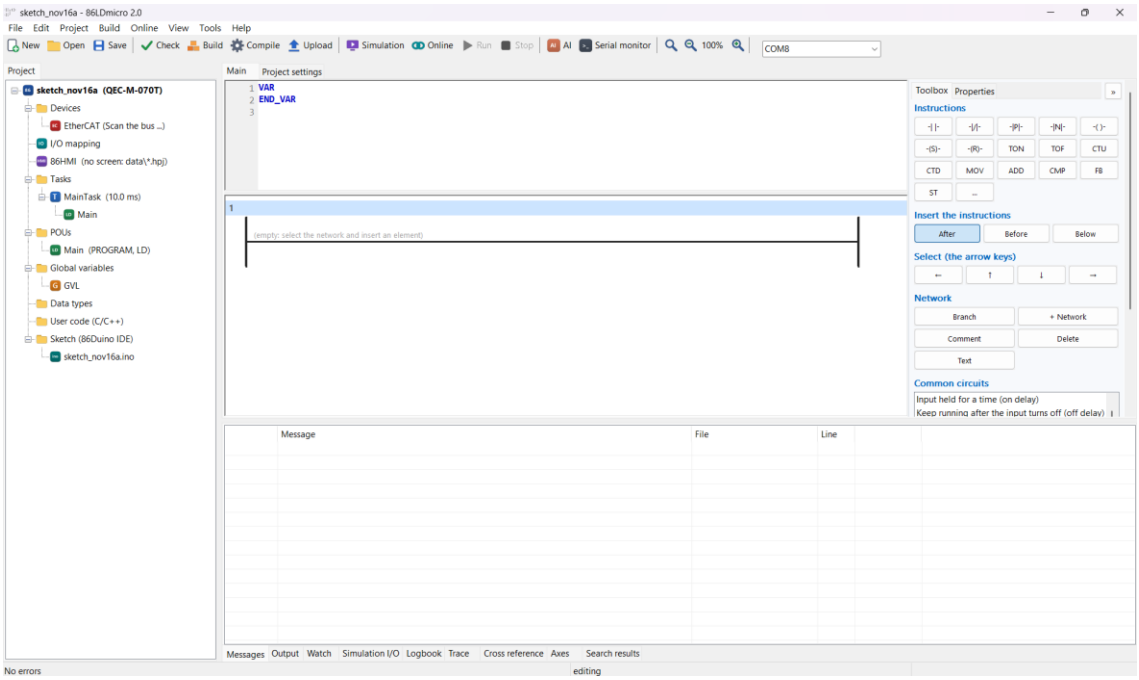


Figure 1-0 86LDmicro 2.0.0 Interface

Unlike a typical C program that runs "line by line", a PLC program runs in "scan cycles":

```
Each cycle: read all inputs (local DI, EtherCAT input PDOs …)
        → execute each PROGRAM (LD / ST / SFC) in the order set in the task
        → write the output variables to DO and EtherCAT output PDOs
        → wait for the next cycle (by default, MainTask runs every 10 ms)
```

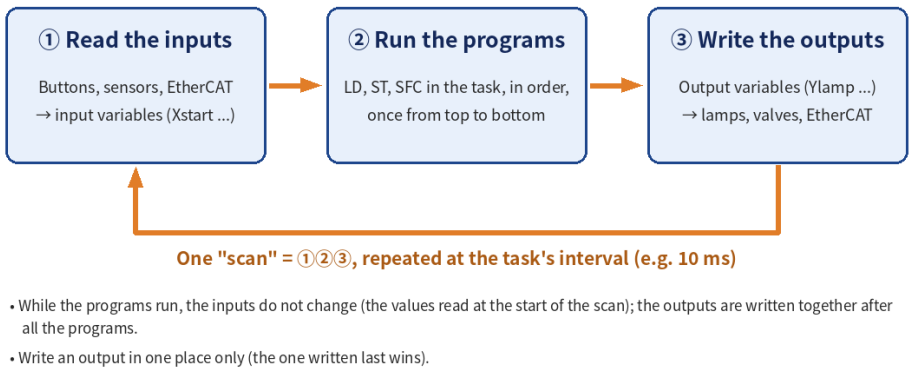


Figure 1-1 One scan cycle: read inputs → execute program → write outputs (schematic)

1.2.1 System architecture

86LDmicro does not interpret the program on the controller. Instead, it converts the whole PLC project into an 86Duino C++ sketch, which is then compiled and uploaded with the compiler of the 86Duino IDE. The PC and the controller are connected with a single USB cable, which is shared by upload and online monitoring:

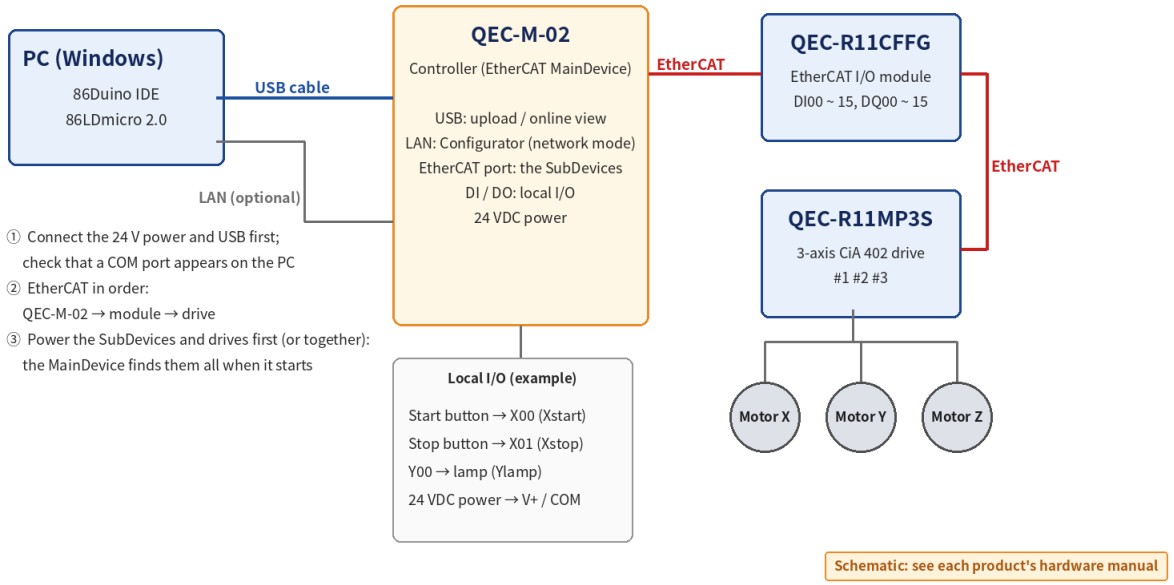


Figure 1-2 Typical system: a PC (86Duino IDE + 86LDmicro) connected over USB to a QEC-M-02, with I/O modules and drives daisy-chained over EtherCAT (schematic)

Step	Who	Output
1. Write	You edit LD / ST / SFC / C/C++ and set up devices and I/O in 86LDmicro	Project files (text files) in `<sketch>\data\86ld\`
2. Build	86LDmicro (Ctrl+B)	In the sketch: `plc86.cpp`, `plc86.h`, `plc86_rt.cpp`, `plc86_ecat.cpp`, etc.
3. Compile	arduino-builder of the 86Duino IDE (F7)	Executable for the controller
4. Upload	v86dude over USB (F8)	The program is written to the QEC, which then restarts
5. Run	86LD run-time on the QEC	Scans the programs at the task interval, reads and writes I/O, EtherCAT, Motion86
6. Monitor	86LDmicro over the same USB cable (Ctrl+F5)	Live values, power flow, forcing, Logbook

Because the end result is a C++ program, PLC programs can be mixed with 86Duino / Arduino libraries (EtherCAT, Motion86, Modbus, Ethernet, Serial, ...), and C/C++ code can read and write PLC variables directly (Chapter 14).

1.2.2 Main features

Category	Features
Programming languages	Ladder diagram (LD), Structured Text (ST), Sequential Function Chart (SFC), C/C++ user code; custom FUNCTIONs and FUNCTION_BLOCKs
Project structure	Tasks (cyclic / freewheeling / EtherCAT cycle), global variable lists (GVL), data types (structures, enumerations, arrays), RETAIN (retained at power-off)
I/O and communication	QEC local DI / DO, EtherCAT (ENI), 86EVA, Modbus RTU / TCP, Ethernet, serial ports
Motion control	Motion86 machines (CiA 402 drives in CSP, up to 9 axes), PLCopen MC_* function blocks, M86_* paths and G-code, commissioning page
Debugging	Full simulation on the PC (including I/O, timers, axes), online monitoring, write / force, power flow, Trace, Logbook, serial monitor
EtherCAT tools	Embedded EtherCAT Configurator, SubDevice pages (live PDO values, write / force), CoE object dictionary
Human-machine interface	86HMI screen binding, interaction settings, simulation window, resource upload
Other	I/O table Excel import / export, project password protection, side-by-side editing, Traditional Chinese / English / Japanese interface, AI assistant and MCP, command-line tools, LDmicro 1.1 and PLCopen XML import

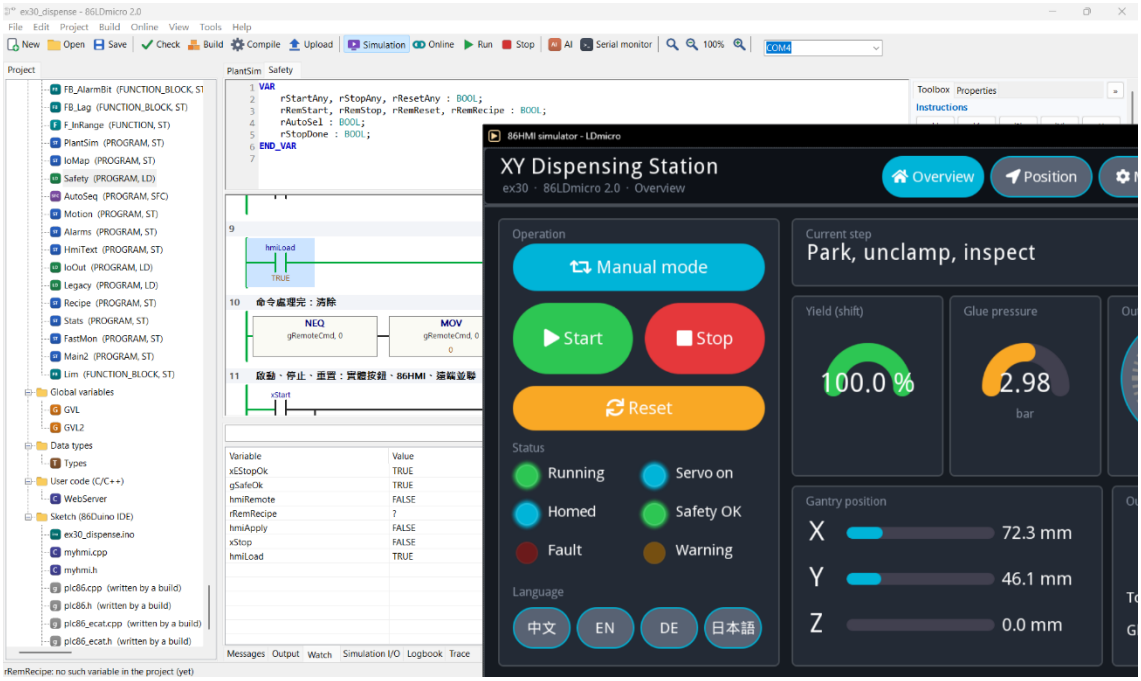


Figure 1-3 86LDmicro 2.0: project tree, ladder diagram and tool windows (during simulation)

1.3 86Duino and QEC-M-02

86Duino is a development platform from DMP based on x86 processors (Vortex86 series) and compatible with the Arduino syntax; the QEC series are industrial controllers built around 86Duino, and the QEC-M-02 is also an EtherCAT MainDevice. The program that 86LDmicro generates is an 86Duino sketch, which is compiled with the 86Duino Coding IDE and runs on the QEC.



Figure 1-4 QEC-M-02 controller

Item	Specification / description
Processor	DMP Vortex86EX2 (32-bit x86)
Development environment	86Duino Coding 502 Preview (20261105), Arduino-compatible syntax; 86LDmicro 2.0.0 is installed in `tools\LDmicro\tool` of the IDE
Local I/O	16 DI points, 16 DO points; in 86LDmicro the channels are `LocalIO.X00` to `X17` (inputs) and `LocalIO.Y00` to `Y17` (outputs), numbered in octal
EtherCAT	EtherCAT MainDevice port: daisy-chains QEC EtherCAT modules (for example QEC-R11CFFG) and CiA 402 drives (for example QEC-R11MP3S)
USB	Program upload, online monitoring, serial monitor, EtherCAT Configurator (USB mode); appears on the PC as a COM port (for example "QEC M02 on COM4")
LAN	EtherCAT Configurator network mode (default 192.168.100.1), Modbus TCP, Ethernet
Serial ports	COM1 to COM3 (RS-232 / 485): Modbus RTU, Serial1 to Serial3 in user code
Power	24 VDC
86LDmicro integration	"Tools > 86LDmicro" in the IDE and the 86LDmicro button on the IDE toolbar (section 2.5)
Other controllers	The "Controller" in the project settings can also be set to QEC-M-01, QEC-M-043T, QEC-M-070T, QEC-M-090T, QEC-M-150T, or other QEC boards in the IDE (section 4.2)

***Safety:** For terminal positions, sink / source wiring and voltage and current ratings, always follow the QEC-M-02 hardware manual.

1.4 Main differences from LDmicro 1.1

The 86Duino IDE used to ship with LDmicro 1.1 (user manual LDmicro UM v1.6.3). 86LDmicro 2.0 keeps the ladder diagram operation and the 1.x instructions, but the project structure, the way settings are made and the debugging tools are different:

Item	LDmicro 1.1	86LDmicro 2.0 (this manual)
Startup	The "Tools" menu of the 86Duino IDE	"Tools > 86LDmicro" in the IDE or the 86LDmicro button on the IDE toolbar; can also be run on its own (section 2.5)
Program structure	One ladder diagram (plus ST blocks)	Multiple POUs (PROGRAM, FUNCTION_BLOCK, FUNCTION), scheduled and executed by tasks
Languages	LD, partial ST	LD, ST, SFC, C/C++ user code
Variables	VAR LIST; the type is determined by the name prefix (X, Y, R, T, C...)	Global variable lists (GVL) + local variables of POUs, IEC types (BOOL, INT, REAL, TIME, STRING, structures, arrays...)
Scan cycle	PLC Cycle Time in Settings → General	Interval of the task; multiple tasks are possible (section 4.3)
Settings	Settings menu: General / Modbus / Virtual Arduino	Project settings, devices, I/O mapping page (Chapter 4)
I/O assignment	Pin Name in the IO List	I/O mapping page: local, EtherCAT, 86EVA, Modbus, pins; can be exported to and imported from Excel (section 4.6)
Code generation	Code Generation button: `ladder.h`, `myplc.cpp`, `myplc.h`, then compile and upload in the IDE	Build (Ctrl+B): `plc86.cpp`, `plc86.h`, etc.; compile (F7) and upload (F8) are done inside 86LDmicro (Chapter 10)
Calls in the .ino	`setupPlc()`/`loopPlc()`	`plc86_setup()` / `plc86_loop()` (replaced automatically on import)
EtherCAT	86EVA virtual devices (86EVA Configurator)	86EVA, or the project's own MainDevice (ENI) + embedded EtherCAT Configurator + SubDevice pages + CoE (Chapter 11)
Motion control	VirtualServo / pulse	Motion86 machines (CiA 402 CSP, up to 9 axes), PLCopen MC function blocks, commissioning page (Chapter 12)
Simulation	Simulate menu: Simulation Mode (F7), Start Real-Time Simulation (F8), Single Cycle (Space)	Simulation (F5) of the whole project (including I/O, timers, axes, the 86HMI simulation window), speed x1 / x10 (Chapter 9)
Debugging	Online monitoring	Online monitoring, write / force, power flow, Trace, Logbook, Cross reference, search in project
Project files	`data\myLD.ld` (single file)	Text files in `data\86ld\` (can be version-controlled with git, section 4.1)
Other	—	Project protection, AI assistant and MCP, command-line tools, PLCopen XML import (Chapter 15)

LDmicro 1.1 projects can be imported directly: when the sketch contains `data\myLD.ld`, 86LDmicro asks whether to import it when you open it from the IDE (section 2.5); you can also use "File > Import a LDmicro 1.x program or a PLCopen XML" (section 15.1). The original `myLD.ld` is not modified.

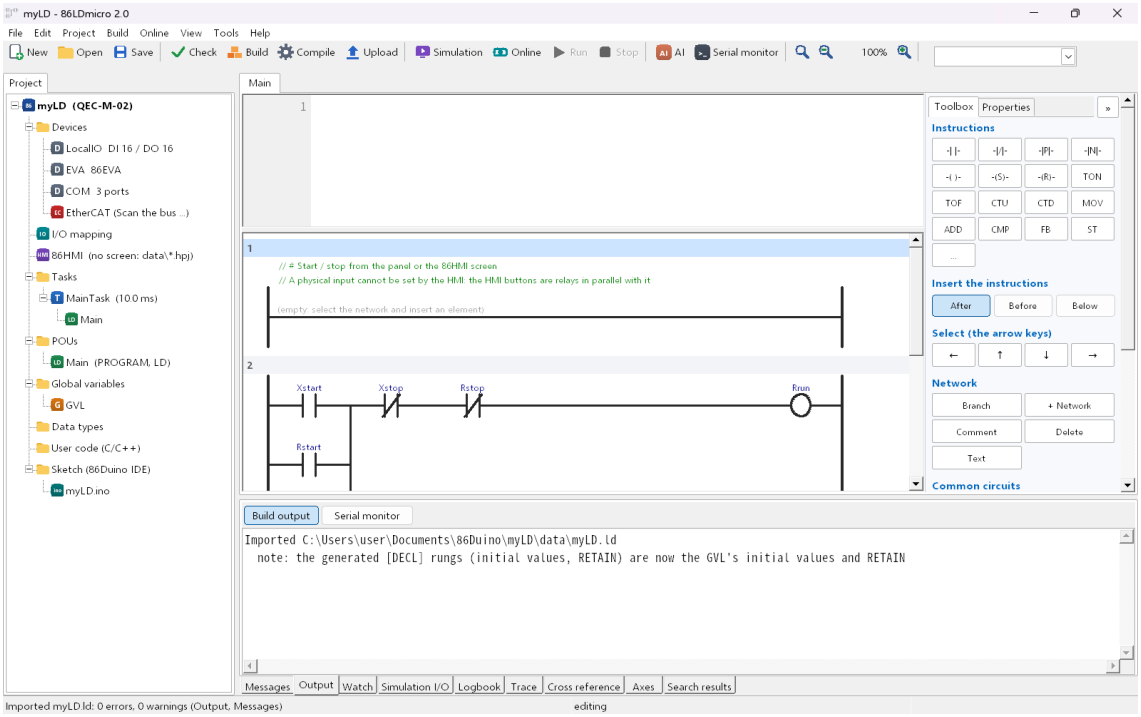


Figure 1-5 A 1.1 project after import: the ladder diagram becomes a POU and the variables go into a GVL

> ***Tip:** 1.x instructions such as timers and counters keep their 1.x semantics (counting in cycles); for new programs, use the IEC function blocks instead, for example `CALL(t1, IN := @, PT := T#1s, Q => @)` (section 7.8).
> To go back to LDmicro 1.1, see the end of section 2.3.

Chapter 2 Installation

This chapter describes the preparation before you start programming: installing the 86Duino IDE and 86LDmicro, connecting the controller and finding its COM port, starting 86LDmicro from the IDE, and the build environment and language settings in "Options".

2.1 System requirements

Item	Requirement
Operating system	Windows 10 / 11 (64-bit)
86Duino IDE	86Duino Coding 502 Preview (20261105), which includes the DMP-ECAT EtherCAT library and the new Motion86 (9 axes)
86LDmicro	86LDmicro 2.0.0 release (zip), extracted to `tools\LDmicro\tool` of the IDE (section 2.3)
Controller	QEC-M-02 (other QEC boards are selected in the project settings, section 4.2)
Connection	A USB cable that carries data (shared by upload, online monitoring and the EtherCAT Configurator); a network cable is also needed when the Configurator works over the network
Optional	Microsoft Edge (for the embedded EtherCAT Configurator screen), Python 3 (for the AI MCP feature, section 15.4)

*__Note:__ Use the 86Duino IDE 502 Preview version **20261105**. Older IDEs lack the EtherCAT functions used by the CoE object dictionary or only have the old Motion86, so building EtherCAT or motion control projects may fail to compile; copying only the new Motion86 into an older IDE also fails to compile.

2.2 Installing 86Duino Coding 502

86LDmicro uses the compiler (arduino-builder), libraries and upload tool (v86dude) of the 86Duino IDE, so install the IDE first.

1. Get 86Duino Coding 502 Preview (20261105) from [the QEC official website](#).

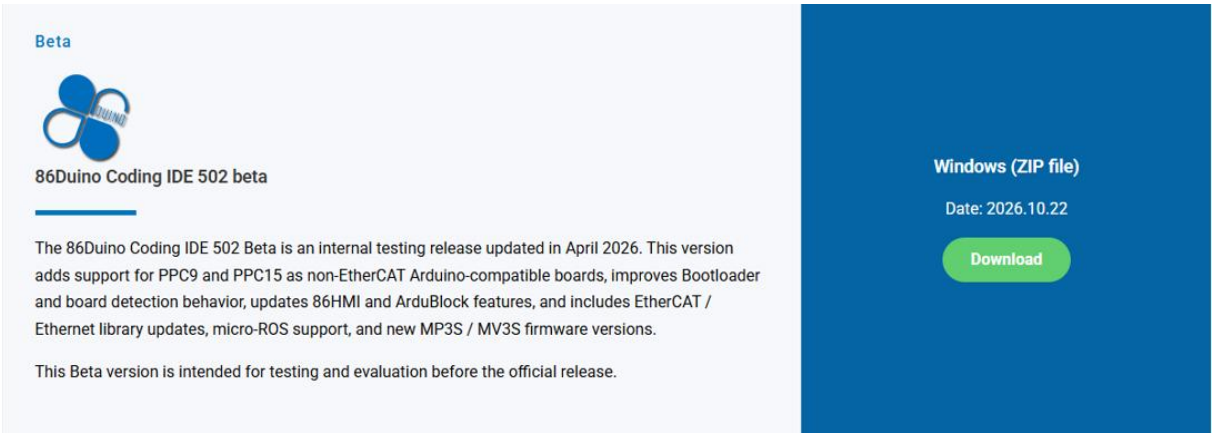


Figure 2-1 Download link for 86Duino IDE 502 on the QEC website

2. Extract the file to a local folder, for example `C:\86Duino_Coding_502_Preview_20261105`. The folder then contains `86duino.exe`, `hardware`, `tools`, etc. This manual calls this folder `<IDE folder>`.
3. Double-click 86duino.exe to open the IDE. Figure 2-2 86Duino IDE folder: 86duino.exe, arduino-builder.exe, drivers, hardware, tools

名稱	修改日期	類型	大小
drivers	2026/11/16 下午 02:00	檔案資料夾	
examples	2026/11/16 下午 02:00	檔案資料夾	
hardware	2026/11/16 下午 02:01	檔案資料夾	
hermes	2026/11/16 下午 02:06	檔案資料夾	
java	2026/11/16 下午 02:06	檔案資料夾	
lib	2026/11/16 下午 02:07	檔案資料夾	
libraries	2026/11/16 下午 02:07	檔案資料夾	
tools	2026/11/16 下午 02:07	檔案資料夾	
tools-builder	2026/11/16 下午 02:07	檔案資料夾	
86duino.exe	2026/11/16 下午 02:00	應用程式	69 KB
86duino_debug.exe	2026/11/16 下午 02:00	應用程式	67 KB
86duino-cli.exe	2026/11/16 下午 02:00	應用程式	36,715 KB
arduino.l4j.ini	2026/11/16 下午 02:00	組態設定	1 KB
arduino_debug.l4j.ini	2026/11/16 下午 02:00	組態設定	1 KB
arduino-builder.exe	2026/11/16 下午 02:00	應用程式	23,401 KB
libusb0.dll	2026/11/16 下午 02:07	應用程式擴充	43 KB
msvcp100.dll	2026/11/16 下午 02:07	應用程式擴充	412 KB
msvcr100.dll	2026/11/16 下午 02:07	應用程式擴充	753 KB
README.md	2026/11/16 下午 02:07	MD 檔案	7 KB
revisions.txt	2026/11/16 下午 02:07	文字文件	97 KB

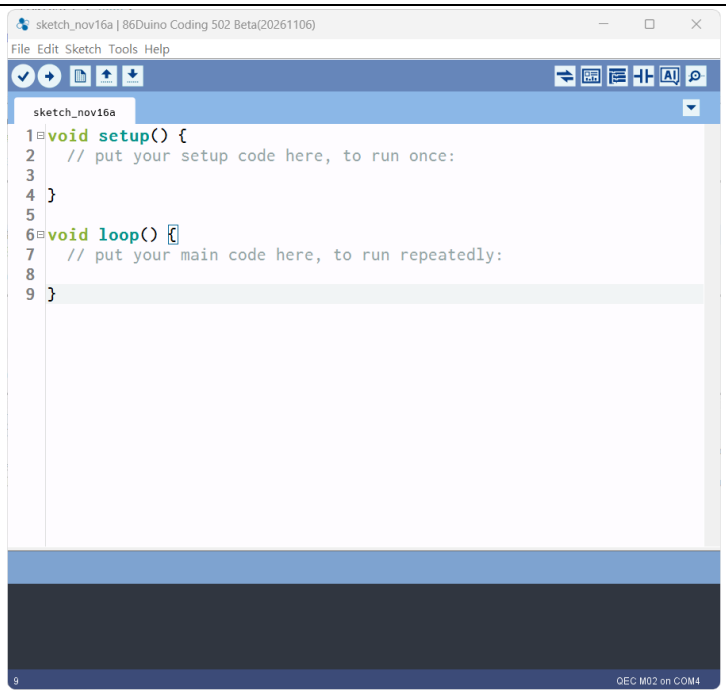


Figure 2-3 86Duino Coding 502 after startup (a new blank sketch)

86LDmicro projects are stored in the sketch folder (`<sketch>\data\86ld`), so the sketch must be saved first: open an existing sketch with "File > Open...", or use "File > New" and then "Save As..." to give it a name. The examples that come with the IDE must first be saved as your own sketch before you can open 86LDmicro.

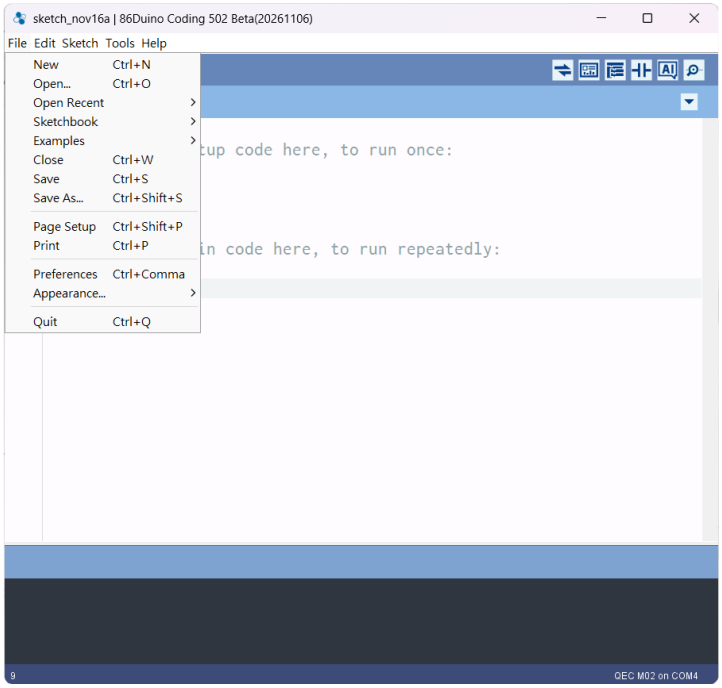


Figure 2-4 The "File" menu of the 86Duino IDE: New, Open, Save As

2.3 Installing the 86LDmicro tools

86LDmicro 2.0.0 is released as a zip file (for example 86LDmicro_2.0.0_v1r2.zip). The top level of the zip is the tool folder, whose contents go into <IDE folder>\tools\LDmicro\tool. If your 86Duino IDE already includes 86LDmicro 2.0.0 (<IDE folder>\tools\LDmicro\tool\VERSION.txt contains 2.0.0 2026-11-11), you can skip this section.

```
<IDE folder>\tools\LDmicro\tool\
    LDmicro.jar          "Tools > 86LDmicro" in the IDE and the IDE toolbar button
    VERSION.txt          Version and date: 2.0.0 2026-11-11 (include it when reporting problems)
    bin\                 86LDmicro.exe (main program), 86ldmicro-cli.exe (command line, section 15.5)
    configurator\        EtherCAT Configurator (Chapter 11)
    docs\                Offline documentation: USER_GUIDE.md, USER_CODE_API.md, ETHERCAT_MOTION.md,
ai\, LICENSE.txt
    hmisim\              86HMI simulation window (section 13.5)
    mcp\                 MCP server for the AI assistant (section 15.4)
```

2.3.1 Installation steps

1. Close the 86Duino IDE.
2. (Recommended) Make a backup copy of the whole <IDE folder>\tools\LDmicro folder, to use if you want to go back to LDmicro 1.1 later.
3. Right-click the downloaded zip file > Properties, check "Unblock" on the "General" tab (if the option is there) and click OK, so that Windows does not block the extracted executables.
4. Extract the zip to <IDE folder>\tools\LDmicro: the tool folder in the zip becomes <IDE folder>\tools\LDmicro\tool; when files with the same name exist, choose "Replace the files in the destination". Do not create an extra folder level (wrong: tools\LDmicro\86LDmicro_2.0.0_v1r2\tool).
5. Check that <IDE folder>\tools\LDmicro\tool contains LDmicro.jar, VERSION.txt, bin, configurator, docs, hmisim and mcp listed above, and that VERSION.txt contains 2.0.0 2026-11-11.
6. Reopen the 86Duino IDE: when 86LDmicro appears in the "Tools" menu, the installation is complete (section 2.5).

2.3.2 Going back to LDmicro 1.1

Close the 86Duino IDE, replace <IDE folder>\tools\LDmicro with the folder you backed up before the installation, and reopen the IDE. 86LDmicro 2.0 projects are stored in data\86ld of the sketch, and importing does not modify data\myLD.ld of 1.1, so both kinds of project can coexist in the same sketch.

2.4 Hardware connection and COM port

The PC and the QEC-M-02 are connected with a USB cable; upload, online monitoring and the serial monitor all use the COM port of this cable on the PC.

Connection	From	To	Purpose
USB	PC	USB port of the QEC-M-02	Program upload, online monitoring, serial monitor, Configurator (USB mode)
EtherCAT	EtherCAT port of the QEC-M-02	IN of the first SubDevice, then OUT → IN of the next one	I/O modules (for example QEC-R11CFFG), CiA 402 drives (for example QEC-R11MP3S)
Local I/O	QEC-M-02 terminal block	Buttons, sensors, indicator lamps, relays	DI / DO of `LocalIO` (for example X00, Y00)
LAN (optional)	PC	LAN port of the QEC-M-02	Configurator network mode (default 192.168.100.1), Modbus TCP
RS-232 / 485 (optional)	External equipment	COM1 to COM3	Modbus RTU, Serial1 to 3 in user code

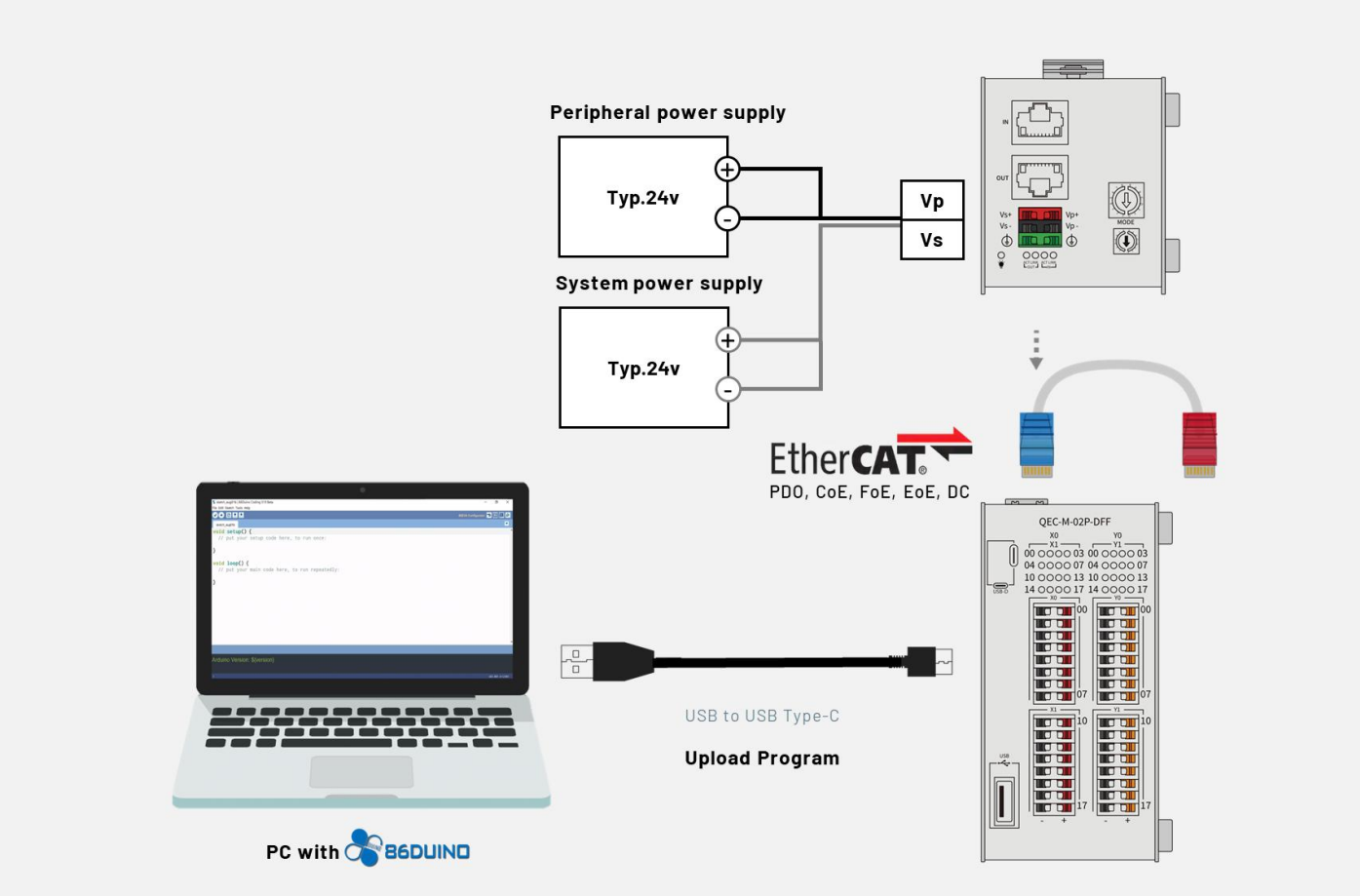


Figure 2-5 Hardware connection between the QEC and the PC (schematic)

***Safety: Switch off the power** before wiring. Add surge suppressors to inductive loads (relays, solenoid valves).
The emergency stop must cut the drive power or STO in hardware; never rely on the PLC program alone.

2.4.1 Checking the COM port

1. Power on the QEC-M-02 and connect it to the PC with the USB cable.
2. Right-click the Windows "Start" button > **Device Manager**.
3. Expand "**Ports (COM & LPT)**", find the newly added "USB Serial Device (COMx)" and note the COM number (COM4 in the figure below). The one that disappears and reappears when you unplug and replug the USB cable is the controller.

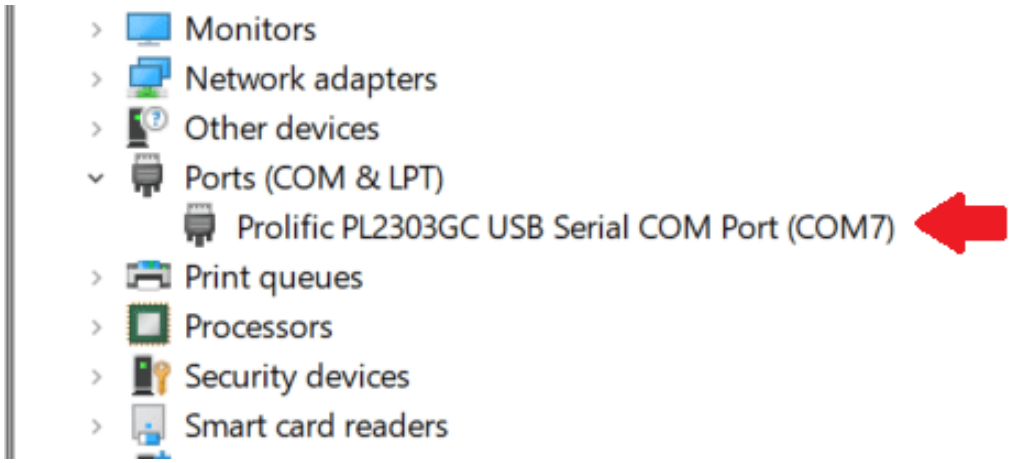


Figure 2-6 Device Manager: USB Serial Device (COM4) under "Ports (COM & LPT)" (schematic)

***Tip:** If no new COM port appears: try another USB cable (one that carries data), another USB port, or check that the controller is powered on.
If a yellow exclamation mark appears: install the driver included with the 86Duino IDE (`<IDE folder>\drivers`), then unplug and replug the cable.

2.4.2 Selecting the board and serial port in the IDE

1. In the 86Duino IDE, click "**Tools > Board**" and select **QEC M02**.

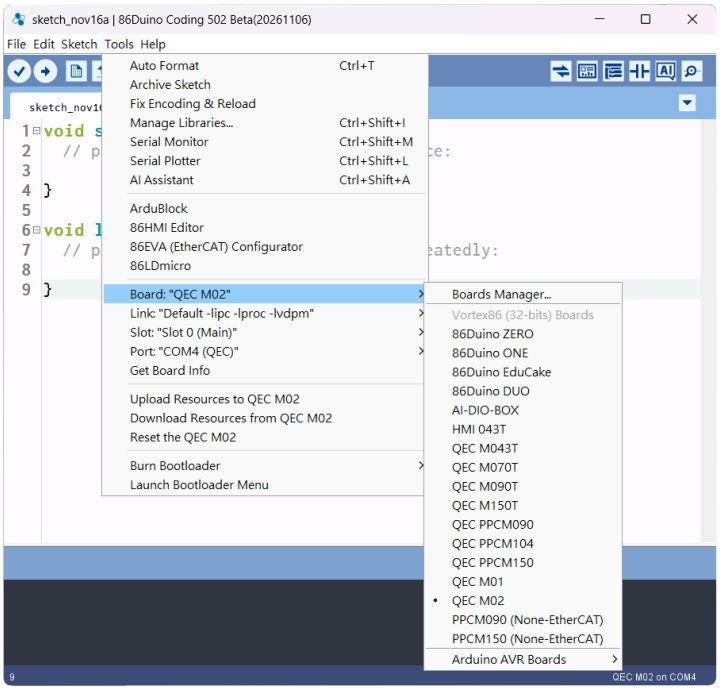


Figure 2-7 Selecting QEC M02 in "Tools > Board"; "86LDmicro" is in the same menu

2. Click "**Tools > Port**" and select the COM port found above (for example COM4). The lower right corner of the IDE shows "QEC M02 on COM4".

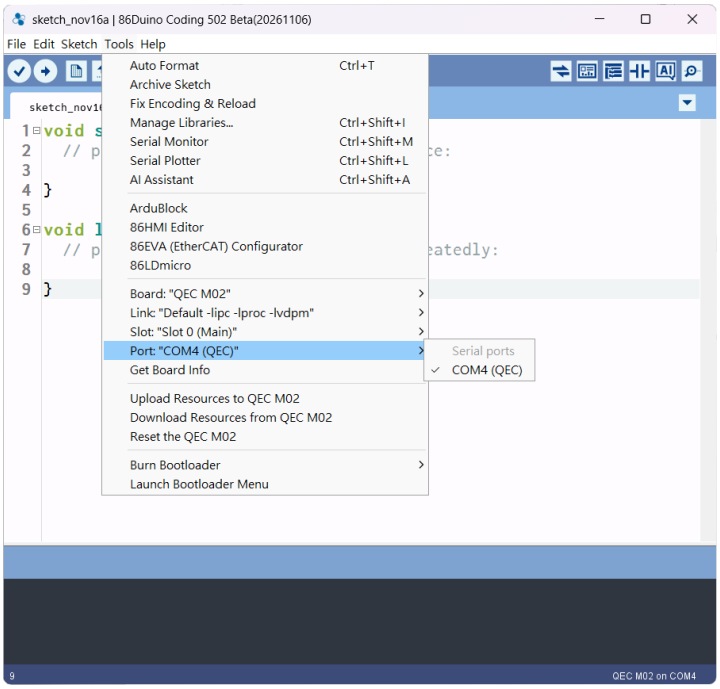


Figure 2-8 Selecting COM4 in "Tools > Port" (schematic)

When 86LDmicro is started from the IDE, it uses the board and serial port selected in the IDE (section 2.5); in 86LDmicro you can also change the port in the serial port box on the right of the toolbar (section 3.3).

#	First connection check	How to check
1	The controller is powered on	The power LED is on
2	The PC recognizes the COM port	USB Serial Device (COMx) appears in Device Manager
3	The board and port in the IDE are correct	The lower right corner of the IDE window shows "QEC M02 on COMx"
4	No other program is using the COM port	Close the serial monitor window of the IDE and other terminal programs
5	86LDmicro can upload and go online	The quick-start program in Chapter 5 uploads successfully, and after Ctrl+F5 the status bar shows "ONLINE COMx: RUN"

2.5 Starting 86LDmicro

We recommend starting 86LDmicro from the 86Duino IDE: 86LDmicro opens (or creates) the project of that sketch and automatically uses the compiler of this IDE and the serial port selected in the IDE.

2.5.1 Starting from the 86Duino IDE (recommended)

1. Open (or create and save) a sketch in the IDE.
2. Check that "Tools > Board" is a QEC board and "Tools > Port" is the correct COM port (section 2.4).
3. Click "**Tools > 86LDmicro**", or the 86LDmicro button on the right of the IDE toolbar (its tooltip shows "86LDmicro").

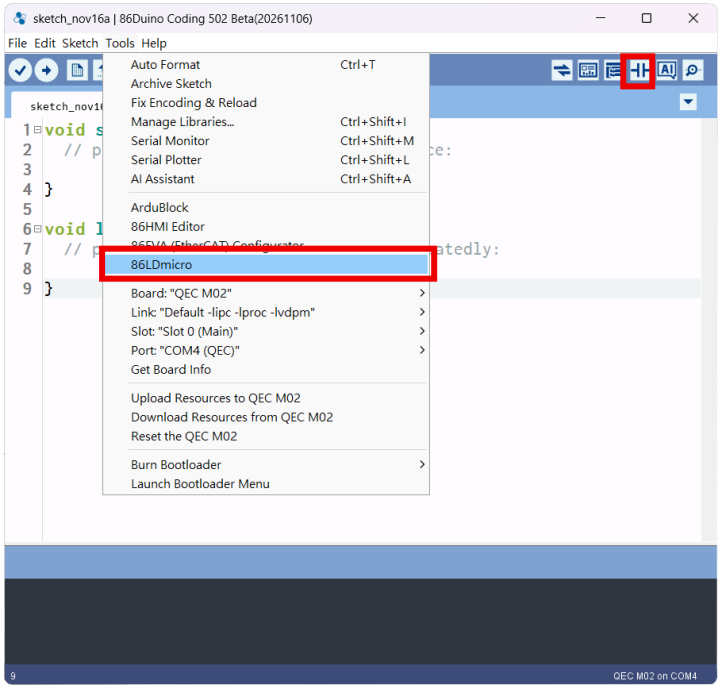
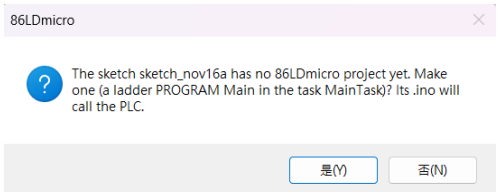
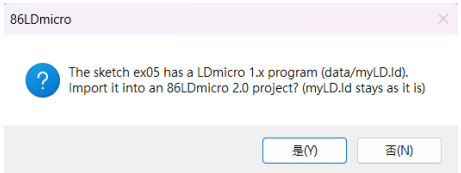
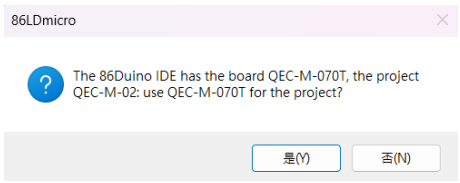


Figure 2-9 The IDE toolbar and the 86LDmicro button on the right

86LDmicro acts automatically according to the state of the sketch:

State of the sketch	What 86LDmicro does	86LDmicro window
No project	Asks "The sketch … has no 86LDmicro project yet. Create one?"; click **Yes** to create a new project (ladder diagram PROGRAM `Main`, executed in the task `MainTask`)	 Figure 2-10 The question when the sketch has no project yet
Has a LDmicro 1.1 `data\myLD.ld`	Asks whether to import it into an 86LDmicro 2.0 project; click **Yes** to import (`myLD.ld` stays unchanged, section 15.1)	 Figure 2-11 The question when the sketch has a LDmicro 1.1 program (data/myLD.ld)

State of the sketch	What 86LDmicro does	86LDmicro window
Already has a 2.0 project (`data\86ld\project.86ld`)	Opens it directly; if the board selected in the IDE differs from the project, asks whether the project should use the board of the IDE	 Figure 2-12 The question when the board selected in the IDE differs from the project
86LDmicro is already open for the same sketch	Brings that window to the front	

After a new project is created, the 86LDmicro main window looks like this: the project tree already contains the task `MainTask` (10 ms), the ladder diagram `Main` and the global variable list `GVL`, and `Main` is opened automatically. Each area is described in Chapter 3.

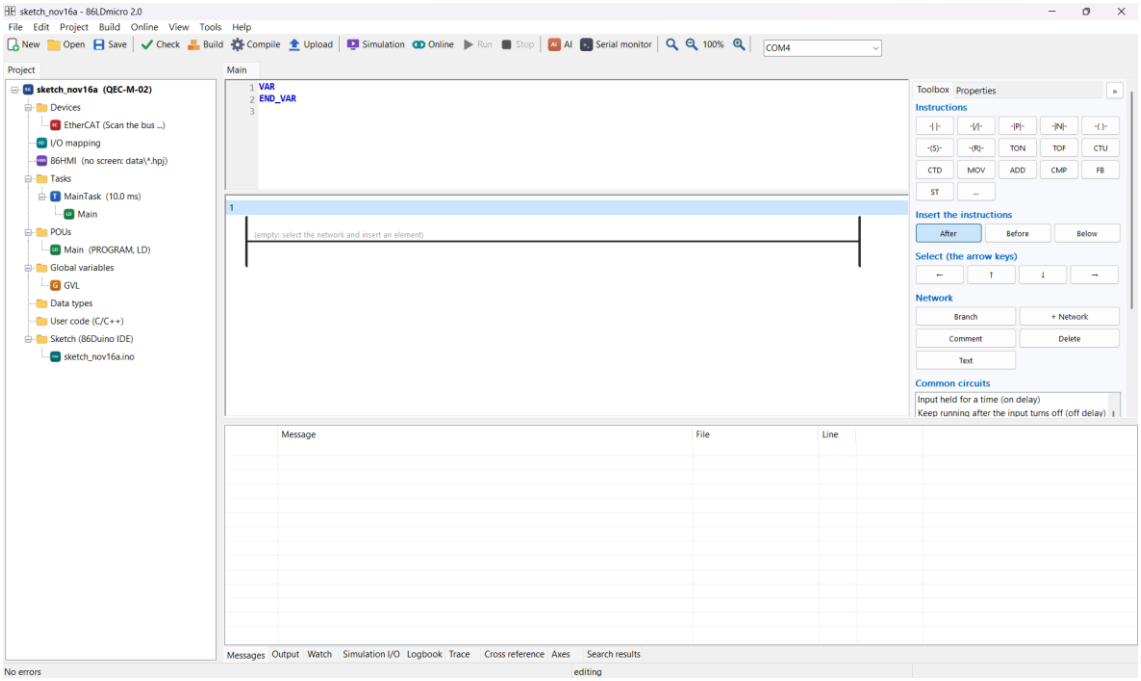


Figure 2-13 86LDmicro after creating a new project from the IDE: MainTask, Main and GVL are already created

- `.ino` gets `#include "plc86.h"`, `plc86_setup()` and `plc86_loop()` (the 1.1 `setupPlc()` / `loopPlc()` are replaced); the rest of the code is not touched.

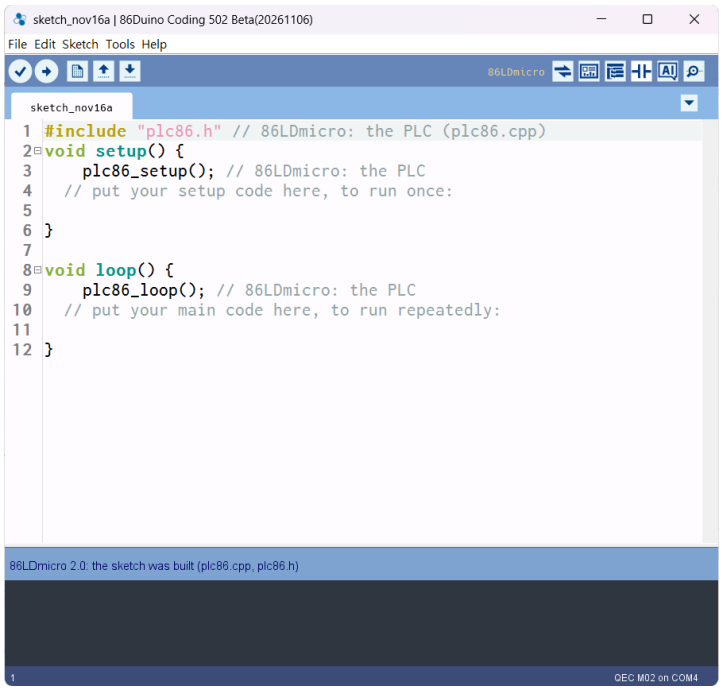


Figure 2-14 The .ino with the 86LDmicro code added.

- After a build, 86LDmicro tells the IDE to reload the sketch, and the IDE shows tabs such as `plc86.cpp`; from then on you can also compile and upload directly in the IDE. Do not edit these generated files by hand (they are rewritten on every build).

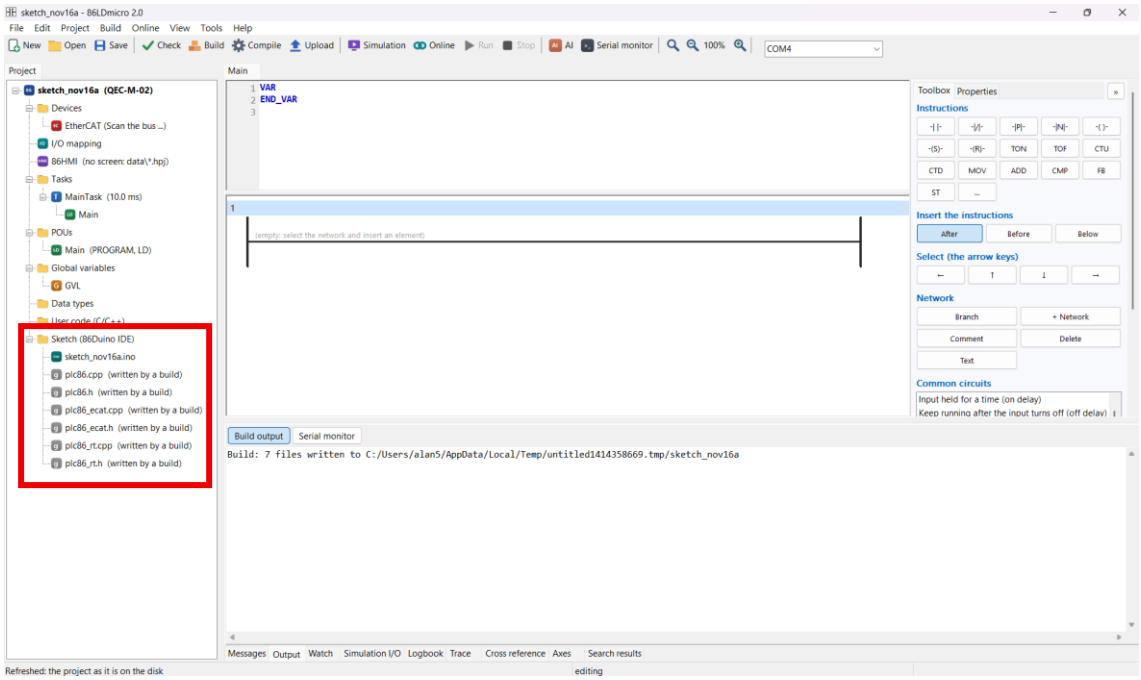


Figure 2-15 Files such as plc86 generated by an 86LDmicro build

If tabs such as `plc86.cpp` do not appear in Sketch (86Duino IDE) in 86LDmicro right after a build, click Refresh (Ctrl + R) in the View toolbar.

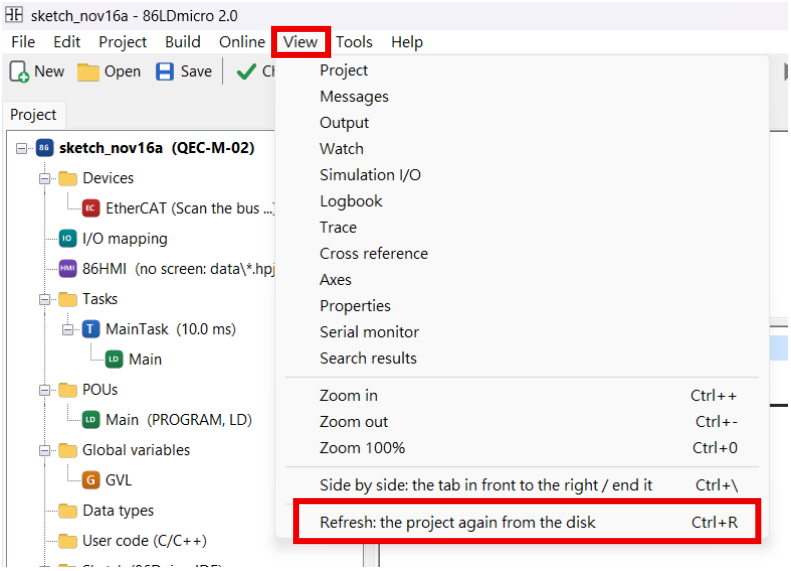
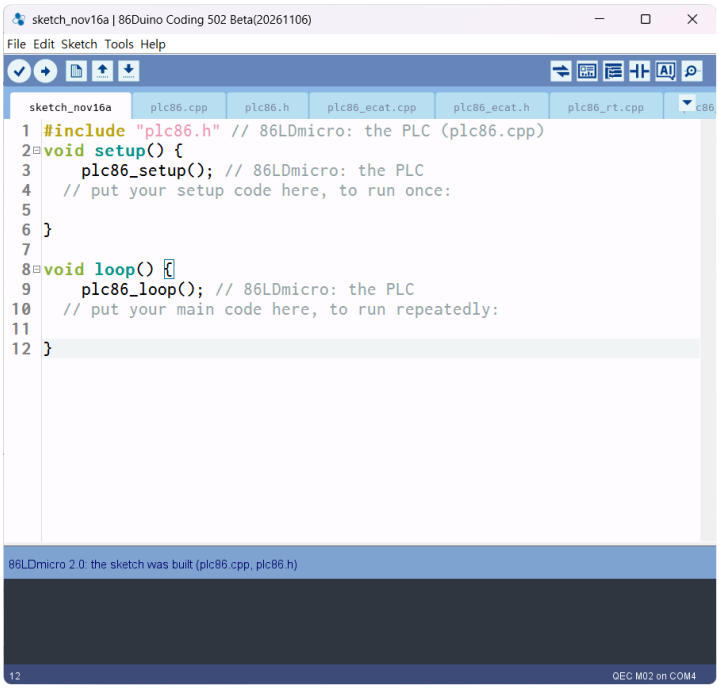


Figure 2-16 After Refresh (Ctrl+R), files such as plc86.cpp appear under Sketch

- When the board is not a QEC board, the status bar at the bottom of the IDE shows "86LDmicro 2.0: choose a QEC board (Tools > Board)" and 86LDmicro does not start; when the sketch is in the examples folder of the IDE, it shows "save the example as a sketch of your own first (File > Save As)".



- Figure 2-17 The 86Duino IDE after a build: tabs such as plc86.cpp, plc86.h, plc86_ecat.cpp

*__Note:__ Unsaved changes to the `.ino` in the IDE are overwritten when the sketch is reloaded after an 86LDmicro build. Save in the IDE first, or edit the `.ino` under "Sketch (86Duino IDE)" in the 86LDmicro project tree instead (section 3.4).

2.5.2 Starting on its own

You can also run `<IDE folder> \tools\LDmicro\tool\bin\86LDmicro.exe` directly.

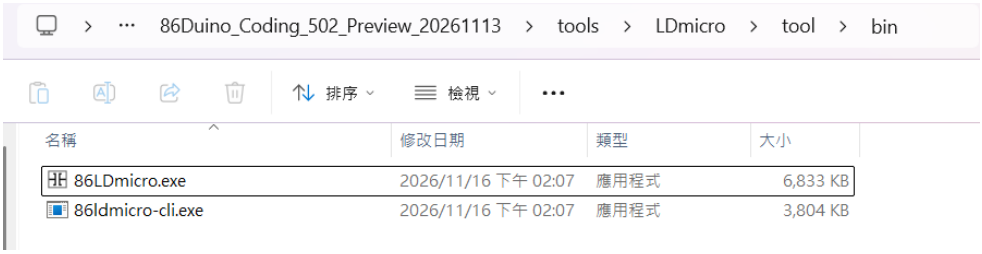


Figure 2-18 Path of 86LDmicro started on its own, with no project open

When started on its own, 86LDmicro automatically opens the project that was opened last; if there is none, the project tree shows "No project: File > New, Open or Import".

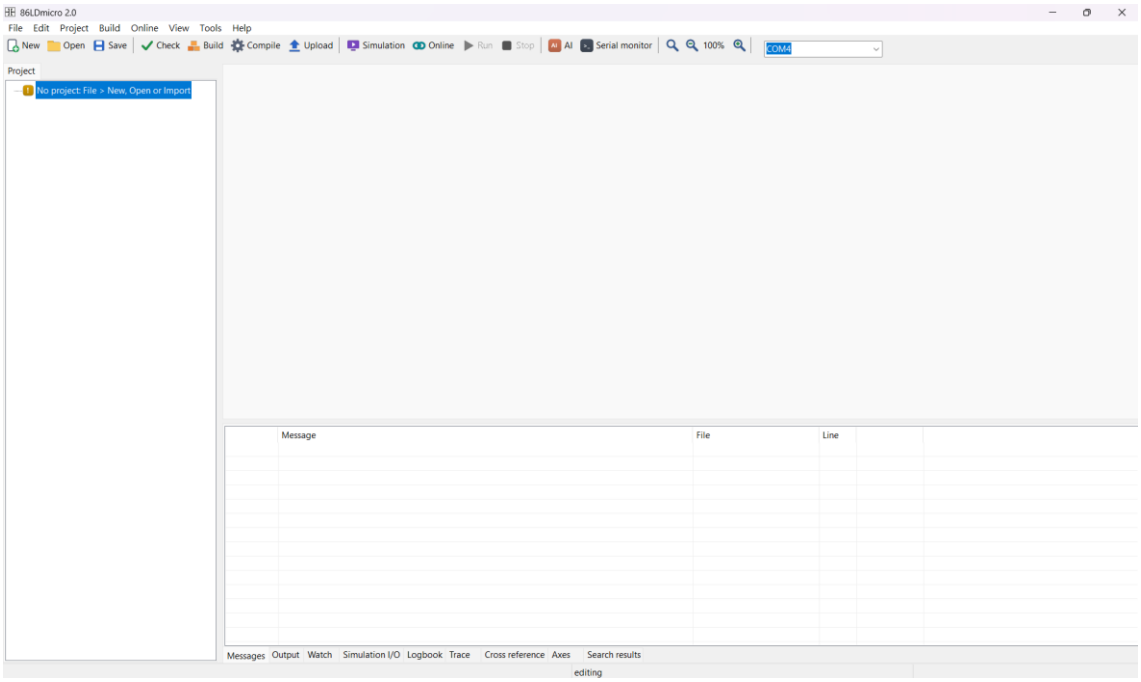
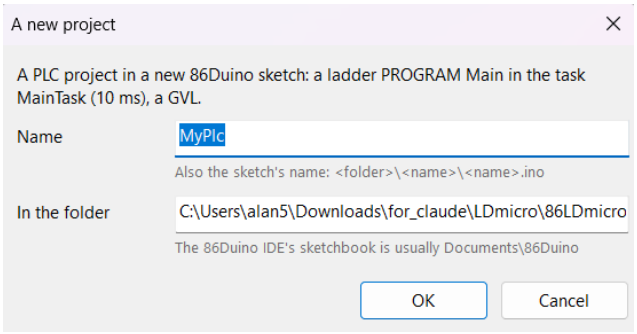


Figure 2-19 86LDmicro started on its own, with no project open

Task	Operation
Create a new project	"File > New project..." (Ctrl+N): enter the name (also the name of the sketch) and the location (by default `Documents\86Duino`, the sketchbook of the IDE) and click **OK**. This creates `<location>\<name>\<name>.ino` and the project, with the ladder diagram `Main`, the task `MainTask` (10 ms) and `GVL`  Figure 2-20 "File > New project...": the name is also the name of the sketch

Task	Operation
Open a project	"File > Open project..." (Ctrl+O): select `data\86ld\project.86ld` or the `.ino` of the sketch
Import	"File > Import a LDmicro 1.x program or a PLCopen XML (.ld, .xml)...", or drag and drop an `.ld` / `.xml` file onto the 86LDmicro window (Chapter 15)
Recent projects	"File > Recent projects" lists the 8 most recently opened projects

***Important:** When you start 86LDmicro on its own for the first time, first set the 86Duino IDE folder in "Tools > Options..." (section 2.6); otherwise you can only check, build and simulate, not compile and upload.

2.6 Options

"**Tools > Options...**" sets the build environment, the interface language and the reconnection after upload for 86LDmicro on this PC (the settings of the project itself are in "Project settings", section 4.2). The **Options...** button on the project settings page opens the same window.

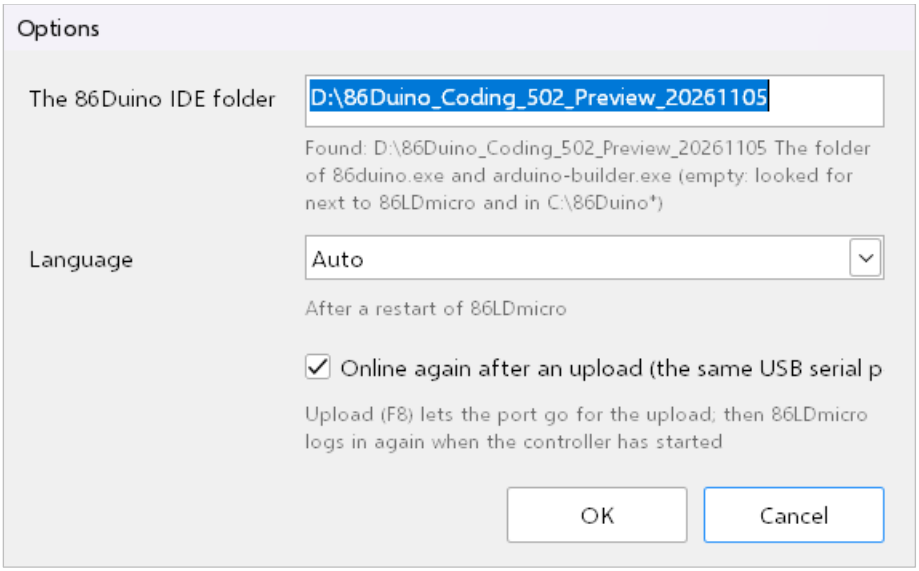


Figure 2-21 Options: 86Duino IDE folder, language, online again after an upload

Field	Description
The 86Duino IDE folder	Build environment: the folder that contains `86duino.exe` and `arduino-builder.exe`. Below it, "Found: ..." or "Not found yet." is shown; when empty, 86LDmicro searches next to itself and in `C:\86Duino*`. When started from the IDE, it is changed automatically to the folder of that IDE. If the folder entered is not an 86Duino IDE, clicking OK shows "There is no 86Duino IDE in this folder"
Language	Auto (follows Windows), English, Traditional Chinese, Japanese; **takes effect after restarting 86LDmicro**
Online again after an upload (the same USB serial port, as in LDmicro 1.1)	Checked by default. When uploading (F8), the serial port is released first, and after the controller starts, 86LDmicro automatically logs in to online monitoring again

2.6.1 Choosing the build environment

- **When the PC has several 86Duino IDEs**: the compiler and libraries of the IDE from which you clicked "Tools > 86LDmicro" are used; when started on its own, the folder set in "Options" is used.
- **Controller (board)**: selected in the project settings (section 4.2); 86LDmicro uses it to find the FQBN in the IDE (for example `86duino:x86:86DuinoQECM02`); for special needs you can enter it directly in "FQBN (optional)" in the project settings.
- **Libraries**: the libraries in `hardware\86duino\x86\libraries` of that IDE (Ethercat, Motion86, Modbus, ...) are used. For EtherCAT and Motion86 projects, use the libraries included with 502 Preview 20261105.
- When the IDE cannot be found, compiling or uploading shows "The 86Duino IDE is not found: Tools > Options > the 86Duino IDE folder".

2.6.2 Interface language

Select it in "Language" and restart 86LDmicro to switch.

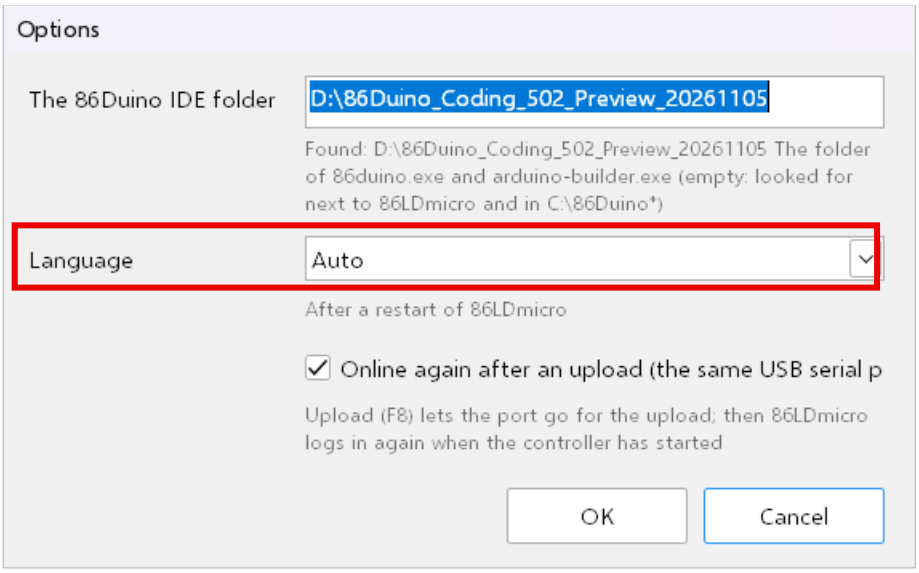


Figure 2-22 Options: 86Duino IDE folder, language, online again after an upload

The menu positions are exactly the same in all three languages; the screenshots in this manual use the English interface.

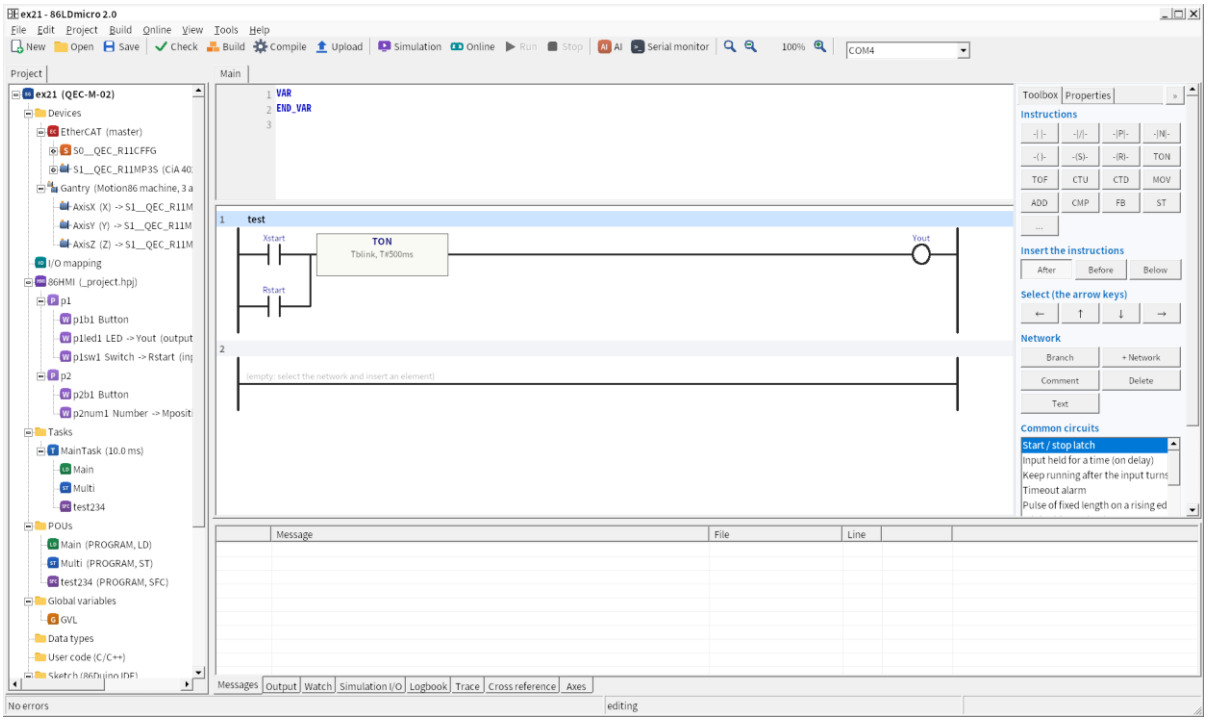


Figure 2-23 The interface with English selected (menu positions are the same as in Traditional Chinese)

Chapter 3 Interface tour

This chapter introduces the 86LDmicro main window piece by piece: every item of every menu, the toolbar, the project tree, the POU editors and the side panel, the tool windows at the bottom, and tabs, side-by-side view, zoom and find. The detailed operation of each feature is in later chapters, indicated by section numbers in the tables.

3.1 Main window and status bar

From top to bottom, the main window consists of the menu bar and the toolbar (with the serial port box on the right); in the middle are the project tree on the left, the document area in the center (one tab per POU or settings page) and the side panel of the POU on the right; below are the tool windows, and at the very bottom the status bar.

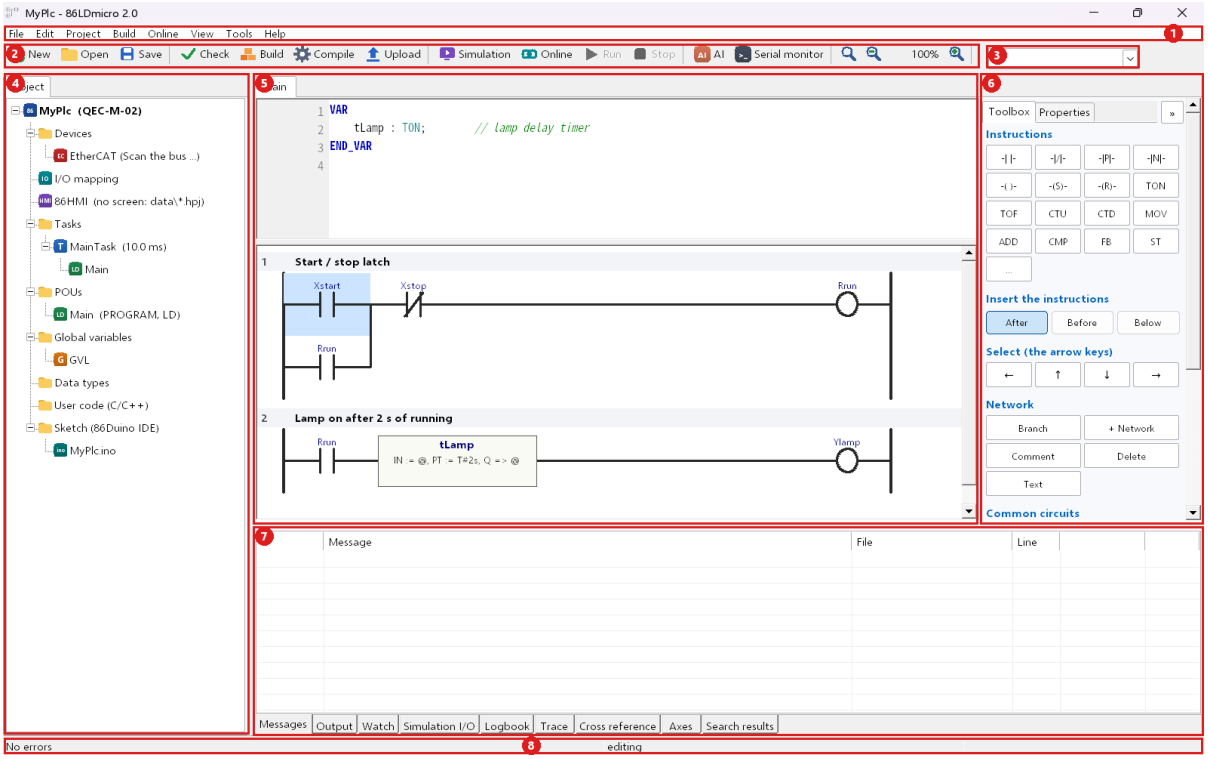


Figure 3-1 Areas of the 86LDmicro main window

No.	Area	Description
1	Menu bar	File, Edit, Project, Build, Online, View, Tools, Help (section 3.2)
2	Toolbar	Frequently used commands (section 3.3); when the window gets narrow, it automatically shows icons only or splits into two rows
3	Serial port	The COM port used by upload, online monitoring, the serial monitor and the USB version of the Configurator
4	Project tree	Everything in the project; double-click to open, right-click for the menu of that node (section 3.4)
5	Document area	One tab for each POU and settings page; can be shown side by side (section 3.7)
6	Side panel	Appears only for LD, ST and SFC tabs: "Toolbox" and "Properties" (section 3.5)
7	Tool windows	Messages, Output, Watch, Simulation I/O, Logbook, Trace, Cross reference, Axes (commissioning), Search results (section 3.6)

No.	Area	Description
8	Status bar	Messages, mode, scan information (see the table below)

The title bar shows "project name - 86LDmicro 2.0"; when the project has unsaved changes, a * follows the name.
The status bar has three fields, from left to right:

Position	Contents	Examples
Left	Result of the last operation, number of errors found by the check	No errors / No errors, 2 warnings / 3 errors, 0 warnings / Saved
Center	Current mode	Editing, Compiling ..., SIMULATION: RUN, SIMULATION: STOP, ONLINE COM4: RUN, ONLINE COM4: STOP
Right	Scan information during simulation or online	Simulation: "1520 scans, 15 s"; online: "scan 230 us (max 410)" (first task)



Figure 3-2 The status bar during simulation: No errors, SIMULATION: RUN, number of scans and time

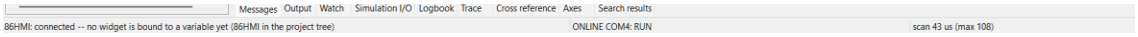


Figure 3-3 The status bar online: ONLINE COM4: RUN, scan time

3.2 Main menu

The main menu has eight menus: File, Edit, Project, Build, Online, View, Tools and Help. The letter in brackets after a menu name can be used with Alt, for example Alt+F opens "File". Shortcut keys work in any tab; items that cannot be used during simulation or online are grayed out. The tables below list all items in each menu from top to bottom (item names are given without the letter in brackets).

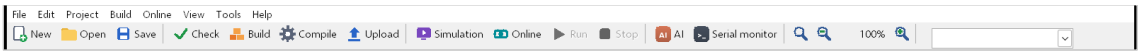


Figure 3-4 Menu bar and toolbar

3.2.1 File

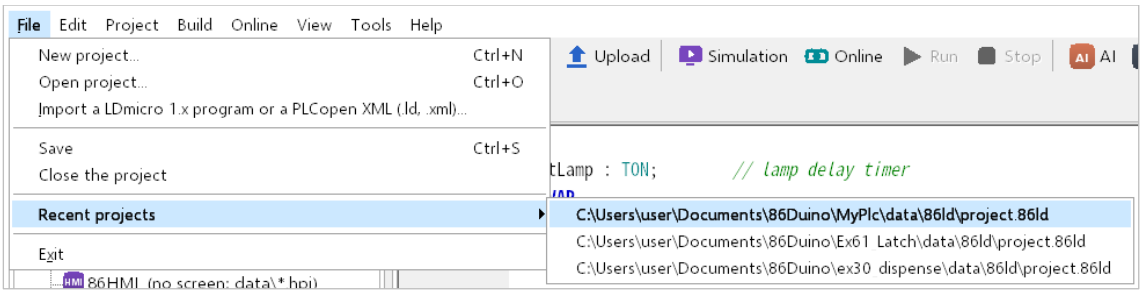


Figure 3-5 The "File" menu

Item	Shortcut	Purpose
New project...	Ctrl+N	Creates a project in a new 86Duino sketch: name, location (section 2.5)
Open project...	Ctrl+O	Opens a project: select `project.86ld` or the `.ino` of the sketch
Import a LDmicro 1.x program or a PLCopen XML (.ld, .xml)...	—	Imports a LDmicro 1.x `.ld` or a PLCopen XML as a new project (sections 15.1, 15.2); you can also drag and drop the file onto the window
Save	Ctrl+S	Saves all changes of the project (including sketch files edited in 86LDmicro); the status bar shows "Saved"
Close the project	—	Closes the project; when there are unsaved changes, asks "Save the changes of the project?"
Recent projects ▸	—	The 8 most recently opened projects; click one to open it; shows "(none)" when there are none
Exit	—	Exits 86LDmicro (asks first when there are unsaved changes)

3.2.2 Edit

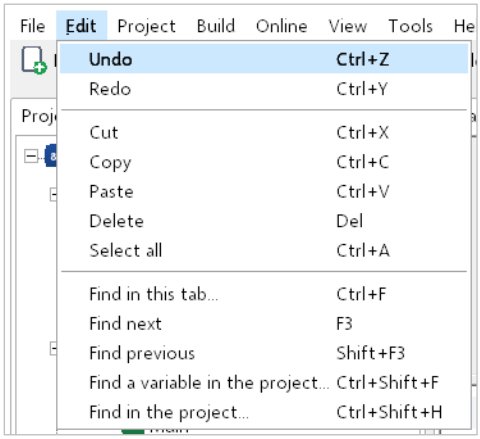


Figure 3-6 The "Edit" menu

Item	Shortcut	Purpose
Undo	Ctrl+Z	Undoes the last change in the current tab
Redo	Ctrl+Y	Redoes an undone change
Cut	Ctrl+X	Code: the selected text; ladder diagram: the selected elements, or the whole network when no element is selected
Copy	Ctrl+C	Same as above, copies
Paste	Ctrl+V	Pastes; in a ladder diagram, at the selected position
Delete	Del	Deletes the selected text, elements or networks
Select All	Ctrl+A	Selects all text in the current tab
Find in this tab...	Ctrl+F	Opens the find bar at the top of the current tab (section 3.7)
Find next	F3	Jumps to the next match
Find previous	Shift+F3	Jumps to the previous match
Find a variable in the project...	Ctrl+Shift+F	Enter a variable name to list in "Cross reference" where it is read, written and declared (section 9.5)
Find in the project...	Ctrl+Shift+H	Searches the whole project for any text; the results are listed in "Search results" (sections 3.7, 9.5)

3.2.3 Project

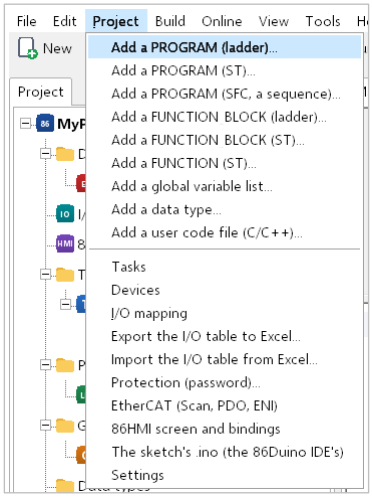


Figure 3-7 The "Project" menu

Item	Shortcut	Purpose
Add a PROGRAM (ladder)...	—	Adds a ladder diagram PROGRAM; it is added to the first task automatically (section 4.7)
Add a PROGRAM (ST)...	—	Adds an ST PROGRAM (Chapter 8)
Add a PROGRAM (SFC, a sequence)...	—	Adds an SFC PROGRAM (section 8.4)
Add a FUNCTION_BLOCK (ladder)...	—	Adds a ladder diagram function block
Add a FUNCTION_BLOCK (ST)...	—	Adds an ST function block (section 8.3)
Add a FUNCTION (ST)...	—	Adds an ST function (section 8.3)
Add a global variable list...	—	Adds a GVL (section 4.5)
Add a data type...	—	Adds a data type file (structures, enumerations, arrays, section 4.5)
Add a user code file (C/C++)...	—	Adds a `.cpp86` user code file (Chapter 14)
Tasks	—	Opens the Tasks page (section 4.3)
Devices	—	Opens the Devices page (section 4.4)
I/O mapping	—	Opens the I/O mapping page (section 4.6)
Export the I/O table to Excel...	—	Saves the variables and I/O mapping as .xlsx or .csv (section 4.6)
Import the I/O table from Excel...	—	Declares variables and sets the I/O mapping from an .xlsx or .csv (section 4.6)
Protection (password)...	—	Makes files read-only or hidden with a password (section 15.6)
EtherCAT (Scan, PDO, ENI)	—	Opens the EtherCAT page (Chapter 11)
86HMI screen and bindings	—	Opens the 86HMI page (Chapter 13)
The sketch's .ino (the 86Duino IDE's)	—	Opens and edits the `.ino` of the sketch in 86LDmicro (section 14.5)
Settings	—	Opens the project settings page (section 4.2)

3.2.4 Build

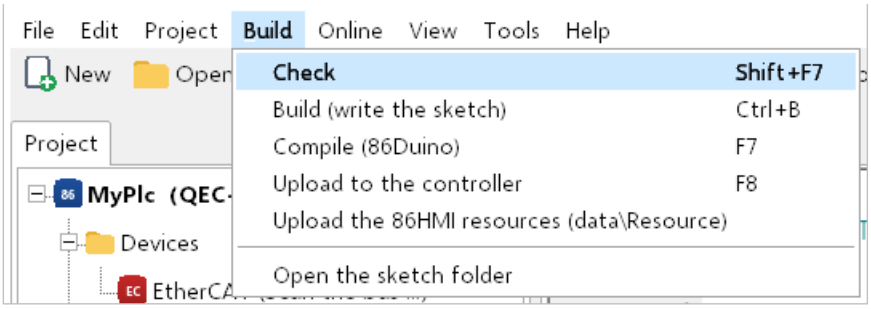


Figure 3-8 The "Build" menu

Item	Shortcut	Purpose
Check	Shift+F7	Only compiles the PLC program and lists errors and warnings (in the "Messages" window) without writing files (section 10.1)
Build (write the sketch)	Ctrl+B	Saves first, then writes the PLC program into the sketch as `plc86.cpp`, `plc86.h`, etc. (sections 10.1, 10.2)
Compile (86Duino)	F7	Builds and then compiles with the compiler of the 86Duino IDE; the progress is shown in "Output" (no upload)
Upload to the controller	F8	Builds, compiles and uploads to the COM port selected on the toolbar (section 10.1)
Upload the 86HMI resources (data\Resource)	—	Uploads only the 86HMI resources such as pictures and fonts, without recompiling the program (section 10.6)
Open the sketch folder	—	Opens the sketch folder in File Explorer

*__Tip:__ While compiling or uploading is in progress, "Compile", "Upload", "Upload the 86HMI resources" and "Login to the controller" are grayed out until it finishes.

3.2.5 Online

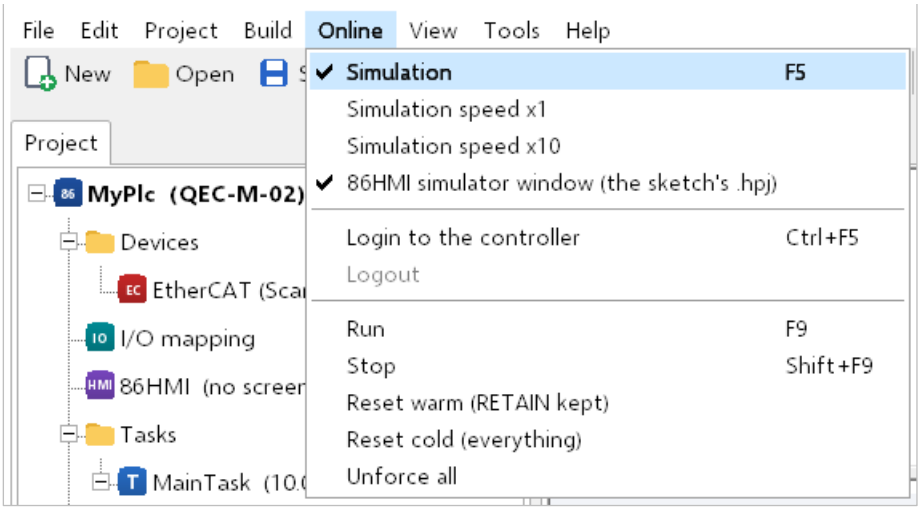


Figure 3-9 The "Online" menu (during simulation)

Item	Shortcut	Purpose
Simulation	F5	Runs the whole project on the PC; click again to end the simulation. Checked during simulation (section 9.1)
Simulation speed x1	—	The simulation runs in real time
Simulation speed x10	—	Simulation time advances 10 times faster (timers etc. act on simulation time)
86HMI simulator window (the sketch's .hpj)	—	When checked, the simulation also opens the 86HMI simulator window (section 13.5)
Login to the controller	Ctrl+F5	Logs in to the controller through the COM port on the toolbar for online monitoring; click again while online to disconnect (section 10.4)
Disconnect	—	Ends online monitoring (available only while online)
Run	F9	Starts the PLC (simulation or controller)
Stop	Shift+F9	The program stops executing; inputs and outputs are still updated, the outputs keep their last values; all Motion86 machines stop (section 10.4)
Reset warm (RETAIN kept)	—	Restarts after confirmation: variables return to their initial values, RETAIN variables are kept
Reset cold (everything)	—	Restarts after confirmation: all variables (including RETAIN) return to their initial values
Unforce all	—	Releases all forced variables (section 9.3)

"Run" to "Unforce all" are available only during simulation or online; while editing they are grayed out.

3.2.6 View

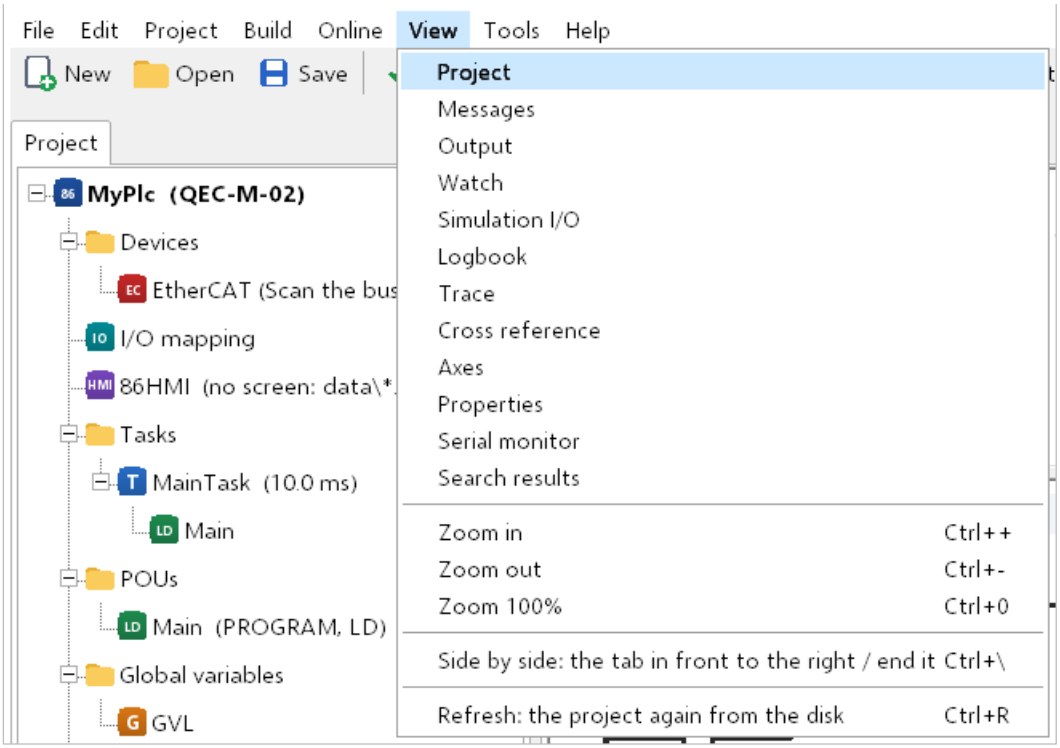


Figure 3-10 The "View" menu

Item	Shortcut	Purpose
Project	—	Shows the project tree (section 3.4)
Messages	—	Shows the "Messages" window: errors and warnings from the check and build
Output	—	Shows the build output in the "Output" window
Watch	—	Shows the "Watch" window (section 9.3)
Simulation I/O	—	Shows the "Simulation I/O" window (section 9.2)
Logbook	—	Shows the "Logbook" (section 10.5)
Trace	—	Shows the "Trace" (section 9.4)
Cross reference	—	Shows the "Cross reference" (section 9.5)
Axes	—	Shows the "Axes" window (sections 12.5, 12.6)
Properties	—	Switches to the "Properties" tab of the side panel (section 3.5)
Serial Monitor	—	Switches the "Output" window to the serial monitor (section 10.3)
Search results	—	Shows the "Search results" (section 3.7)
Zoom In	Ctrl++	Enlarges the whole interface (section 3.7)
Zoom Out	Ctrl+-	Shrinks the whole interface
Zoom 100%	Ctrl+0	Returns to 100%
Side by side: the tab in front to the right / end it	Ctrl+\	Moves the tab in front to the right side by side; ends the side-by-side view when already side by side (section 3.7)
Refresh: the project again from the disk	Ctrl+R	Rereads the whole project from the disk and refreshes the serial port list (see the explanation below)

Refresh (Ctrl+R) is used after the project files have been changed by an external editor, the command line or AI / MCP: POUs, GVLs, data types, user code, `project.86ld`, the sketch and 86HMI are all reread, the open tabs are restored, and the status bar shows "Refreshed: the project as it is on the disk". When there are unsaved changes, it first asks whether to save them (Yes: save, then reread; No: discard the changes; Cancel: do not refresh). During simulation or online, it shows "Stop the simulation / go offline first".

3.2.7 Tools

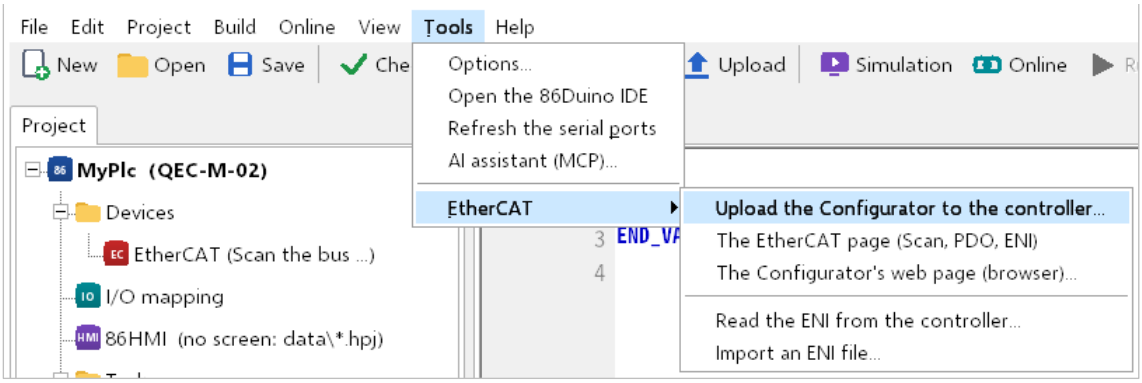


Figure 3-11 The "Tools" menu and the EtherCAT submenu

Item	Shortcut	Purpose
Options...	—	86Duino IDE folder, language, online again after an upload (section 2.6)
Open the 86Duino IDE	—	Opens the 86Duino IDE and loads the `.ino` of the project
Refresh the serial ports	—	Lists the COM ports in the serial port box on the toolbar again (use it after plugging in the controller)
AI assistant (MCP)...	—	Opens the "AI Assistant" tab: questions, inserting code, AI service settings and Claude Desktop (MCP) settings (sections 15.3, 15.4)
EtherCAT ▸ Upload the Configurator to the controller...	—	Uploads the EtherCAT Configurator to the controller (section 11.3)
EtherCAT ▸ The EtherCAT page (Scan, PDO, ENI)	—	Opens the EtherCAT page (same as "Project > EtherCAT")
EtherCAT ▸ The Configurator's web page (browser)...	—	Opens the Configurator web page in the browser (network mode)
EtherCAT ▸ Read the ENI from the controller...	—	Reads the ENI back from the controller and creates the SubDevices and PDOs (section 11.4)
EtherCAT ▸ Import an ENI file...	—	Creates the SubDevices and PDOs from an ENI file on the PC (section 11.4)

3.2.8 Help

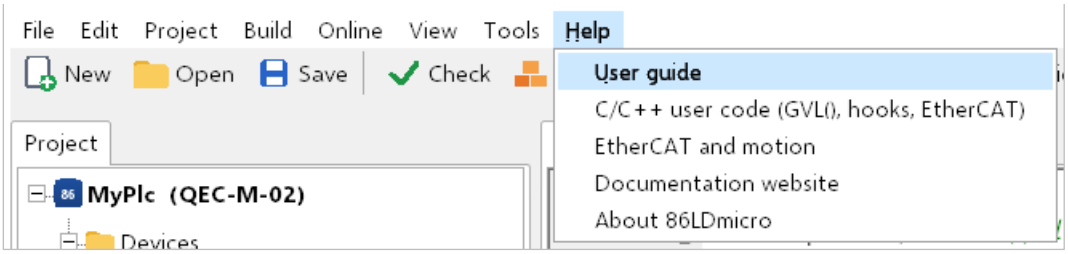


Figure 3-12 The "Help" menu

Item	Shortcut	Purpose
User guide	—	Opens the online documentation website
C/C++ user code (GVL(), hooks, EtherCAT)	—	Opens the online documentation website
EtherCAT and motion	—	Opens the online documentation website
Documentation website	—	Opens the online documentation website https://www.qec.tw/86duino/86ldmicro/
About 86LDmicro	—	Version and copyright information

The first four items all open https://www.qec.tw/86duino/86ldmicro/ in the browser. When offline, the documentation and the user manual PDF are in `<IDE folder>\tools\LDmicro\tool\docs` (the Japanese version is in `docs\ja`).

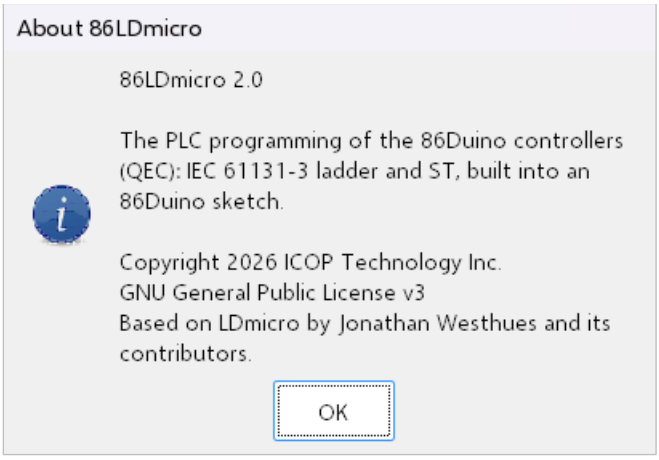


Figure 3-13 "About 86LDmicro"

3.3 Toolbar

The toolbar gathers the most frequently used commands, with the serial port box on the right. Hovering the mouse over a button shows a tooltip.



Figure 3-14 Toolbar and serial port box

Button	Shortcut	Function
New / Open / Save	Ctrl+N / Ctrl+O / Ctrl+S	New project, open project, save
Check	Shift+F7	Only compiles the PLC program and lists the errors
Build	Ctrl+B	Writes the sketch files
Compile	F7	Compiles with the 86Duino IDE
Upload	F8	Compiles and uploads to the controller
Simulate	F5	Starts / ends the simulation; the button stays pressed during simulation
Online	Ctrl+F5	Logs in to / disconnects from the controller; the button stays pressed while online
RUN / STOP	F9 / Shift+F9	Run / stop during simulation or online
AI	—	Opens the "AI Assistant" tab (section 15.3)
Serial monitor	—	Switches the "Output" window to the serial monitor to view the output of Serial.print (section 10.3)
Magnifier	Ctrl+F	Find in this tab
Zoom out / 100% / Zoom in	Ctrl+- / Ctrl+0 / Ctrl++	Zooms the whole interface; the current ratio is shown in the middle, click it to return to 100%
Serial port box	—	The COM port of the controller; select it from the list or type it in. When it is empty, upload or login first asks for "The controller's serial port"

When the window gets narrow, the toolbar changes in this order: only Compile, Upload, Simulate and Online show text, the others show icons only; when it is narrower still, all buttons show icons only (the text becomes a tooltip), and if necessary the toolbar splits into two rows.



Figure 3-15 The toolbar in a narrow window: icons only, in two rows

3.4 Project tree

The project tree (the "Project" pane on the left) lists everything in the project. **Double-click** a node to open its page or editor, **right-click** it for the menu of that node. When a project is opened for the first time, everything is expanded; after that the expanded state is remembered.

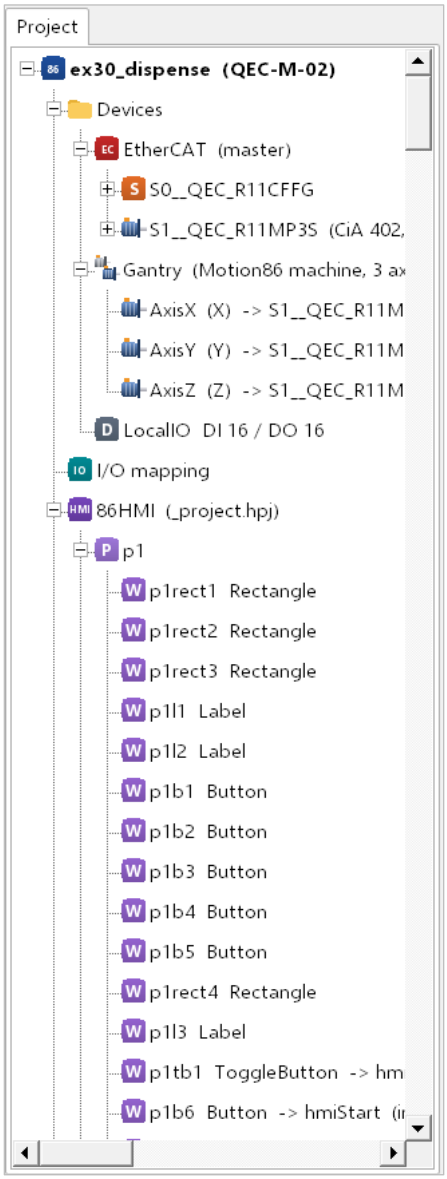


Figure 3-16 Project tree (ex30_dispense): devices, I/O mapping, 86HMI, tasks, POUs, global variables, data types, user code, Sketch

Node	Contents	Double-click
Project name (controller)	For example "MyPlc (QEC-M-02)"	Opens the project settings page (section 4.2)
Devices	Each device; under the EtherCAT MainDevice are the SubDevices, drives and PDO entries; under a Motion86 machine are its axes; without EtherCAT there is "EtherCAT (Scan the bus ...)"	Devices page (section 4.4); a SubDevice opens the SubDevice page, a PDO entry is mapped to a variable, machines and axes open their pages (Chapters 11, 12)
I/O mapping	Variable ↔ channel	Opens the I/O mapping page (section 4.6)
86HMI	Pages and widgets of the `.hpj`; the bound variable is shown after each widget; without a screen, "no screen: data*.hpj" is shown	Double-click a widget to open its binding form (Chapter 13)

Node	Contents	Double-click
Tasks	Each task (interval or type) and the PROGRAMs it runs in order	Double-click a task to open its settings (section 4.3); double-click a program to open its editor
POU	All PROGRAMs / FUNCTION_BLOCKs / FUNCTIONs, e.g. "Main (PROGRAM, LD)"; protected ones are marked read only / hidden	Opens the editor
Global variables, Data types	GVLs, data type files	Opens the editor (section 4.5)
User code (C/C++)	`.cpp86` files	Opens the editor (Chapter 14)
Sketch (86Duino IDE)	The sketch's `.ino`, `.cpp` and `.h` files; `plc86*` files are marked "written by a build"	Opens them for editing inside 86LDmicro

When a file has unsaved changes, a * appears after the node name. The context menus of the main nodes:

Node	Context menu
Project name	Project settings... / Build (write the sketch) / Compile (86Duino) / Upload to the controller / Simulate / Login to the controller / Open the sketch folder / Open the 86Duino IDE / Close the project
Devices	The devices page / Add a device ▸ (localio, eva, com, modbus, ethernet, modbus-server, ethercat, motion86)
A device	Edit... / Remove / The devices page / I/O mapping... (a Motion86 machine also has "The Axes panel (commissioning)")
EtherCAT MainDevice	The EtherCAT page / Upload the Configurator to the controller... / Open the Configurator (RD's web page) / Read the ENI from the controller... / Import an ENI file... / Edit... / Remove
PDO entry	Link to a variable... / Unlink / Add to watch / Cross reference / The EtherCAT page
I/O mapping	Open / Add a link...
Tasks	Add a task... / The tasks page
A task	Edit the task... / Add a PROGRAM (ladder)... / Add a PROGRAM (ST)... / Remove the task / Run a PROGRAM in this task ▸
A program in a task	Open / Earlier in the task / Later in the task / Out of this task
POUs (folder)	Add a PROGRAM (ladder / ST / SFC), Add a FUNCTION_BLOCK (ladder / ST), Add a FUNCTION (ST)
A POU	Open / Rename... / Cross reference / Remove from the project / Run in the task ▸ (PROGRAMs only; the task it is currently in is checked)
Global variables, Data types, User code	Folders: Add a global variable list... / Add a data type... / Add a user code file (C/C++)...; files: Open / Remove from the project (data types also have "Rename...")
Sketch (86Duino IDE)	The sketch's .ino (the 86Duino IDE's) / A new .h / .cpp in the sketch... / Open the sketch folder / Open the 86Duino IDE

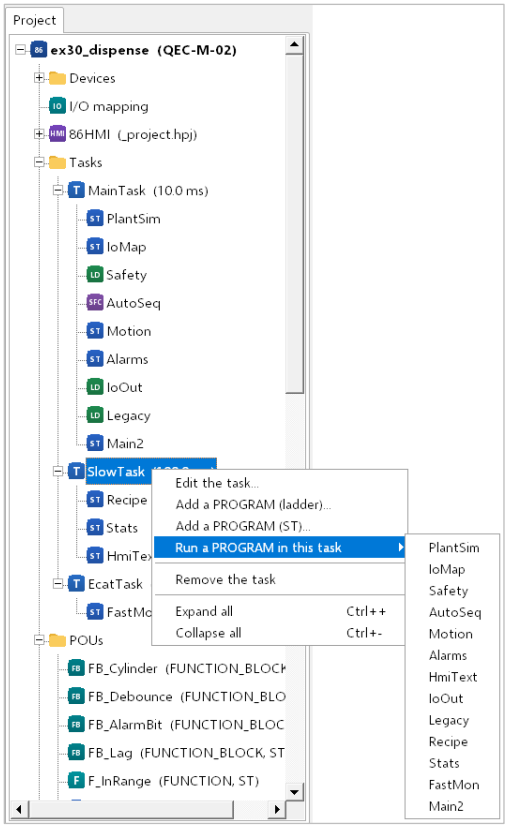


Figure 3-17 The context menu of a task node: the "Run a PROGRAM in this task" submenu lists all PROGRAMs

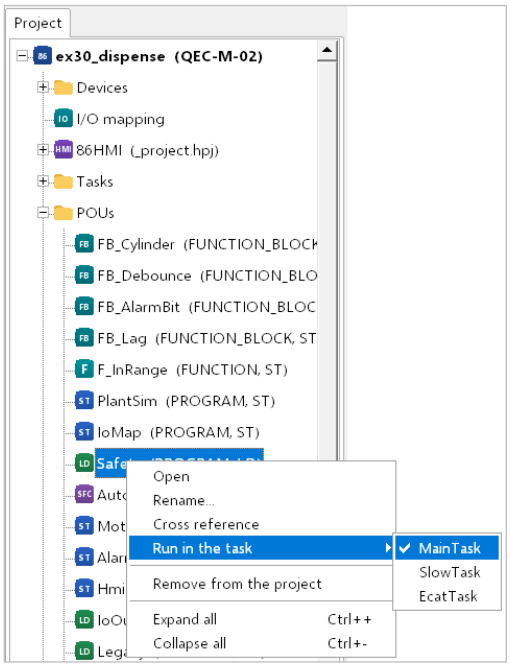


Figure 3-18 The context menu of a POU: Open, Rename, Cross reference, Remove from the project, Run in the task

- Every context menu (including the one you get by right-clicking an empty area) has **Expand all** and **Collapse all** at the bottom; when the project tree has the focus, you can also press **Ctrl++** / **Ctrl+-**.
- "Remove from the project" only takes the file out of the project; the file stays in `data\86ld` (you are asked first). Delete it yourself if necessary.
- When you select a PDO entry or an 86HMI widget, "Properties" in the side panel shows its variable and value.

3.5 POU editors and the side panel

Double-click a POU in the project tree to open its editor in the document area. The LD and SFC editors have a declaration area at the top (`VAR ... END_VAR`, the same code editor as for ST, with syntax colors and completion) and the diagram below; the ST editor is the text of the whole program. The LD, ST and SFC tabs all have a side panel on the right with two tabs at the top, "**Toolbox**" and "**Properties**"; the **»** button in the upper-right corner folds / unfolds the panel.

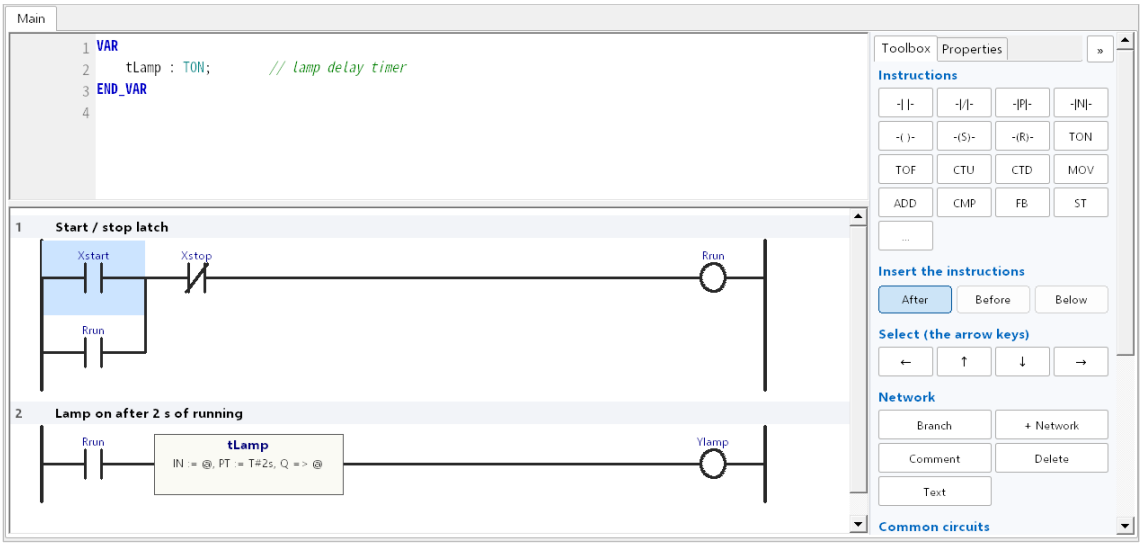


Figure 3-19 The ladder diagram editor: the declaration area at the top, networks in the middle, "Toolbox" in the side panel

Editor	Contents of "Toolbox"	Details
Ladder diagram (LD)	Instructions (contacts, coils, TON, TOF, CTU, CTD, MOV, ADD, CMP, FB, ST; "..." lists all instructions); Insert the instructions (After, Before, Below); Select (the arrow keys) (← ↑ ↓ →); Network (Branch, + Network, Comment, Del, Text); Common circuits (templates such as Start / stop latch)	Chapter 7
Structured Text (ST)	ST snippets (IF, CASE, FOR, timers...); double-click to insert	Section 8.1
Sequential Function Chart (SFC)	Chart: + Step, Alternative, Parallel, + Branch, Jump, Edit, Del, Text	Section 8.4

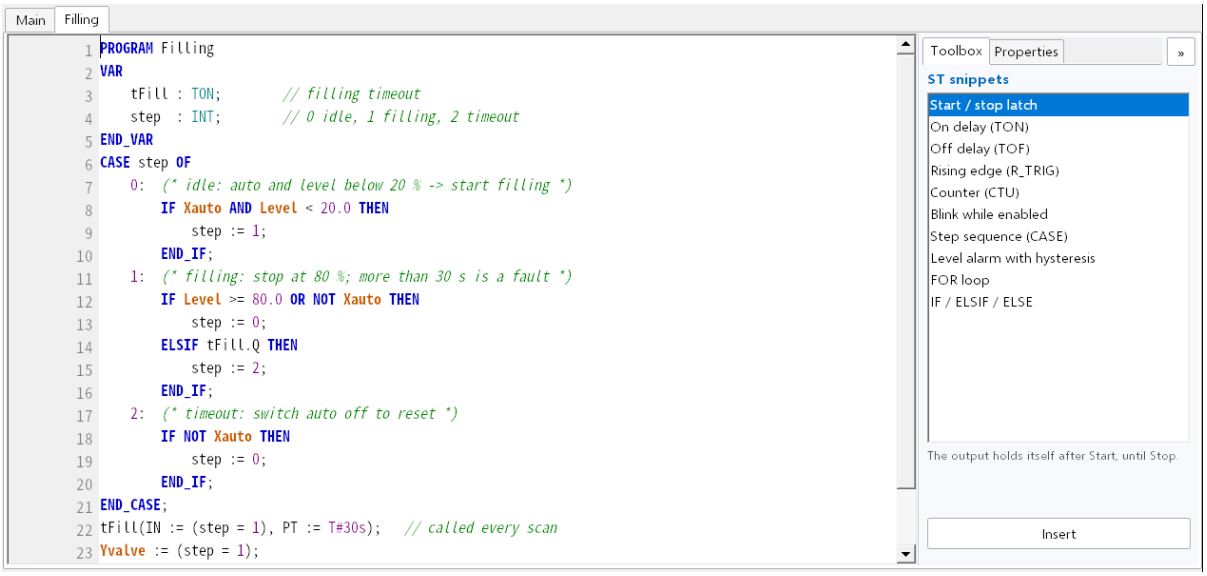


Figure 3-20 The ST editor and "ST snippets" on the right



Figure 3-21 The SFC editor: the declaration area at the top, the chart in the middle, the "Chart" tools on the right

The "**Properties**" tab shows the variable of the selected ladder element, watch row, EtherCAT entry or 86HMI widget: its type, where it is declared, initial value, comment, channel (I/O mapping), 86HMI binding, and its value during simulation or online. With nothing selected it shows "Select a ladder element, a row of the watch, an EtherCAT entry or an 86HMI widget: its variable, value, I/O and 86HMI are here."

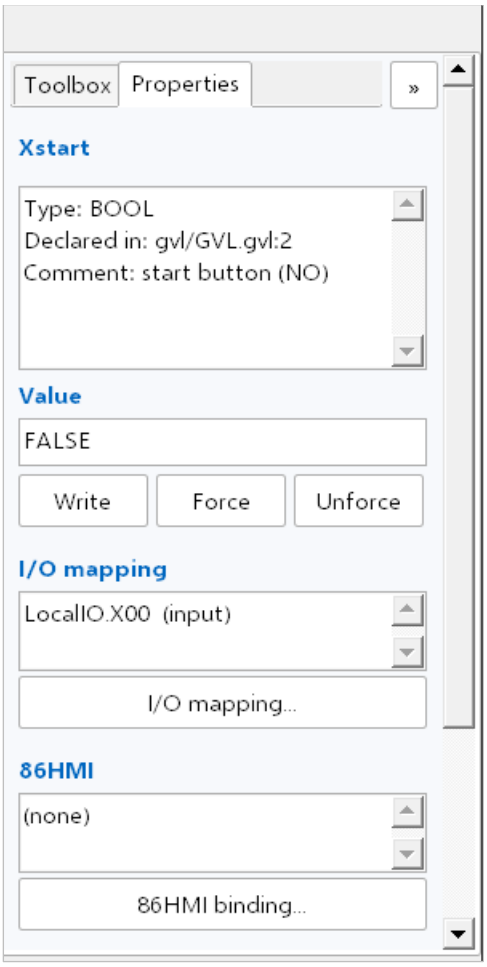


Figure 3-22 "Properties" in the side panel: the selected element's variable, type, I/O mapping and value

Right-clicking a ladder element, a variable in the code or a watch row gives the same variable menu (shown here for the variable `Rrun`): Write Rrun..., Force Rrun..., Unforce (these three only during simulation or online), I/O mapping of Rrun..., 86HMI binding of Rrun..., Add Rrun to the watch window, Add to the trace, Where is Rrun used?, Its declaration, Properties.

*__Tip:__ In the ladder and SFC editors, press Ctrl+T to switch to text mode (the .lad / .sfc text of the whole POU); click "Back to the diagram" to return to graphical editing (sections 7.1 and 8.4).

3.6 Tool windows

The tool windows at the bottom are switched with tabs; the View menu brings back a window you have closed. Right-click a tab and choose "**Float (a window of its own)**" to make it a separate window (for example to drag it to a second monitor); closing the floating window puts it back in place. Drag the splitters between the areas to resize them.

Window	Purpose	Details
Messages	Errors and warnings from check and build: message, file, line. Double-click to jump to the faulty element or line; right-click: Go to, Fix with AI (copies the question), Copy the message	Section 9.6
Output	Switch at the top between "Build output" and "Serial monitor": the progress of build, compile and upload, and what Serial.print prints	Sections 10.1 and 10.3
Watch	Type a variable name at the top and click **Add**; shows variable, value and type; double-click a BOOL to toggle it; right-click to write, force, add to the trace or remove from the watch; **Unforce all**	Section 9.3
Simulation I/O	During simulation, lists the physical point of each I/O mapping (channel, variable, input / output); double-click to toggle a digital input or enter an analog value	Section 9.2
Logbook	Events of the controller (or the simulator): start, EtherCAT state, errors, watchdog, force release	Section 10.5
Trace	Right-click a variable in the watch and choose "Add to the trace" (up to 8); sampled every 100 ms, showing the last minute; right-click: Pause, Clear the samples, Remove all variables, Copy as CSV	Section 9.4
Cross reference	Where a variable is read, written and declared in each POU (Ctrl+Shift+F)	Section 9.5
Axes	Power-on state, position and status of all axes; commands can be issued directly	Sections 12.5 and 12.6
Search results	Results of searching the project (Ctrl+Shift+H); double-click to jump to the line	Sections 3.7 and 9.5

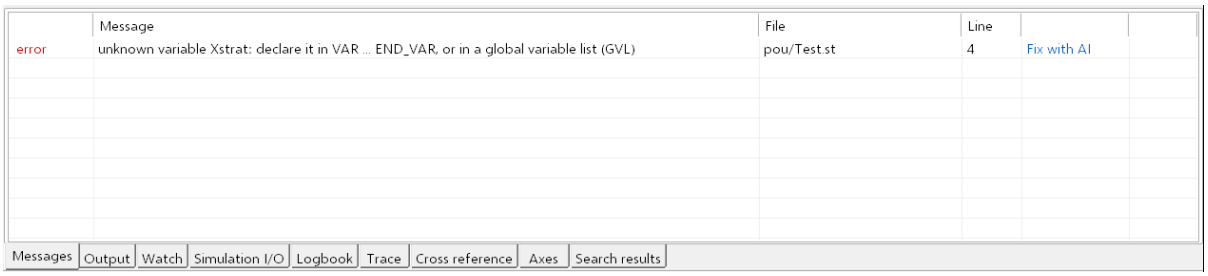


Figure 3-23 The "Messages" window: one error per row; double-click to jump to the element

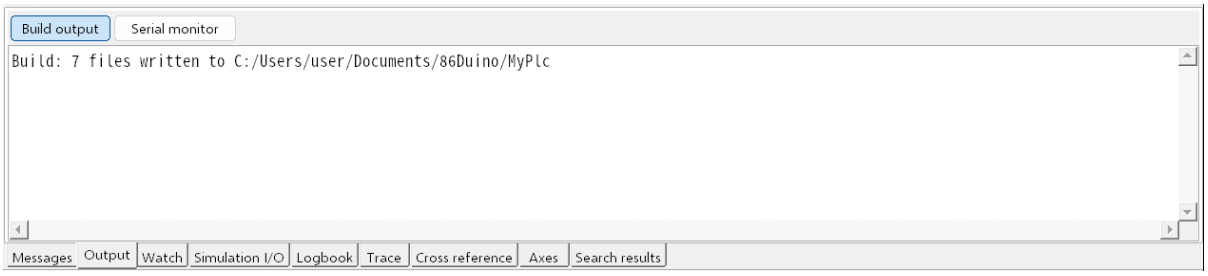


Figure 3-24 "Build output" in the "Output" window: the progress of build, compile and upload

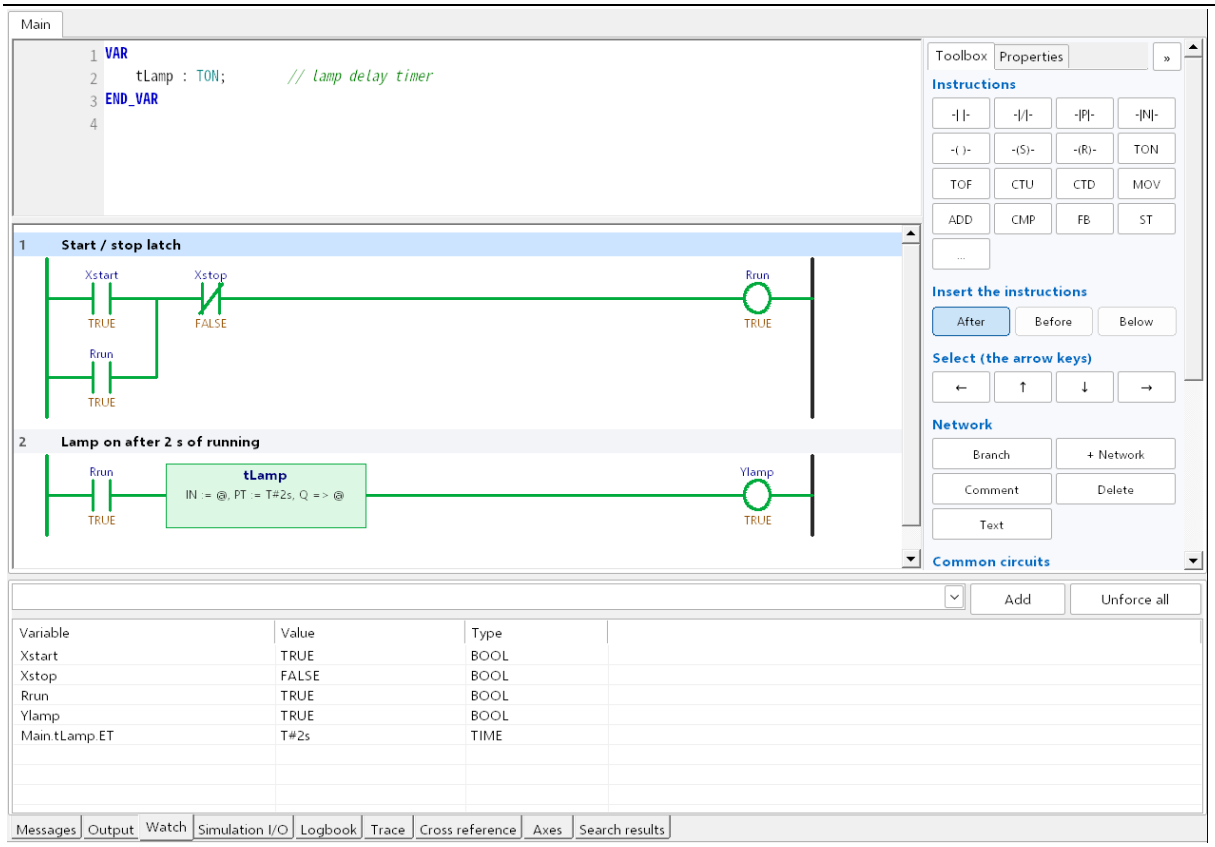


Figure 3-25 The "Watch" window: values are shown live once variables are added (they can also be written and forced)

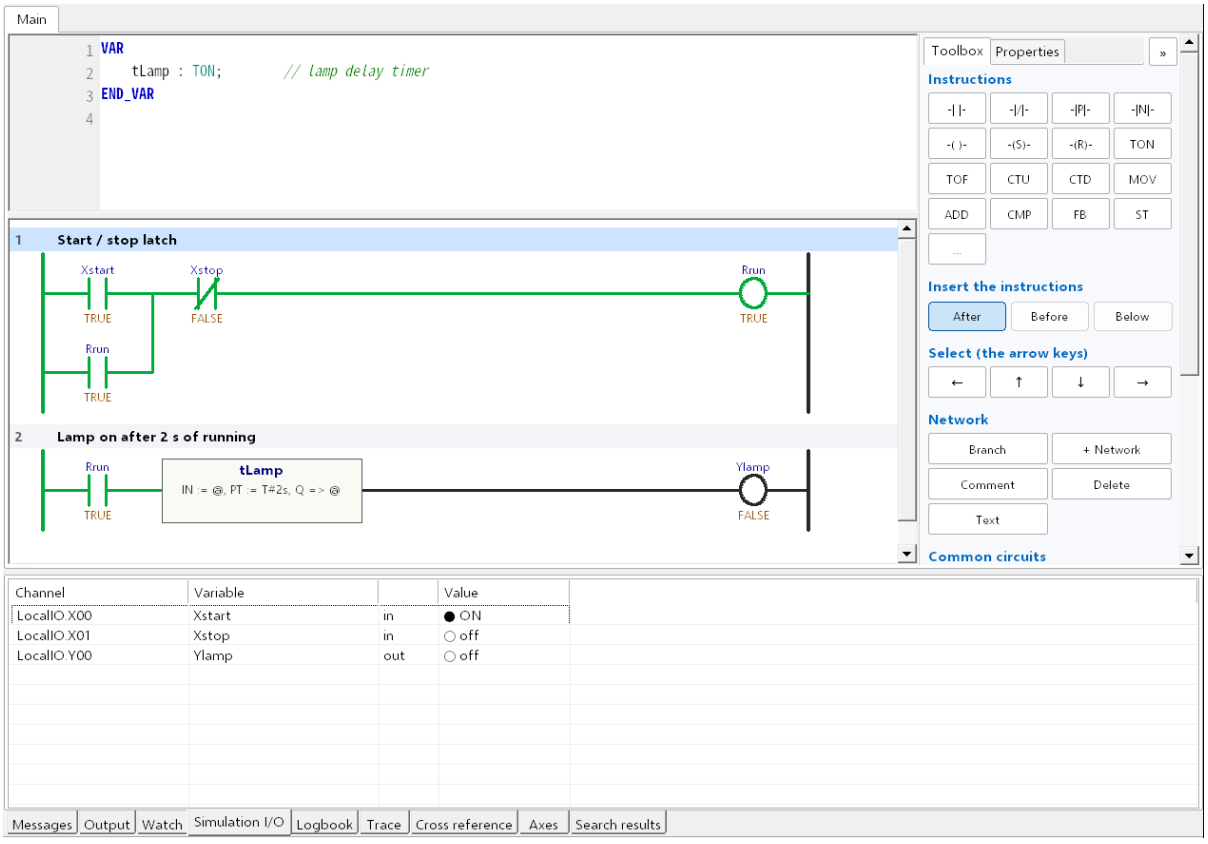


Figure 3-26 "Simulation I/O": physical points such as LocalIO.X00

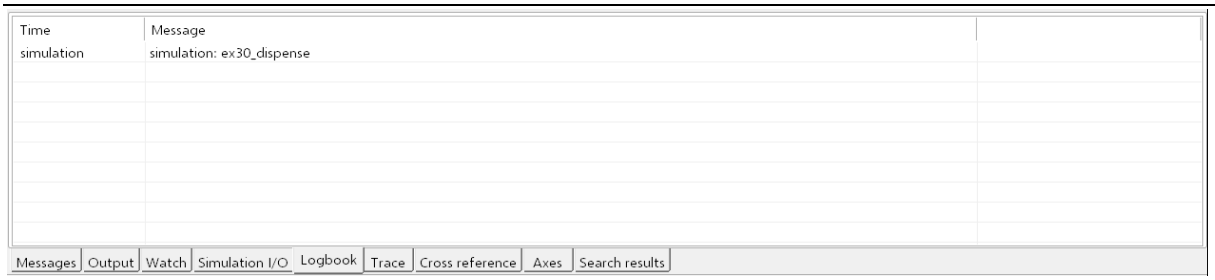


Figure 3-27 "Logbook": the EtherCAT MainDevice start messages and the program starting to run

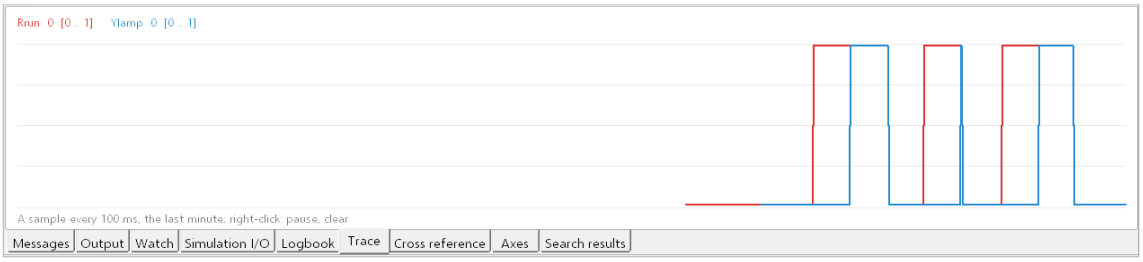


Figure 3-28 "Trace": the waveforms of variables added from the watch

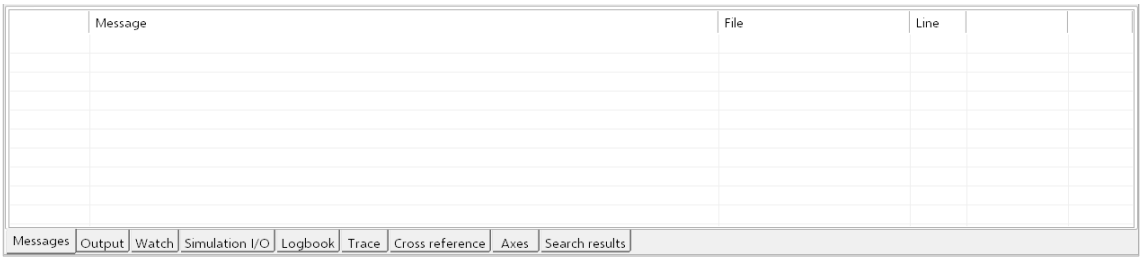


Figure 3-29 "Cross reference": where the variable Rrun is read, written and declared

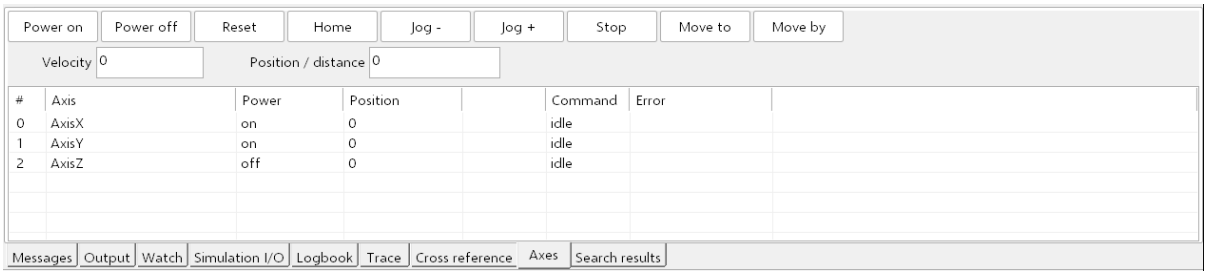


Figure 3-30 The "Axes" window



Figure 3-31 The context menu of a tool window tab: Float (a window of its own)

3.7 Tabs, side by side, zoom and find

3.7.1 Tabs

- Each POU, GVL and settings page has its own tab. **Ctrl+Tab** / **Ctrl+Shift+Tab** switch tabs; **Ctrl+W** or a click with the **middle mouse button** on a tab closes it.
- Right-click a tab: Close (Ctrl+W), Close the others, Close all, Split: move to the right side (when already side by side: "Move to the right side" / "Move to the left side"), End the split (all on the left).
- Each project remembers its open tabs and split state and restores them the next time you open the project.

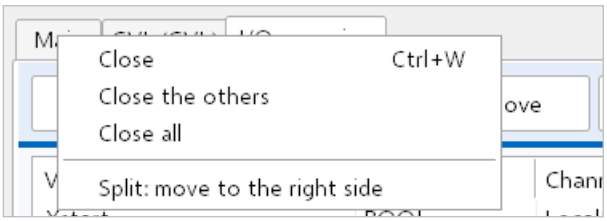


Figure 3-32 The context menu of a tab

3.7.2 Side by side

1. Right-click a tab > "**Split: move to the right side**", drag a tab to the right third of the document area, or use "View > Side by side" (**Ctrl+**\). At least two tabs must be open.
2. Drag the splitter in the middle to resize; move tabs between the two sides with the right-click "Move to the left side / Move to the right side" or by dragging.
3. The side you click is the active side (marked by a blue line at the top); Save, find and Ctrl+Tab act on that side. During simulation and online, the values on both sides are updated at the same time.
4. When all the tabs on one side are closed, or you right-click and choose "End the split (all on the left)", the view returns to a single column.

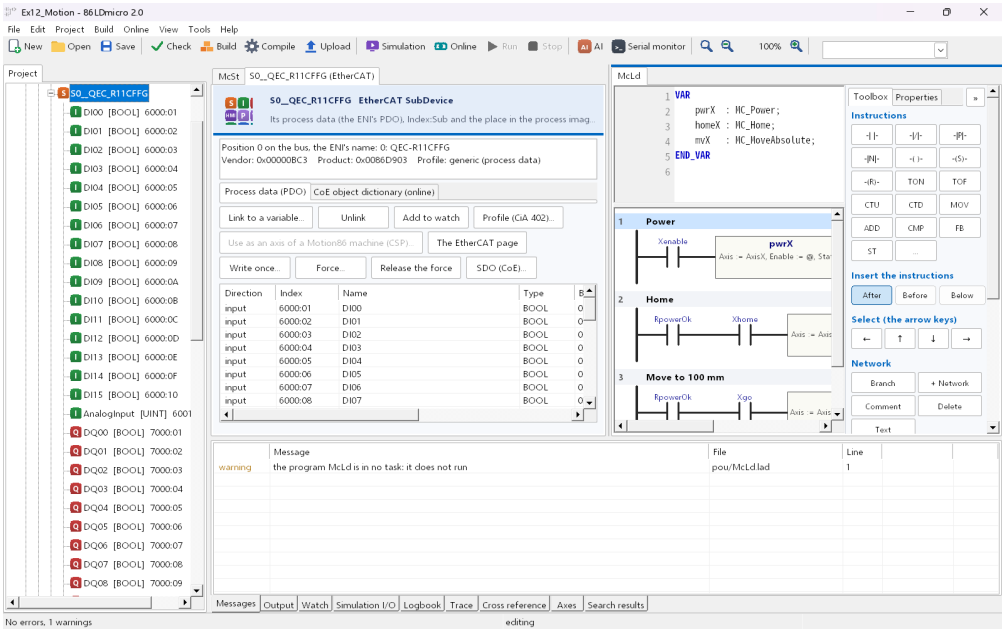


Figure 3-33 Side by side: a SubDevice page on the left, a ladder diagram on the right

3.7.3 Zoom

Ctrl+mouse wheel, **Ctrl++** / **Ctrl+-** / **Ctrl+0**, or the Zoom out, Scale and Zoom in buttons on the toolbar. Code, ladder diagrams, settings pages, the project tree and the tool windows zoom together; the scale is remembered and used again at the next start.

3.7.4 Find in this tab (Ctrl+F)

1. Press **Ctrl+F** (or the magnifier on the toolbar, or "Edit > Find in this tab..."), and a find bar appears at the top of the tab. If you select some text first, that text is searched for right away.
2. Type the text to find; tick "Match case" if needed. The right end shows "n of m", or "Not found" when there is no match.
3. **Enter**, **F3** or ▼ finds the next match; **Shift+Enter**, **Shift+F3** or ▲ finds the previous one; **Esc** or ✕ closes the bar. In a ladder diagram, the matching contacts, coils and network titles are selected in turn.

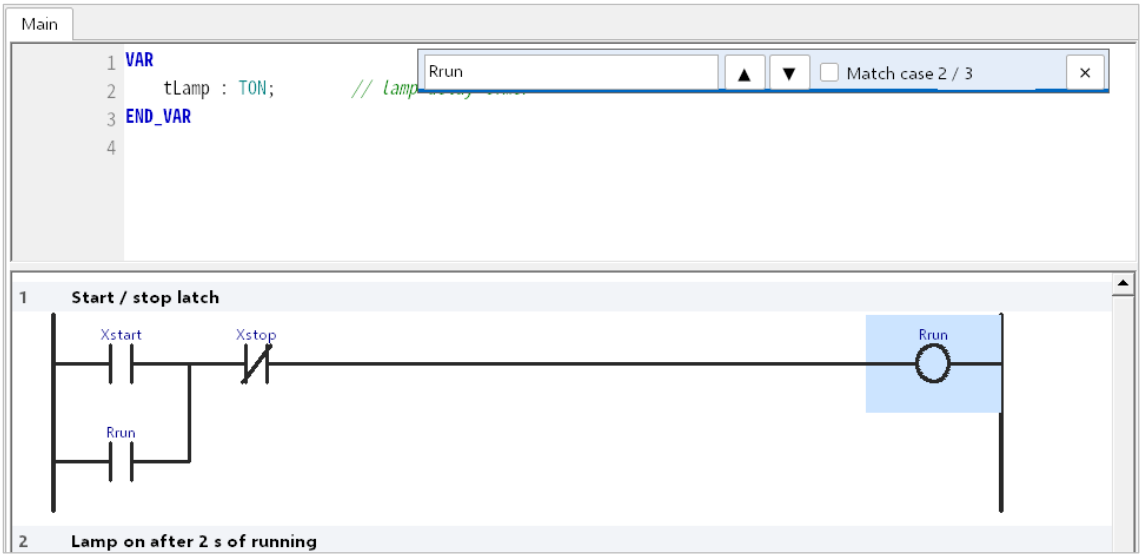


Figure 3-34 The find bar: finding Rrun in a ladder diagram (1 of 3)

3.7.5 Find a variable in the project (Ctrl+Shift+F) and Find in the project (Ctrl+Shift+H)

Function	What it searches	Result	Best for
Find in this tab (Ctrl+F)	The text or ladder elements of the current tab	Find bar	Stepping through one program
Find a variable in the project (Ctrl+Shift+F)	Where one variable is read, written and declared	Cross reference	Finding out who writes a variable
Find in the project (Ctrl+Shift+H)	Any text in the whole project: all POUs (including network titles and comments), GVLs, data types, user C/C++, the sketch's .ino	Search results	Finding comments, strings, functions, or things whose exact name you are unsure of

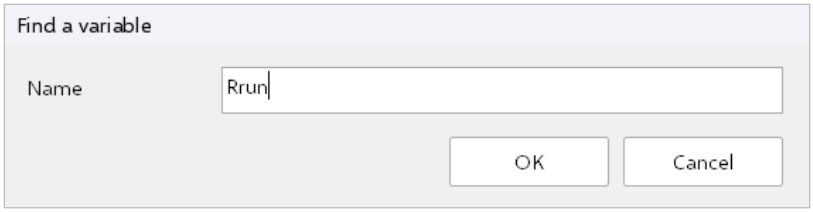


Figure 3-35 "Find a variable": enter the variable name

1. Press **Ctrl+Shift+H**; the bottom area switches to "Search results". (If you select some text in the program first, that text is searched for right away.)
2. Type the text and press **Enter** or click **Find**; tick "Match case" or "Whole word" if needed (for example, so that a search for `Pump` does not list `PumpSpeed`). The upper-right corner shows the number of places found.
3. **Double-click** a row (or select it and press Enter) to open the file and jump to that line: in ST and C/C++ the line is selected; in a ladder diagram the view scrolls to that network and selects it.

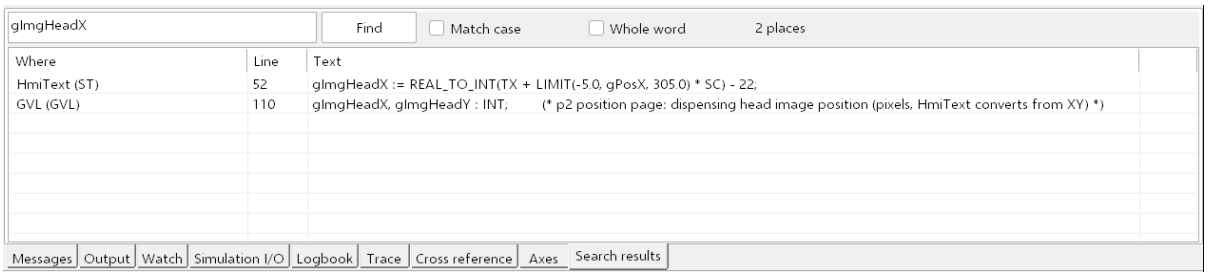


Figure 3-36 "Search results": 2 places found (an ST program and the global variable list)

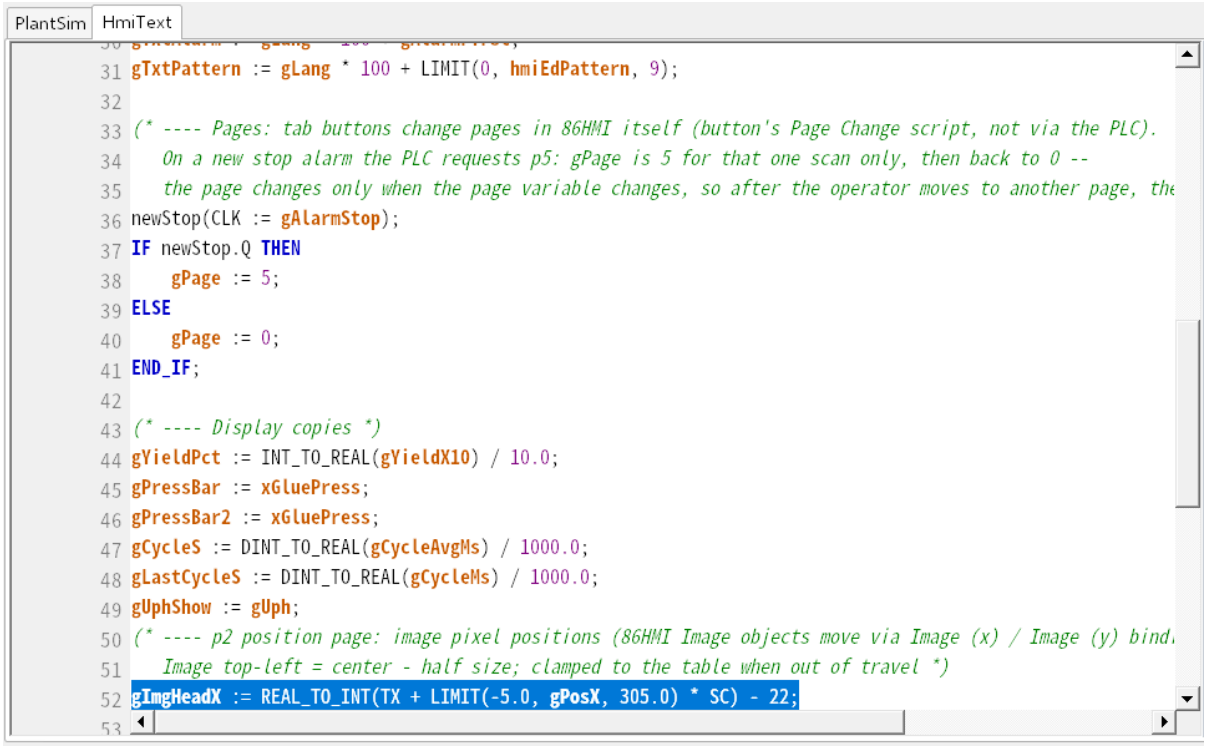


Figure 3-37 Double-clicking a row opens that ST program and selects the line

*__Tip:__ The search only finds; it does not replace. To rename a variable, open each result and change it, then press "Check" (Shift+F7) to make sure nothing was missed.
After changing files outside 86LDmicro, press **Ctrl+R** (View > Refresh) to reload the project from disk (section 3.2.6).

Chapter 4 Projects and settings

This chapter describes the files that make up an 86LDmicro project and how to set up the project settings, tasks, devices, global variables and data types, and I/O mapping; finally, it shows how to add POUs and user code.

4.1 Project file structure

An 86LDmicro project lives in the folder of the 86Duino sketch it belongs to. All its files are UTF-8 text files, so you can put them under version control with tools such as git. Taking `MyPlc` from Chapter 5 as an example (located by default in `Documents\86Duino`):

```
Documents\86Duino\MyPlc\
    MyPlc.ino                       calls plc86_setup() / plc86_loop() (written once only if missing;
you maintain it afterwards)
    plc86.h, plc86.cpp              PLC program (generated by the build, rewritten every time; do not
edit by hand)
    plc86_rt.h, plc86_rt.cpp        86LD run-time (generated by the build)
    plc86_ecat.h, plc86_ecat.cpp    EtherCAT (generated by the build)
    myhmi.h, myhmi.cpp              written by "Code Generation" in the 86HMI Editor (when there are
86HMI screens)
    data\
        *.hpj                       86HMI screens (if any)
        Resource\                   86HMI images, fonts and other resources (if any)
        86ld\
            project.86ld            project settings (JSON): controller, tasks, devices, I/O mapping,
bindings
            pou\                    POUs: *.lad (ladder), *.st (ST), *.sfc (SFC)
            gvl\                    global variable lists *.gvl
            dut\                    data types *.dut
            usercode\               user C/C++ *.cpp86
            ethercat\               copies of imported ENIs, CoE object dictionary cache
            build\                  plc86.symbols.json (symbol table for the online view), compiler
output
```

A new project has only `MyPlc.ino`, `project.86ld`, `pou\Main.lad` and `gvl\GVL.gvl`; the other folders appear only when you add the corresponding files, read in an ENI or build. The `project.86ld` of a new project looks like this:

```
{
  "format": "86LDmicro project",
  "version": 1,
  "name": "MyPlc",
  "target": {
    "board": "QEC-M-02",
    "mcu": "Vortex86EX2"
  },
  "tasks": [
    {
      "name": "MainTask",
      "kind": "cyclic",
      "intervalUs": 10000,
      "priority": 1,
      "programs": ["Main"]
    }
  ],
  "pous": ["pou/Main.lad"],
  "gvls": ["gvl/GVL.gvl"],
  "duts": []
}
```



Figure 4-1 The sketch folder: the .ino, the plc86* files generated by the build, and data\86ld

- The `plc86*` files and `data\86ld\build` are regenerated at every build, so you can leave them out of version control.
- The `.ino` is your file: 86LDmicro writes it once only if it does not exist; after that it only makes sure it calls `plc86_setup()` and `plc86_loop()` (section 14.5).
- Files removed from the project are not deleted; they stay in `data\86ld`. Delete them yourself if necessary.
- These text files can also be edited in an external editor, from the command line or by AI / MCP; afterwards press **Ctrl+R** (View > Refresh) in 86LDmicro to reload them (section 3.2.6). The text format of ladder `.lad` files is described in section 7.1.

4.2 Project settings

Double-click the project name (the top node) in the project tree, or choose "Project > Settings", to open the "Project settings" page. The page lists the project name, folder, controller, online view and the location of the 86Duino IDE; there are three buttons at the top:

Button	Function
Edit...	Opens the "Project settings" dialog (see below)
Options...	Opens "Options" (86Duino IDE folder, language; section 2.6)
Open the folder	Opens the sketch folder in File Explorer

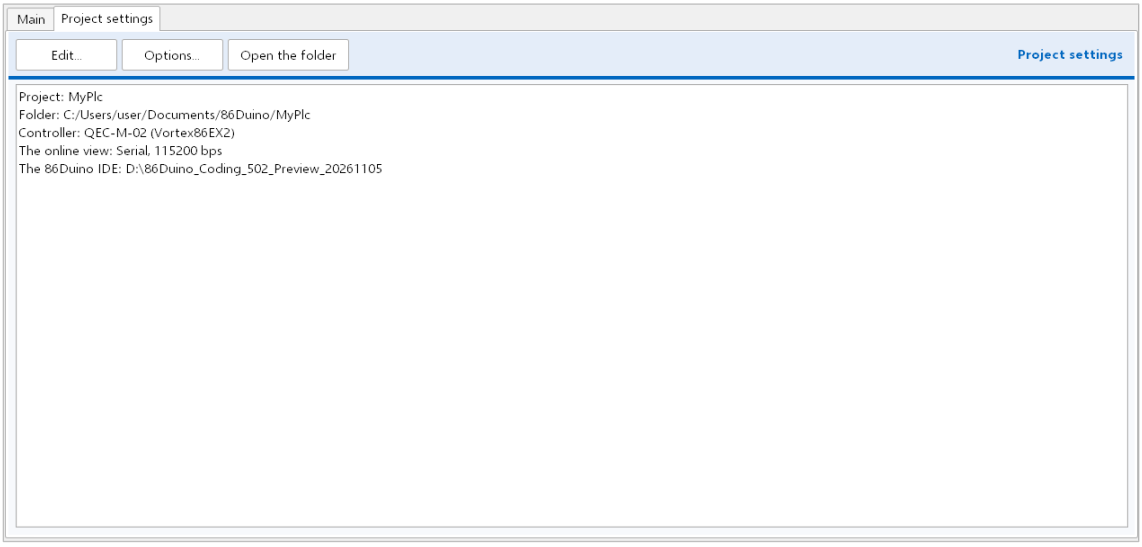


Figure 4-2 The "Project settings" page: project, folder, controller, online view, 86Duino IDE

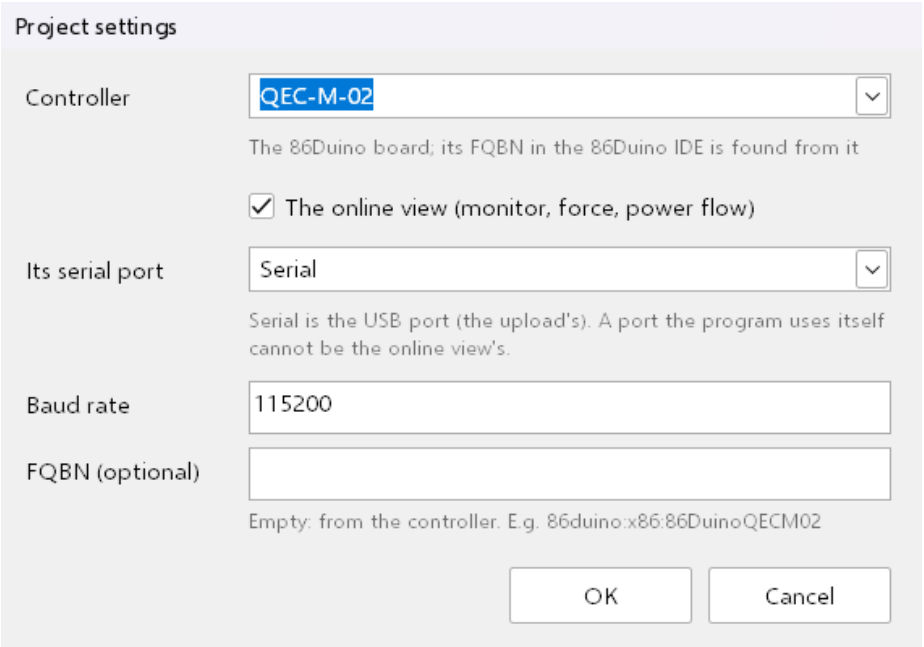


Figure 4-3 The "Project settings" dialog: controller, online view, serial port, baud rate, FQBN

Field	Default	Description
Controller	QEC-M-02	Determines the compile target (the FQBN in the 86Duino IDE) and the local I/O. The list has QEC-M-02, QEC-M-01, QEC-M-043T, QEC-M-070T, QEC-M-090T, QEC-M-150T, and the other QEC boards in the 86Duino IDE's boards.txt (for example the PPC series; they appear automatically with a newer IDE version)
The online view (monitor, force, power flow)	Ticked	When ticked, the uploaded program includes the online view (section 10.4)
Its serial port	Serial	The port for the online view: `Serial` is USB (the upload port; recommended); Serial1, Serial2, Serial3 and SerialCOM can also be chosen. A serial port that the program itself uses cannot be the online view port
Baud rate	115200	86LDmicro uses this baud rate automatically when it goes online; after changing it, build and upload again
FQBN (optional)	Empty	When empty, it is determined from the controller; for a special board you can enter it directly, for example `86duino:x86:86DuinoQECM02`

*__Note:__ If the sketch's `.ino` calls `Serial.begin()` again after `plc86_setup()`, it overrides the baud rate of the online view and the connection fails (section 10.4).

When 86LDmicro is started from the 86Duino IDE and the board selected in the IDE differs from the project's controller, 86LDmicro asks whether to switch to the IDE's board (section 2.5).

4.3 Tasks

Tasks determine when PROGRAMs run and in what order. Double-click "Tasks" in the project tree (or choose "Project > Tasks") to open the tasks page: one row per task, showing the task, kind, interval, watchdog and programs. The buttons at the top are **Add...**, **Edit...**, **Remove**, **Up** and **Down**; double-clicking a row is the same as **Edit...**.

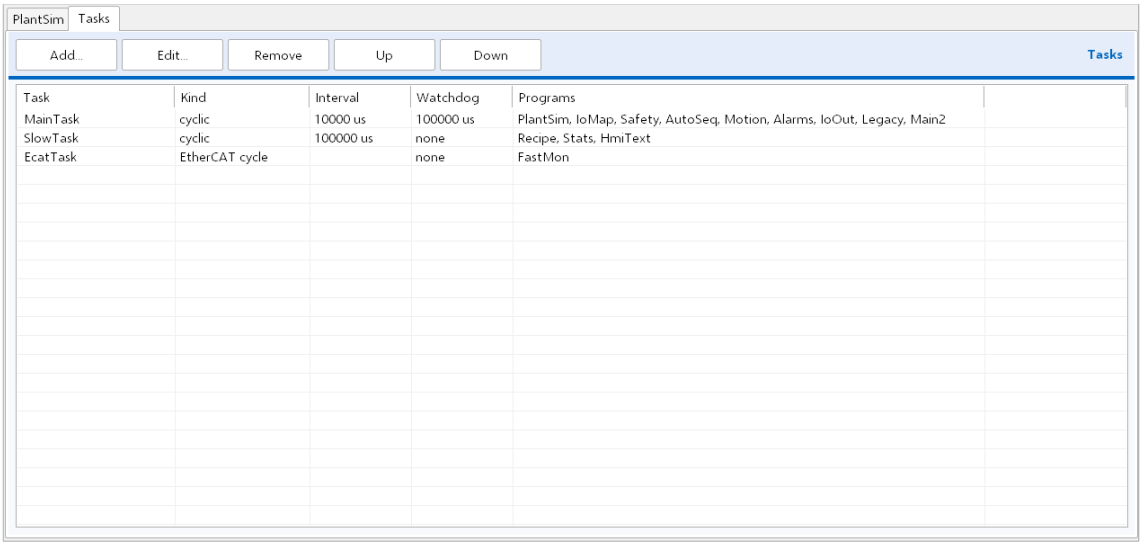


Figure 4-4 The tasks page: task, kind, interval, watchdog, programs

Field	Description
Name	Letters, digits and _ (for example `MainTask`)
Kind	cyclic, freewheel, EtherCAT cycle (see the table below)
Interval (us)	The execution interval of a cyclic task, 100 us to 10 s (enter 100 to 10000000). The MainTask of a new project has 10000 (10 ms)
Watchdog (us)	The PLC is stopped when a single scan takes longer than this; 0 means not used (see below)
Programs, in order	PROGRAM names separated by commas, run in that order; all PROGRAMs of the project are listed below the field

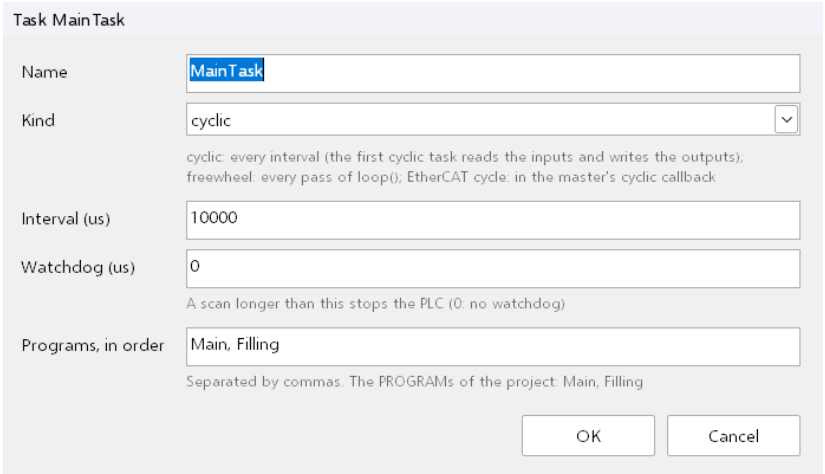


Figure 4-5 Task settings: name, kind, interval, watchdog, programs (in order)

Kind	When it runs	Use
cyclic	Once every "interval"	General PLC logic. The **first cyclic task** reads all inputs before its programs and writes all outputs after them
freewheel	Every pass of loop()	Background, low-priority processing
EtherCAT cycle	The cyclic callback of the EtherCAT MainDevice (86EVA or EtherCAT device)	Control that must be synchronized with EtherCAT; it interrupts the other tasks, so keep it short

- A project needs at least one **cyclic** task, and at most one **EtherCAT cycle** task.
- A PROGRAM can be in only one task (and not twice in the same task); otherwise the check reports an error. FUNCTION_BLOCKs and FUNCTIONs are not put in tasks; they are called by PROGRAMs.
- Programs created with "Project > Add a PROGRAM" are added automatically at the end of the first task.
- You can also do this in the project tree: right-click a task node > "Run a PROGRAM in this task ▸"; right-click a program in a task > "Earlier in the task", "Later in the task", "Out of this task"; right-click a POU > "Run in the task ▸" (section 3.4).
- When one scan of a cyclic task takes longer than its interval, the controller's Logbook records "task <task name>: a scan took … us (the interval is … us)" (the first time, and then once every 1000 times).

4.3.1 Watchdog

The watchdog finds programs that are too slow (scan time-out check). It works only on the controller and only for cyclic tasks (not in simulation):

- The time checked is the execution time of the programs themselves in a single scan: the time spent in the EtherCAT cycle interrupt, EEPROM writes (RETAIN), the motion control mailbox and 86HMI screen updates is not counted.
- The first 3 scans of each task are not checked (loading RETAIN at start-up, first initialization, etc.).
- When the time is exceeded, the PLC goes to STOP (the same as pressing STOP: the program no longer runs, the outputs keep their last values, all Motion86 machines stop and jogging ends; section 10.4). The Logbook records the task name, the time of that scan and the slowest program, for example "MainTask watchdog: STOP, 120 ms (3 other), AutoSeq 110 ms".
- When online, the "Output" of 86LDmicro shows "The PLC is in STOP: a scan took longer than the task's watchdog (Project > Tasks). RUN (F9) starts it again." and the status bar shows "The controller's watchdog stopped the PLC: see Output".
- After removing the cause, press **RUN** (F9) to run again. Remedies: reduce the processing in each scan (split loops and string handling over several scans), or increase the watchdog time.

*__Tip:__ Set the watchdog time much longer than the normal scan time (for example 100000 us for a task with a 10 ms interval); it is only meant to catch abnormal cases. The normal scan time is shown in the status bar when online (section 3.1).

4.4 Devices

Devices are the hardware and communications the project uses: local I/O, EtherCAT, 86EVA, serial ports, Modbus, network and Motion86 machines. Double-click "Devices" in the project tree (or choose "Project > Devices") to open the devices page, with one row per device (device, type, settings summary); the buttons at the top are **Add...**, **Edit...** and **Remove**.

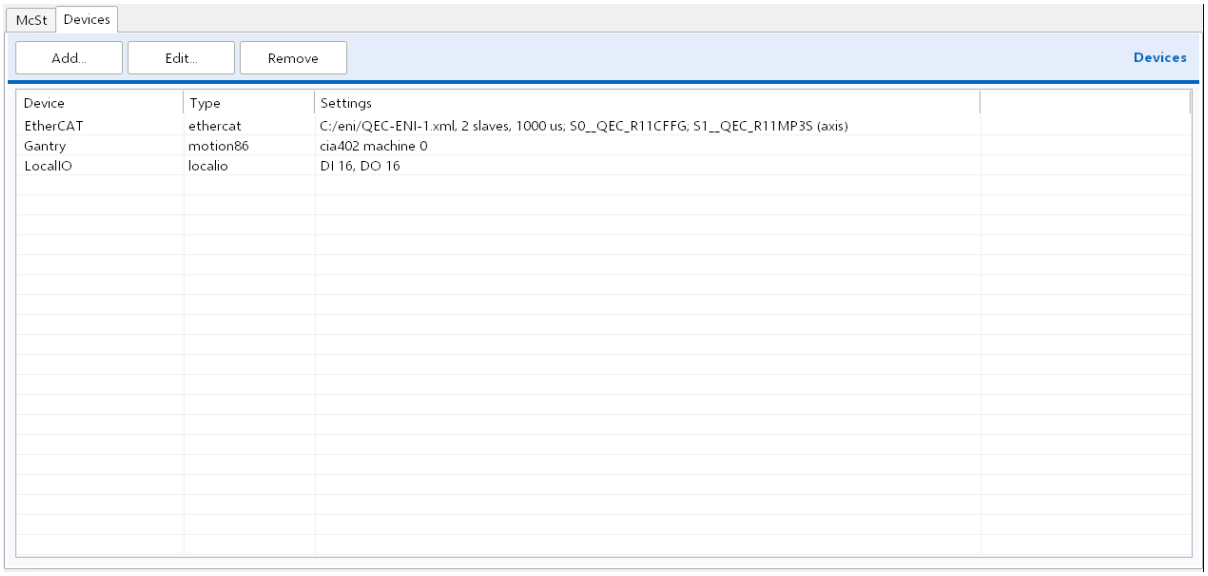


Figure 4-6 The devices page: localio (DI 16, DO 16)

1. Click **Add...** (or right-click "Devices" in the project tree > "Add a device ▸" to choose the type directly).
2. In "Add a device", choose the kind under "Type" and click **OK**.
3. The settings form of the device appears: one row per field (check boxes are on / off; numbers and text are typed directly); list fields (for example the serial port list or the Modbus SubDevices) are edited as JSON text. Click **OK** to add it.

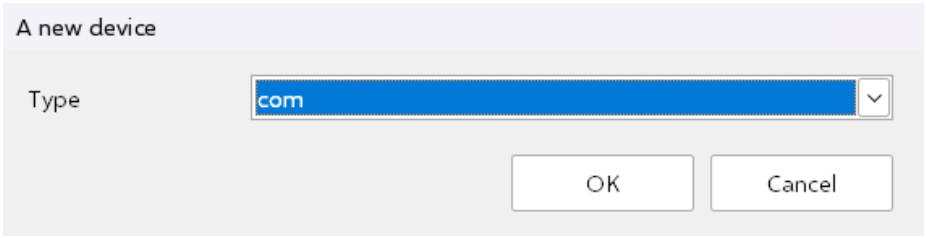


Figure 4-7 "Add a device": choosing the type

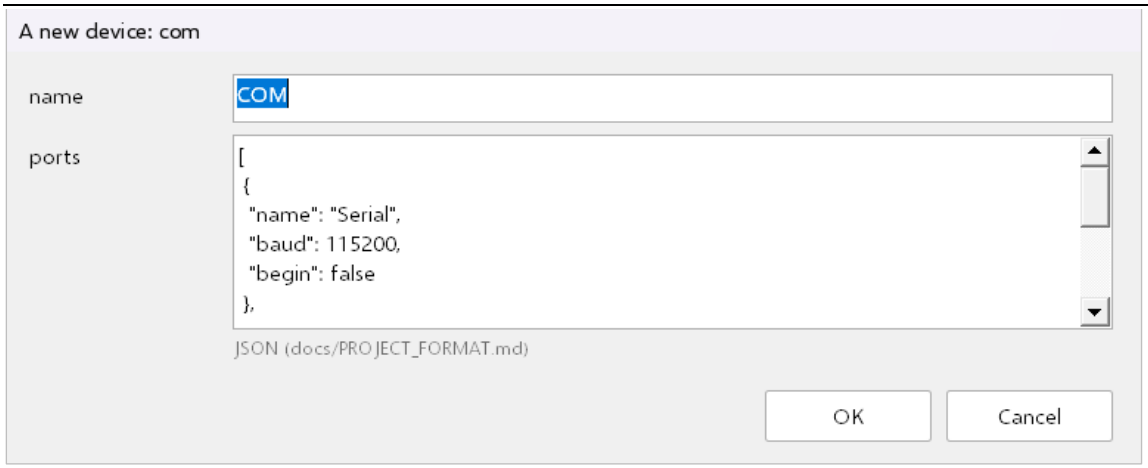


Figure 4-8 The settings form of a com device: name and ports (JSON)

Type	Description	Default contents when added
localio	QEC local DI / DO points (QEC-M-02: 16 / 16)	Name LocalIO, di 16, do 16
eva	86EVA virtual device (the EtherCAT method of LDmicro 1.x)	enable, io 16, analog 4, decimal 0, servo 0, serial 0
com	Serial ports: COM0, COM1… in order (for the UART instructions)	Serial 115200 (no begin), Serial1 9600
modbus	Modbus RTU / TCP MainDevice: buses and SubDevice list	Bus 0: RTU, COM1, enabled; SubDevice list empty
ethernet	Network settings (static IP / dhcp)	static, 192.168.1.100 / 255.255.255.0, gateway and DNS 192.168.1.1
modbus-server	Modbus TCP SubDevice: opens variables to SCADA / HMI for reading and writing	on, port 502, unit 1, vars empty
ethercat	EtherCAT MainDevice: usually created by reading in an ENI; the SubDevices and PDOs come from the ENI (Chapter 11)	eni QEC-ENI.xml, port 0, cycleUs 1000; also mode (auto / sync / freerun) and master (plc / sketch)
motion86	Motion86 machine and axes: opens "A new Motion86 machine" (Chapter 12)	—

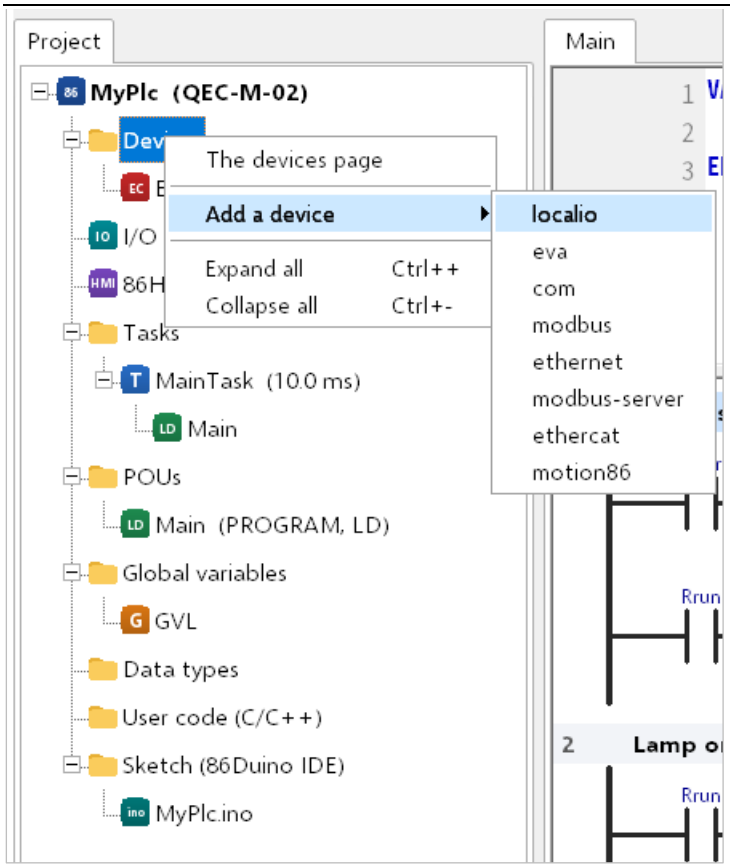


Figure 4-9 The context menu of "Devices" in the project tree: Add a device ▸ lists 8 types

Tip: If a new project has no localio device, you can still use the QEC-M-02 local I/O channels `LocalIO.X00` to `X17` and `LocalIO.Y00` to `Y17` (16 / 16 points by default).
The mode of an EtherCAT device: auto = DC sync when there are CiA 402 servos, free run otherwise; sync = DC sync (free run if it fails). master: plc = 86LDmicro starts the MainDevice (plc86_ecat.cpp); sketch = your .ino does begin, attach and start itself and then sets `plc86_ecOk = true`.

4.5 Global variables and data types

Every variable in 86LDmicro must be declared before it can be used: an undeclared name is shown with a red background in the ladder diagram, and the check lists it as an error. Variables declared in a **global variable list (GVL)** are visible to all POUs, 86HMI and C/C++; those declared in the VAR of a POU can be used only in that POU.

4.5.1 Global variable list (GVL)

Double-click "Global variables > GVL" in the project tree to open the editor; declare one variable per line and press Ctrl+S to save. The editor has syntax colors and completion; anything after `//` or inside `(* *)` is a comment (shown in the completion list and in Properties).

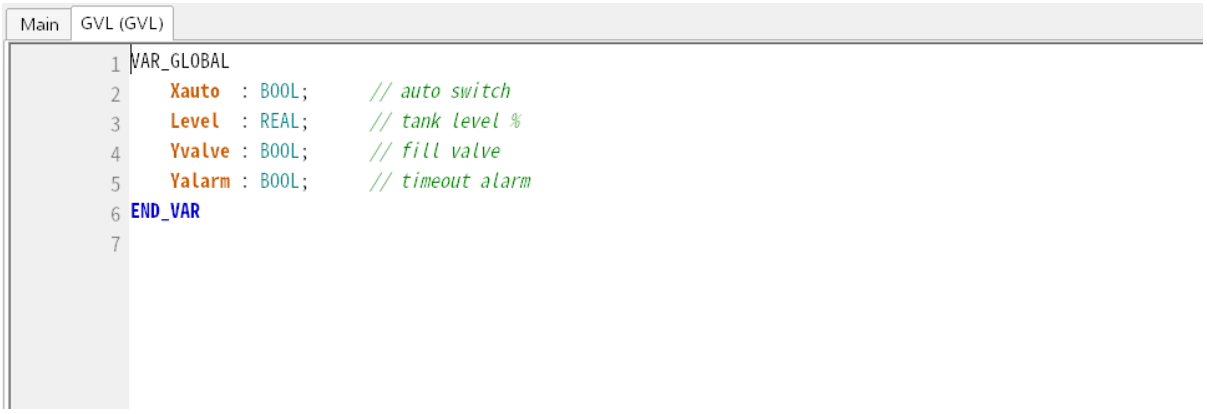


Figure 4-10 The GVL editor: one variable per line; the text after // is a comment

```
VAR_GLOBAL
    Xstart   : BOOL;                    // the comment is shown in the completion list and in
Properties
    Speed    : REAL := 12.5;            // initial value
    Recipe   : ARRAY[1..5] OF INT;      // array (index 1 to 5)
    Axis1    : Axis_t;                  // user-defined structure (data type, see below)
    Mode     : Mode_t;                  // user-defined enumeration
END_VAR
VAR_GLOBAL RETAIN                       // retentive: stored in EEPROM, restored at the next power-up
    TotalRuns : DINT;
END_VAR
VAR_GLOBAL CONSTANT                     // constants: cannot be changed by the program
    MAX_SPEED : REAL := 100.0;
END_VAR
```

Section	Description
VAR_GLOBAL	Ordinary global variables; `:= value` sets the initial value (0 / FALSE / empty string if omitted)
VAR_GLOBAL RETAIN	Retentive: the values are stored in the controller's EEPROM and restored at the next power-up; kept by a warm restart, cleared by a cold restart (section 3.2.5)
VAR_GLOBAL CONSTANT	Constants: the value is given with `:= value`; writing to one in the program is an error in the check

*__Note:__ RETAIN is not written at every assignment: the controller checks about once a second and, when something has changed, writes the new snapshot to the EEPROM in parts (between scans); it takes effect only when it has been written completely. If the power fails before the new snapshot is complete, the last

completely written values are restored at the next power-up. EEPROM has a limited number of write cycles. Do not make constantly changing variables (counters, timer values, analog values) RETAIN; if you need to keep them, copy them to RETAIN variables only when necessary (at shutdown, on a recipe change).

You can split the variables into several GVLs by purpose ("Project > Add a global variable list...", for example `IO`, `Recipe`); variable names must be unique across all GVLs.

4.5.2 Local variables of a POU

The declaration area of a POU uses VAR (local variables; function block instances usually go here, for example tLamp : TON;) and VAR_TEMP (temporary: starts from the initial value at every call and is not kept); FUNCTIONs and FUNCTION_BLOCKs also have the parameters VAR_INPUT, VAR_OUTPUT and VAR_IN_OUT (PROGRAMs have no parameters). See section 8.2.1 for the full range of what is supported. Other POUs and the Watch window read local variables as programname.variable, for example Main.tLamp.ET.

***Tip:** FUNCTION_BLOCKs and FUNCTIONs can read global variables and constants of elementary types (for example MAX_SPEED); global variables of array and structure types cannot be accessed directly inside them: pass them in with VAR_INPUT (a copy is passed); if the block is to modify an array or structure, pass a local variable of the calling ST program with VAR_IN_OUT and copy it back to the global variable after the call (section 8.2.1).

4.5.3 Elementary data types

Type	Range / description	Type	Range / description
BOOL	TRUE / FALSE	REAL	32-bit floating point
SINT / USINT	8-bit signed / unsigned	LREAL	64-bit floating point
INT / UINT	16-bit signed / unsigned	TIME	Time, for example `T#1s500ms`, `T#100ms`
DINT / UDINT	32-bit signed / unsigned	STRING	String, for example `'hello'`
LINT / ULINT	64-bit signed / unsigned	BYTE / WORD / DWORD / LWORD	Bit strings (8 / 16 / 32 / 64 bits)

4.5.4 Data types (structures, enumerations, arrays)

"Project > Add a data type..." creates a `.dut` file; define structures, enumerations and named arrays in `TYPE ... END_TYPE`, and you can then use them as types in GVLs and POUs:

```
TYPE
    Axis_t : STRUCT
        pos   : REAL;
        speed : REAL;
        ready : BOOL;
    END_STRUCT;
    Mode_t : (Idle, Auto, Manual);      // enumeration
    Recipe_t : ARRAY[1..5] OF INT;      // named array
END_TYPE
```

In the program, access them as `Axis1.speed`, `Recipe[2]`, `Mode := Auto;`. A new data type starts as a structure with only `x : INT;`, which you can rewrite directly.

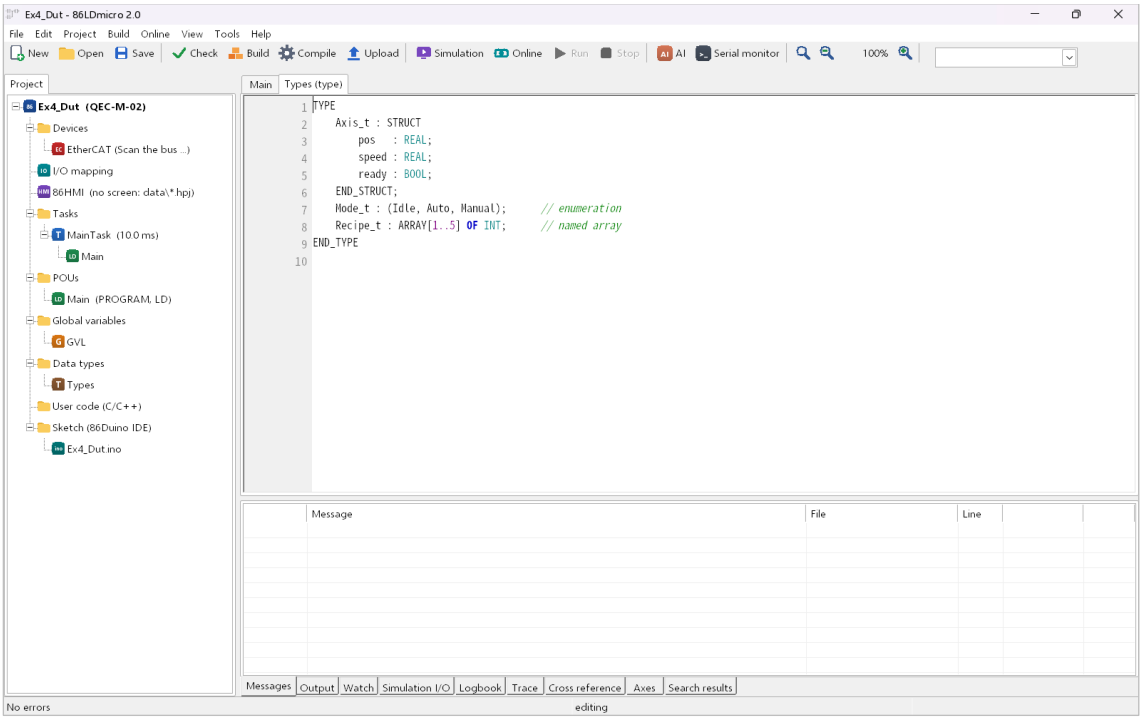


Figure 4-11 The data type editor: structure, enumeration and named array

4.5.5 Ways to add variables

Method	Operation	Where the variable goes
1. Edit the GVL directly	Double-click "Global variables > GVL" in the project tree and type `name : type;`	Global
2. Declaration area of a POU	Type it in the `VAR ... END_VAR` at the top of the LD / ST / SFC tab	Local (usable only in that POU)
3. I/O mapping	Type a new name in the "Variable" field when adding an I/O mapping (section 4.6)	Added automatically to the first GVL
4. Common circuits	New names in the form when inserting a common circuit (section 7.1)	Inputs / outputs / internal bits go to the GVL; timers and the like go to the VAR of that POU
5. 86HMI binding	Type a new name in the "Variable" field when binding a widget (section 13.2)	Added automatically to the GVL
6. Excel import	"Import from Excel..." on the I/O mapping page (section 4.6)	The specified GVL (RETAIN ones in the RETAIN section)
7. Code inserted by the AI assistant	Click "Insert the code" on code in an AI answer (section 15.3)	New variables in the VAR block are declared automatically

*__Tip:__ Naming convention: names starting with `X` are inputs, `Y` outputs, `R` internal bits, `T` timers, `C` counters, `M` numeric values (the same as LDmicro 1.1), so you can tell them apart at a glance when mapping I/O. Names may contain only letters, digits and _, must start with a letter, are not case-sensitive, and must not be IEC reserved words (Appendix C).

4.6 I/O mapping

A PLC program only reads and writes variables; the mapping between variables and physical points (local I/O, EtherCAT PDOs, 86EVA, Arduino pins, Modbus) is set on the "I/O mapping" page, so the program does not need to know the wiring. Double-click "I/O mapping" in the project tree (or choose "Project > I/O mapping") to open it:

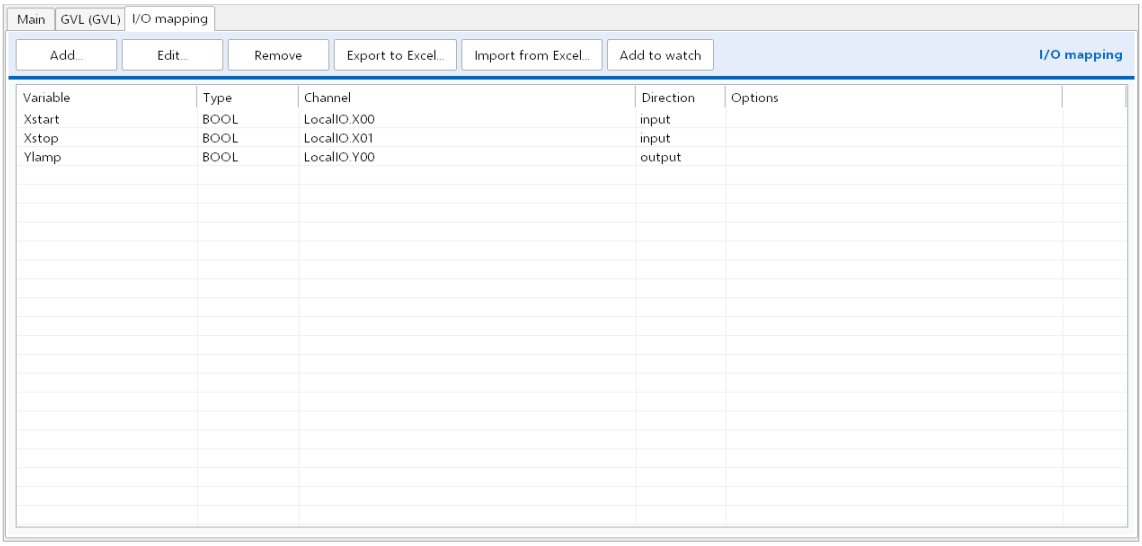


Figure 4-12 The I/O mapping page: variable, type, channel, direction, options; six buttons at the top

Button	Function
Add...	Adds a mapping (the "A new link" window)
Edit...	Changes the selected mapping; double-clicking a row does the same
Remove	Removes the selected mapping (the variable itself is not deleted)
Export to Excel...	Saves the variables and mappings as .xlsx or .csv
Import from Excel...	Declares variables and sets up mappings from .xlsx or .csv
Add to watch	Adds the variables of the selected rows (Ctrl / Shift for multiple selection) to the "Watch" window

4.6.1 Adding an I/O mapping

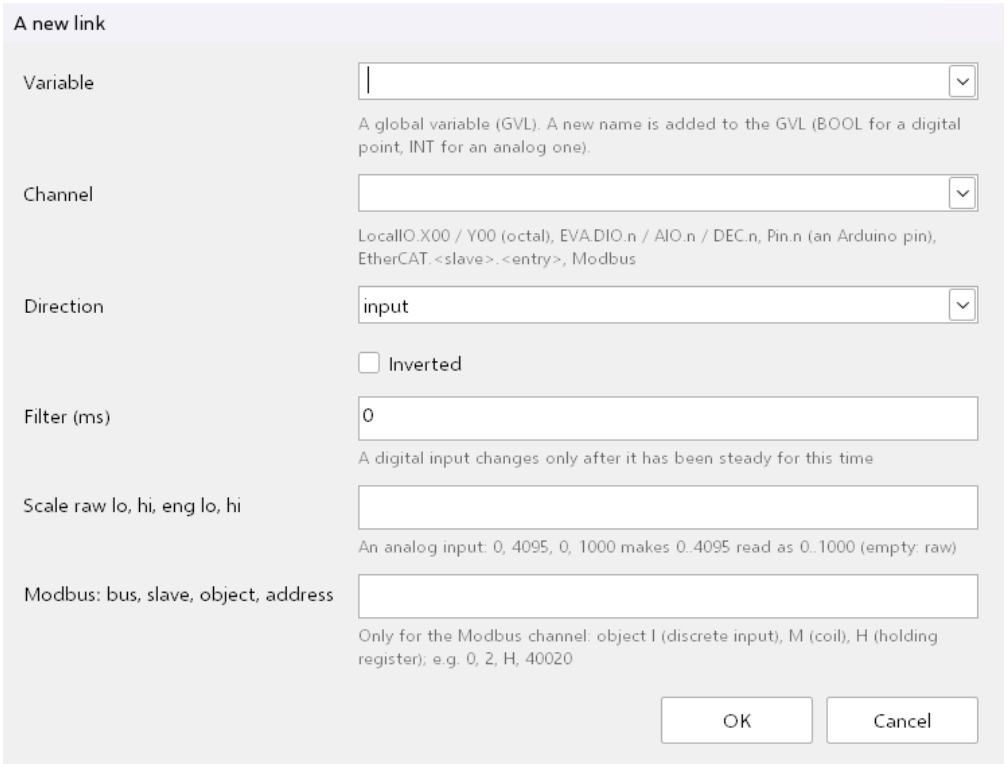


Figure 4-13 "A new link": variable, channel, direction, invert, filter, scaling, Modbus

Field	Description
Variable	A global variable (the drop-down lists the GVL variables). A new name is added automatically to the first GVL: as BOOL for a digital point, INT for an analog point, the PDO's type for an EtherCAT entry, with the channel name as the comment
Channel	`LocalIO.X00` to `X17` / `Y00` to `Y17` (local, octal), `EVA.DIO.n` / `AIO.n` / `DEC.n`, `Pin.n` (Arduino pin), `EtherCAT.<SubDevice>.<entry>`, `Modbus`; the drop-down lists the available channels
Direction	Input / output. For LocalIO and EtherCAT the direction is determined by the channel; for the others by this field
Invert	Inverts a digital point (for example a button wired normally closed)
Filter (ms)	A digital input changes only after it has stayed stable for this time (debouncing)
Scale raw lo, hi, eng lo, hi	For analog inputs, for example `0, 4095, 0, 1000` scales 0..4095 to 0..1000; empty means the raw value
Modbus: bus, slave, object, address	Modbus channels only: object I (discrete input), M (coil), H (holding register), for example `0, 2, H, 40020` (add a modbus device first, section 4.4)

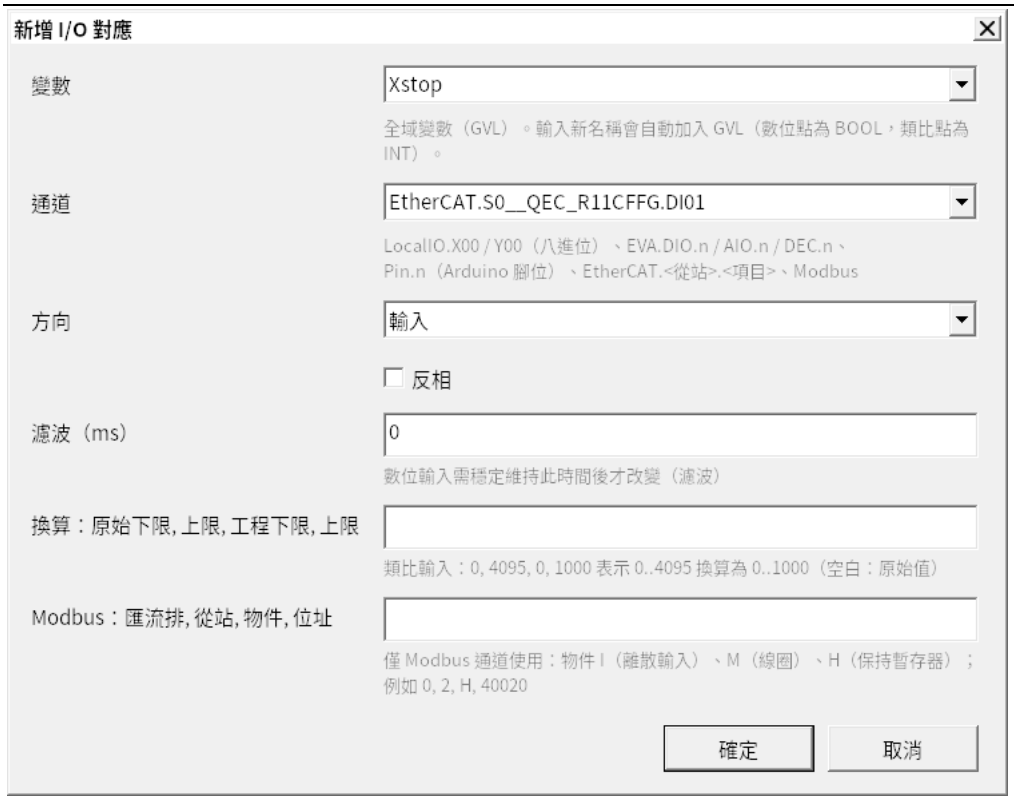


Figure 4-14 Mapping an EtherCAT channel: Xstop → EtherCAT.S0__QEC_R11CFFG.DI01

EtherCAT channels appear only after an ENI has been read in (Chapter 11); you can also right-click a PDO entry in the project tree and choose "Link to a variable...".

4.6.2 Adding to the watch

On the I/O mapping page, select one or more rows (Ctrl / Shift for multiple selection) and click **Add to watch** at the top or right-click > **Add to watch**; the variables are added to the "Watch" window and the status bar shows "N variables added to the watch window". During simulation or online you can see their values and force them right there. The context menu also has **Add...**, **Edit...** and **Remove**.

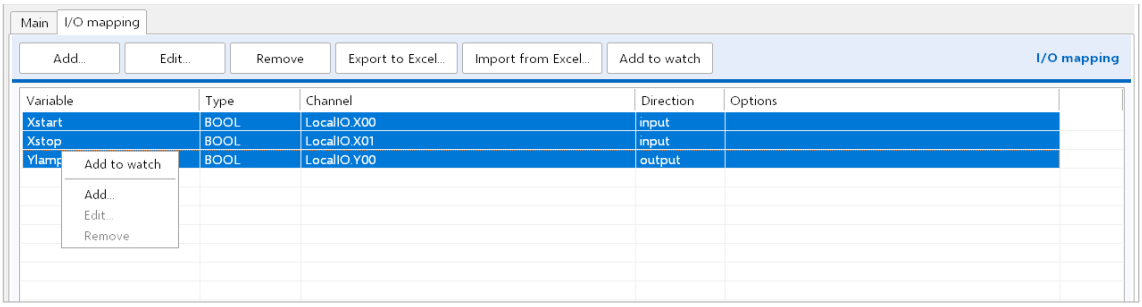


Figure 4-15 The I/O mapping page: right-clicking three selected rows > "Add to watch"

4.6.3 Importing / exporting the I/O table with Excel

When there are many I/O points, you can plan them in Excel and import them all at once.

1. On the I/O mapping page click **Export to Excel...** (or "Project > Export the I/O table to Excel..."), and save as .xlsx (.csv UTF-8 also works). The file opens in Excel automatically after saving.
2. The "IO" worksheet has one row per variable; the "Channels" worksheet lists all available channels and the variables currently mapped to them.
3. Plan or change the I/O in Excel (see the table below for the columns) and save.
4. Click **Import from Excel...** (or "Project > Import the I/O table from Excel...") and choose the file. The header row may be in Chinese or English, and the columns may be in any order. Confirm "N rows: declare the new variables in the GVL and set their I/O mapping (the links of these variables are replaced)?" with **Yes**.
5. When it is done, a summary of the result is shown (variables added and changed, rows skipped). To undo, do not save and reopen the project.

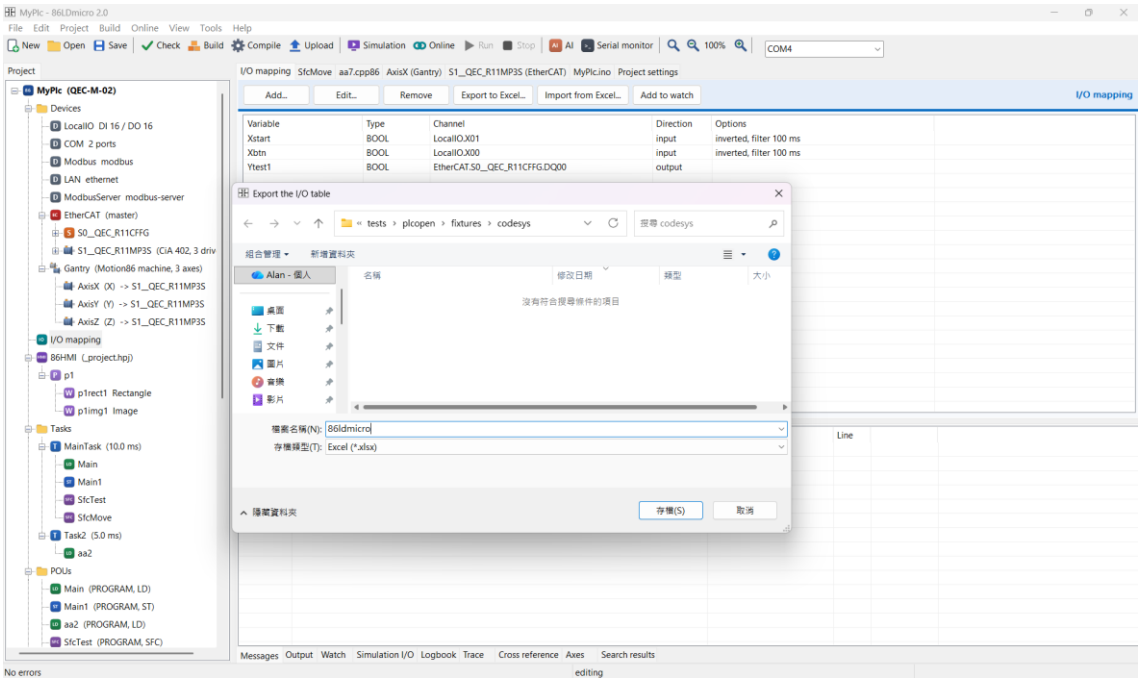


Figure 4-16 The exported I/O table (Excel): the "IO" worksheet

Column	Description
Variable	The variable name (required)
Type	BOOL, INT, REAL…; if empty, a new variable is BOOL and an existing variable is not changed
Initial value, comment	Written into the declaration
Channel	As in "A new link"; empty means the mapping of this variable is removed
Direction, invert, filter ms, scale (raw lo, raw hi, eng lo, eng hi)	As in "A new link"
RETAIN	A new variable goes into the RETAIN section of the GVL
GVL	Which GVL a new variable goes into (empty: the first one)
Other (JSON)	Other mapping attributes, for example bus, slave, object and address for Modbus

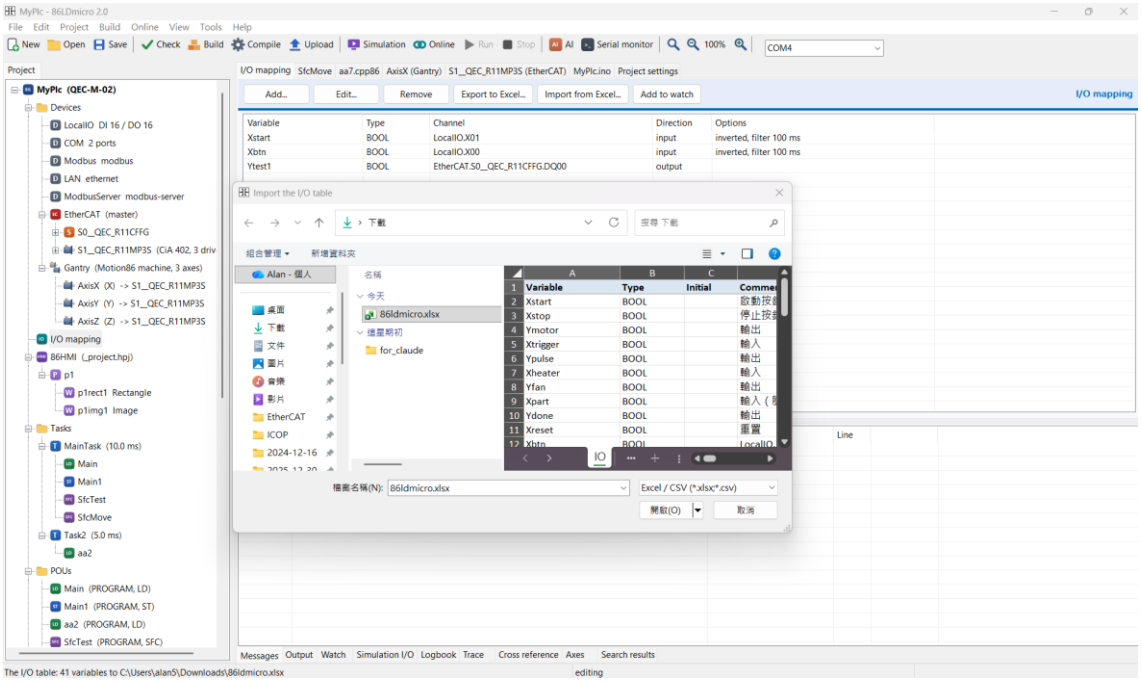


Figure 4-17 The confirmation before importing from Excel

***Tip:** Importing is available only while editing; during simulation or online you are told "Stop the simulation / go offline to edit".
If the same variable appears twice in the table, only the first row is used; rows with invalid names are skipped. Both are listed in the result summary. Command line: `86ldmicro-cli iotable export|import <project> <file>` (section 15.5).

4.7 Adding POUs and user code

Add them from the "**Project**" menu (or the context menu of the corresponding node in the project tree): enter a name and click **OK**, and the new file opens immediately. Names may contain only letters, digits and _, must start with a letter, and must not be the same as any other POU, GVL, data type or user code file in the project.

Command	File created	Default name	Description
Add a PROGRAM (ladder / ST / SFC)...	`pou\<name>.lad / .st / .sfc`	Program	A new PROGRAM is added automatically to the first task
Add a FUNCTION_BLOCK (ladder / ST)...	`pou\<name>.lad / .st`	MyFb	A reusable function block with internal state (section 8.3)
Add a FUNCTION (ST)...	`pou\<name>.st`	MyFunction	A function without memory (section 8.3)
Add a global variable list...	`gvl\<name>.gvl`	GVL2	An empty `VAR_GLOBAL ... END_VAR`
Add a data type...	`dut\<name>.dut`	MyType	A structure template (section 4.5)
Add a user code file (C/C++)...	`usercode\<name>.cpp86`	MyCode	With explanations and hook function templates (Chapter 14)

*__Important:__ The default name of a new PROGRAM, `Program`, is the same as the IEC reserved word PROGRAM (names are not case-sensitive), so clicking OK without changing it causes an error in the check. Change it to a meaningful name, for example `Filling` or `Conveyor`.

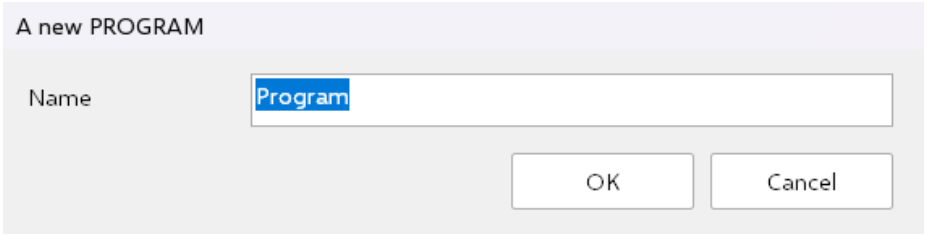


Figure 4-18 "A new PROGRAM": entering the name

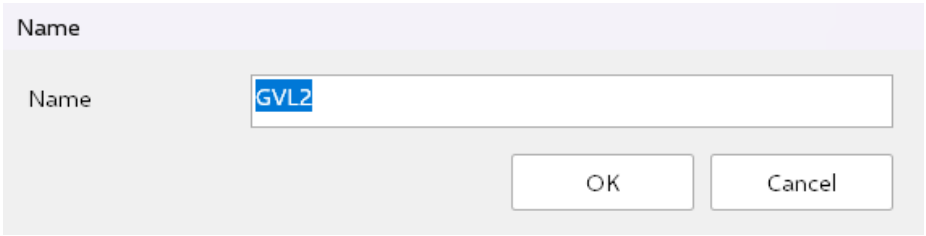


Figure 4-19 "A new global variable list": entering the name (default GVL2)

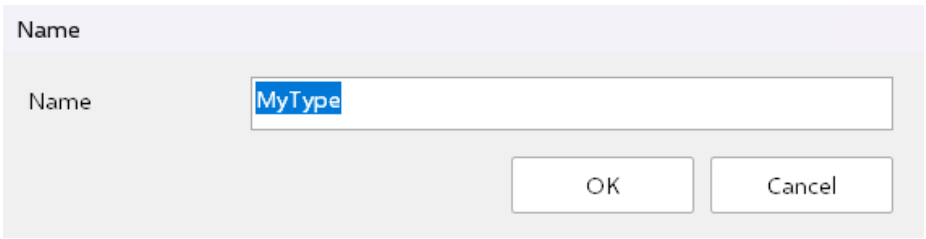


Figure 4-20 "A new data type": entering the name

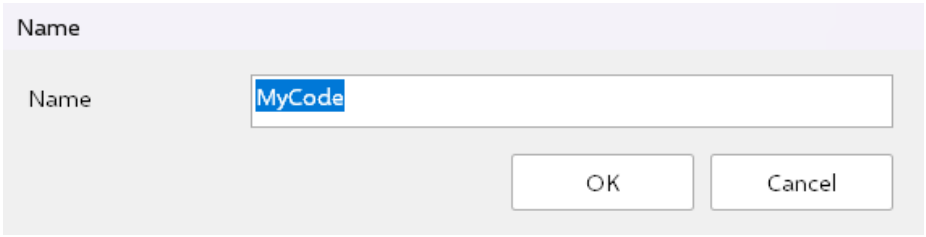


Figure 4-21 "A new user code file (C/C++)": entering the name

The new file contains a template you can edit directly, for example an ST PROGRAM and FUNCTION_BLOCK:

```
PROGRAM Filling
VAR
END_VAR

END_PROGRAM

FUNCTION_BLOCK MyFb
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
END_VAR

END_FUNCTION_BLOCK
```

The FUNCTION template is `FUNCTION MyFunction : INT` (returns INT; the last line is `MyFunction := 0;`); the SFC template is an initial step `Init` and a step `Step1` that jumps back after 1 second (section 8.4).

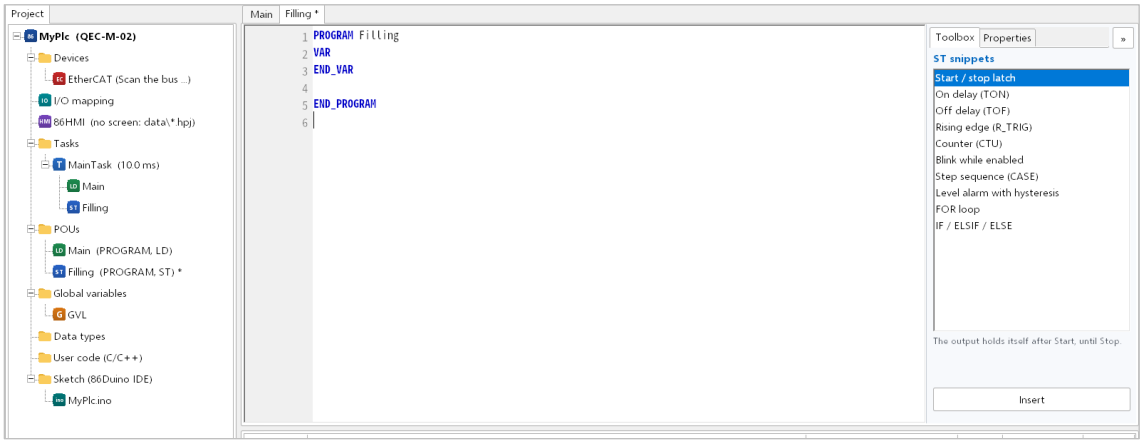


Figure 4-22 The new ST PROGRAM Filling: added to MainTask and opened

4.7.1 User code (C/C++)

User code files (`.cpp86`) are compiled into `plc86.cpp` together with the PLC. A new file starts with an explanation: PLC variables are written as `GVL(name)` (global variables), `ST(program.variable)` (a program's variables) and `FUN(name)(arguments)` (the project's ST FUNCTIONs), followed by templates for hook functions such as `plc86_user_setup`, `plc86_user_before`, `plc86_user_after` and `plc86_user_loop`. See Chapter 14 for details.

Main | SendMsg.cpp86

```
1  // SendMsg: when Nmsg changes, send one line on Serial1
2
3  void plc86_user_setup(void) {
4      Serial1.begin(9600);              // COM1, 9600 bps
5  }
6
7  // every pass of loop(): non-blocking work only (no delay())
8  void plc86_user_loop(void) {
9      static int32_t sent = 0;          // number already sent
10     int32_t n = GVL(Nmsg);
11     if(n != sent) {
12         sent = n;
13         Serial1.print("Button pressed, count = ");
14         Serial1.println(n);
15     }
16 }
17
```

Figure 4-23 The user code editor: hook functions and access to PLC variables with GVL()

- Renaming a POU: right-click the POU in the project tree > "Rename..."; the program name in the task is updated too.
- Removing: right-click > "Remove from the project"; the file stays in `data\86ld`.
- The sketch's own `.h` / `.cpp` files: right-click "Sketch (86Duino IDE)" in the project tree > "A new .h / .cpp in the sketch..." (section 14.5).

Chapter 5 Quick start: from a blank project to upload

The following 12 steps demonstrate a minimal but complete workflow: create a sketch in the 86Duino IDE, open 86LDmicro and create a project, declare variables, draw a ladder diagram for "start / stop latch + light the indicator lamp after a 2-second delay", map it to the local I/O of the QEC-M-02, simulate it on the PC first, then build, upload it to the controller and view it online.

The finished behavior: press the start button (X00) once, and the internal bit `Rrun` becomes TRUE and latches; after `Rrun` has been on for 2 seconds, the indicator lamp `Ylamp` (Y00) lights up; press the stop button (X01), and both `Rrun` and the lamp go off.

Variable	Type	I/O mapping	Purpose
`Xstart`	BOOL	`LocalIO.X00` (input)	Start button (NO)
`Xstop`	BOOL	`LocalIO.X01` (input)	Stop button (wired NO, used as an NC contact in the program)
`Rrun`	BOOL	— (internal bit)	Running (latched)
`Ylamp`	BOOL	`LocalIO.Y00` (output)	Run indicator lamp
`tLamp`	TON (IEC function block)	—	2-second delay for the indicator lamp (declared in the VAR of Main)

*__Important:__ After drawing the ladder diagram, you must map `Xstart`, `Xstop` and `Ylamp` to the physical `LocalIO.X00`, `X01` and `Y00` in "I/O mapping" (Step 9). An unmapped variable is just a bit in memory: the check passes (an unmapped input only gives a warning in the build), but the physical I/O does nothing after the upload.

*__Tip:__ Before you start, complete Chapter 2: 86Duino Coding 502 and 86LDmicro are installed, the QEC-M-02 is connected to the PC with a USB cable, and you know its COM port (section 2.4).

5.1 Step 1: Open the 86Duino IDE, create a sketch, select the board and serial port

Open 86Duino Coding 502. Choose "**File > New**" (Ctrl+N) to create a new sketch, then "**File > Save As...**" (Ctrl+Shift+S) to save it as `MyPlc`. The name of the sketch is also the name of the 86LDmicro project later on.



Figure 5-1 The "File" menu of the 86Duino IDE: New, Save As

Choose "**Tools > Board**" > **QEC M02**, then "**Tools > Serial Port**" > the controller's COM port. The lower-right corner of the IDE window shows "QEC M02 on COMx".

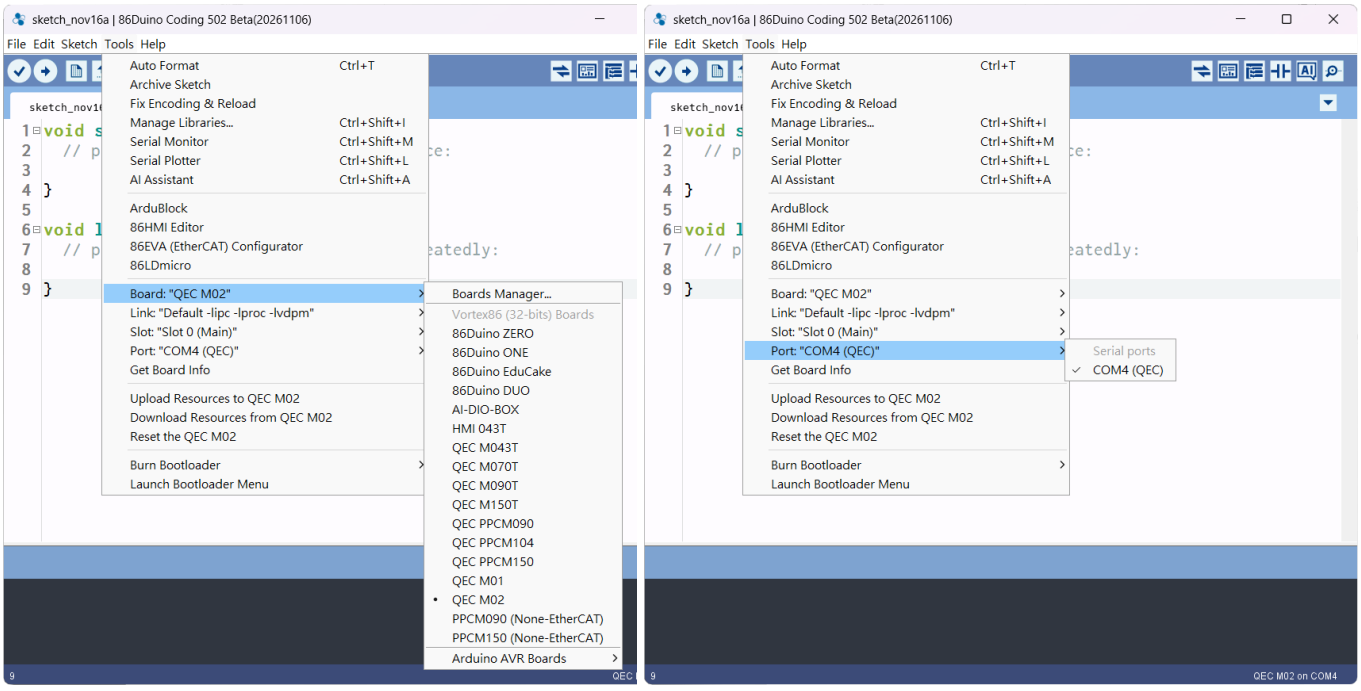


Figure 5-2 & Figure 5-3 "Tools > Board" > QEC M02, "Tools > Serial Port" > the controller's COM port

5.2 Step 2: Open 86LDmicro from the IDE and create a new project

Choose "**Tools > 86LDmicro**", or click the **86LDmicro** button on the right of the IDE toolbar. This sketch has no 86LDmicro project yet, so a question appears; click **Yes** to create a new project.

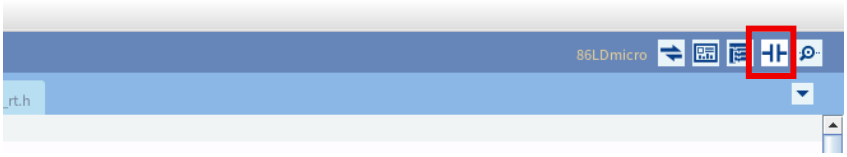


Figure 5-4 The 86LDmicro button on the right of the IDE toolbar

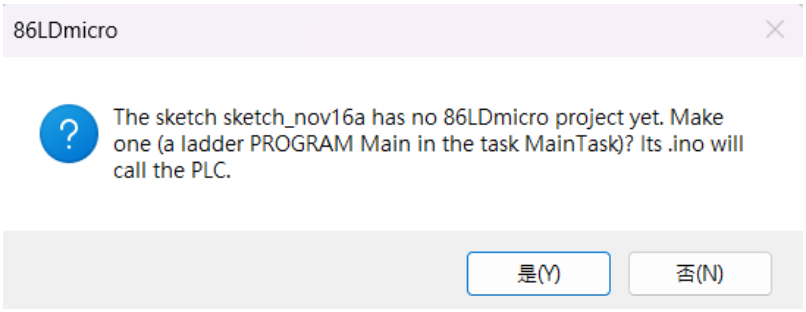


Figure 5-5 The question shown when the sketch has no 86LDmicro project yet

- When opened from the IDE, 86LDmicro automatically uses that IDE's compiler and the serial port selected in the IDE; no further setup is needed.
- `.ino` automatically gets `#include "plc86.h"`, `plc86_setup()` and `plc86_loop()`; the rest of the code is not touched.
- You can also start 86LDmicro on its own: choose "**File > New project...**" (Ctrl+N), enter the name `MyPlc` and the location (by default `Documents\86Duino`), and click **OK** (section 2.5).

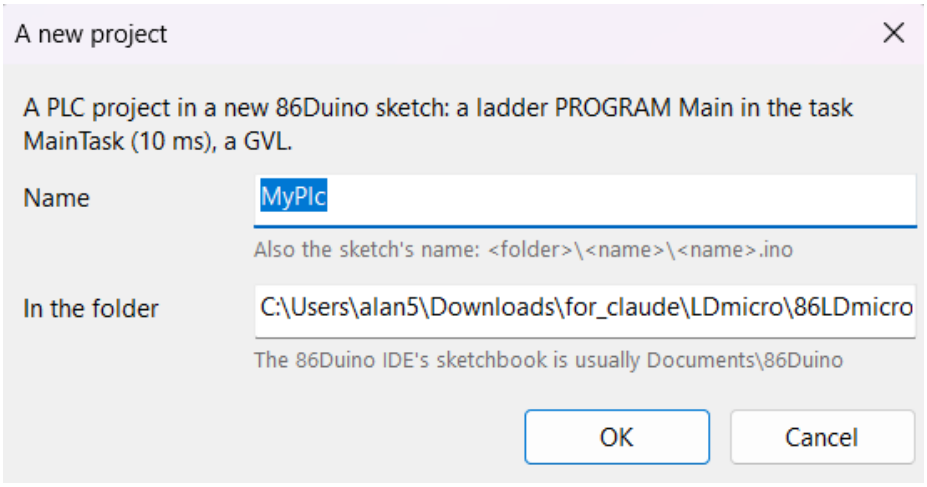


Figure 5-6 "A new project" when started on its own: the name is also the name of the sketch

5.3 Step 3: Get to know the new project: the project tree and the ladder diagram Main

The new project already has the following items, and the ladder diagram `Main` opens automatically:

Project tree node	Contents
Devices	The controller's devices (local I/O, EtherCAT, etc.; section 4.4)
I/O mapping	The mapping between variables and physical channels (Step 9)
Tasks > MainTask (10.0 ms)	Runs every 10 ms and runs `Main`
POUs > Main (PROGRAM, LD)	The ladder program; for now it has only one empty network 1
Global variables > GVL	The global variable list (Step 4)
Sketch (86Duino IDE)	The sketch's `.ino`

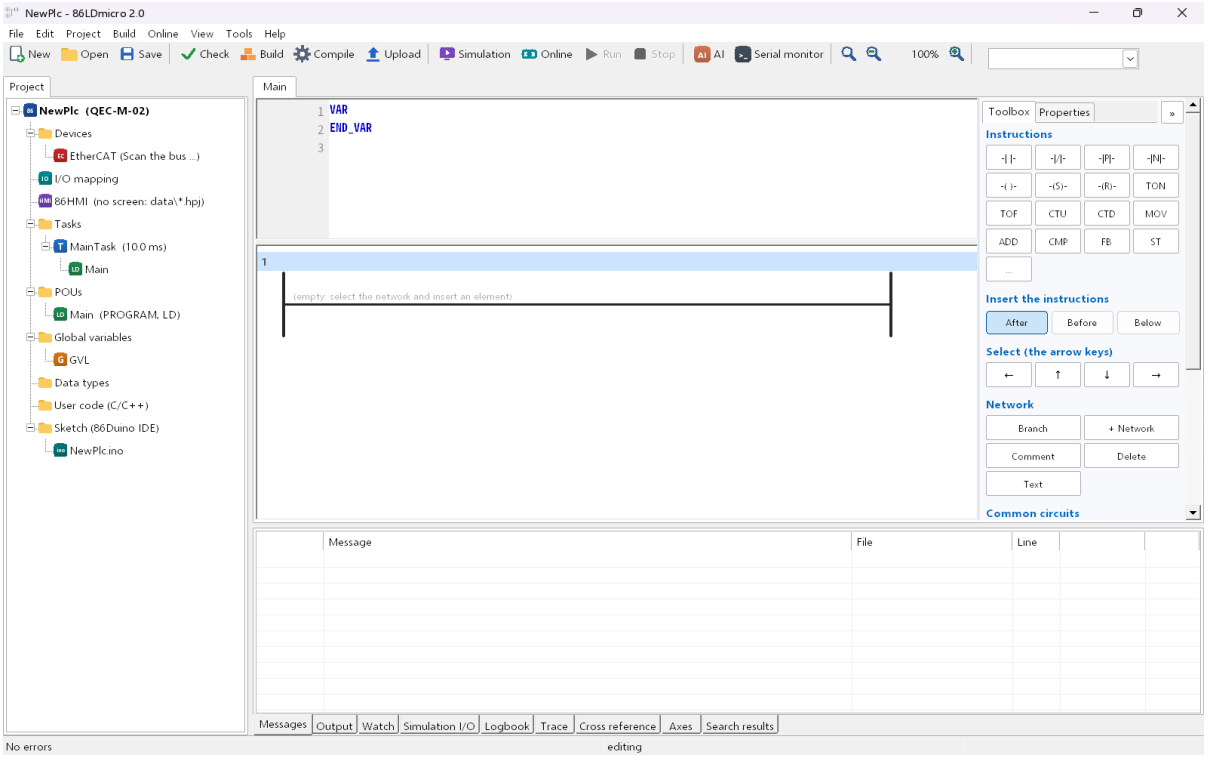


Figure 5-7 The new project: Devices, I/O mapping, Tasks, POUs (Main) and GVL have all been created

The ladder diagram tab has three parts: at the top the **declaration area** (`VAR ... END_VAR`, for this POU's own variables and function block instances), in the middle the **networks**, and in the side panel the **Toolbox** ("Instructions", "Insert the instructions", "Select (the arrow keys)", "Network", "Common circuits"). See section 3.5 for details of each area.

5.4 Step 4: Declare the variables in the GVL

In 86LDmicro 2.0, every variable must be declared with a type first. Double-click "Global variables > GVL" in the project tree, replace the entire contents of the GVL (including VAR_GLOBAL and END_VAR) with the following, and press Ctrl+S to save:

```
VAR_GLOBAL
    Xstart : BOOL;      // start button (NO)
    Xstop  : BOOL;      // stop button (wired NO)
    Rrun   : BOOL;      // running
    Ylamp  : BOOL;      // run indicator lamp
END_VAR
```

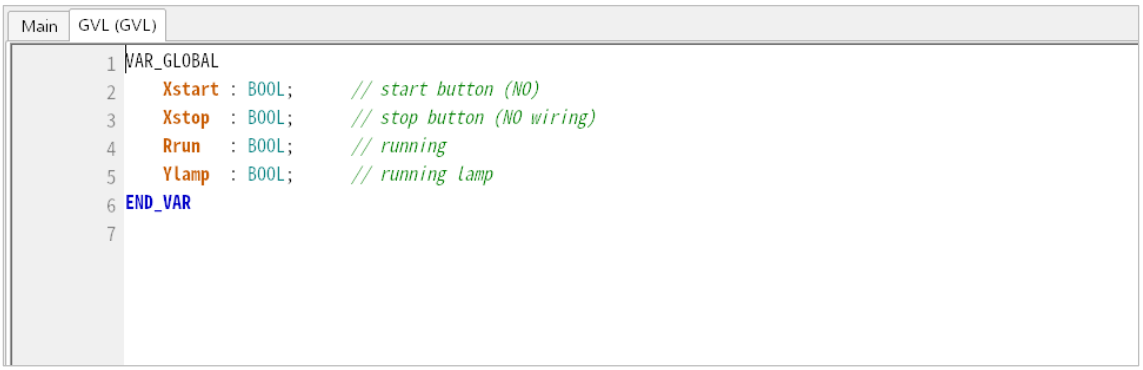


Figure 5-8 GVL: declaring Xstart, Xstop, Rrun and Ylamp

***Tip:** Naming convention: names starting with X are inputs, Y outputs and R internal bits (as in LDmicro 1.1), so you can tell them apart at a glance when mapping I/O. The names are only a convention; the type is determined by the declaration.
When the ladder diagram uses a name that has not been declared yet, the element is shown with a red background as an error (section 6.0). See section 4.5 for the declaration syntax and data types.

5.5 Step 5: Insert the NO contact Xstart

Double-click "**POUs > Main**" in the project tree to return to the ladder diagram. Make sure "**Insert the instructions**" in the side panel is set to **After**.

1. Click network 1 (the empty network) to select it.
2. Click **-| |-** (NO contact) under "**Instructions**" in the side panel, or simply press **C** on the keyboard.
3. In the window that appears, the drop-down of the "Variable" field lists all declared variables; choose `Xstart` and click **OK**.

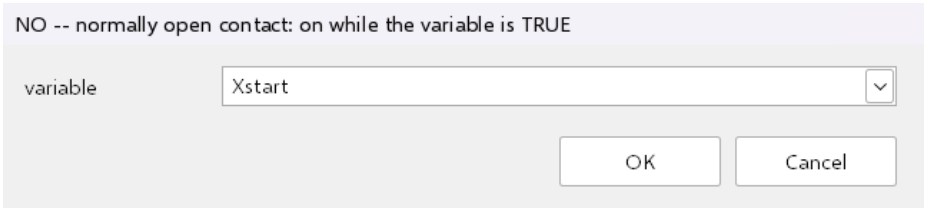


Figure 5-9 The parameter window of an NO contact: choosing the variable

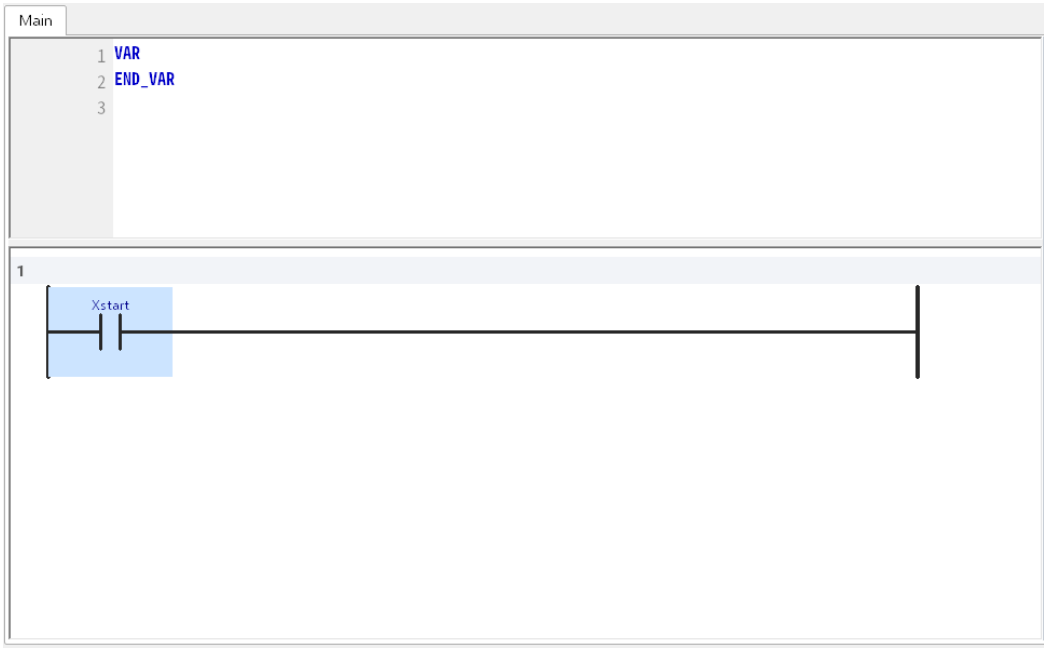


Figure 5-10 The NO contact Xstart appears in network 1 (the blue background marks the current selection)

5.6 Step 6: Add the NC contact Xstop and the coil Rrun in series

1. While `Xstart` is still selected, click **-|/|-** (normally closed contact, key **N**), choose the variable `Xstop` and click **OK**. A new element is inserted after the selected element, that is, **in series**.
2. Click **-( )-** (coil, key **L**), choose the variable `Rrun` and click **OK**. The coil is placed automatically at the right side of the network (next to the right power rail).



Figure 5-11 The parameter window of a coil (OUT)

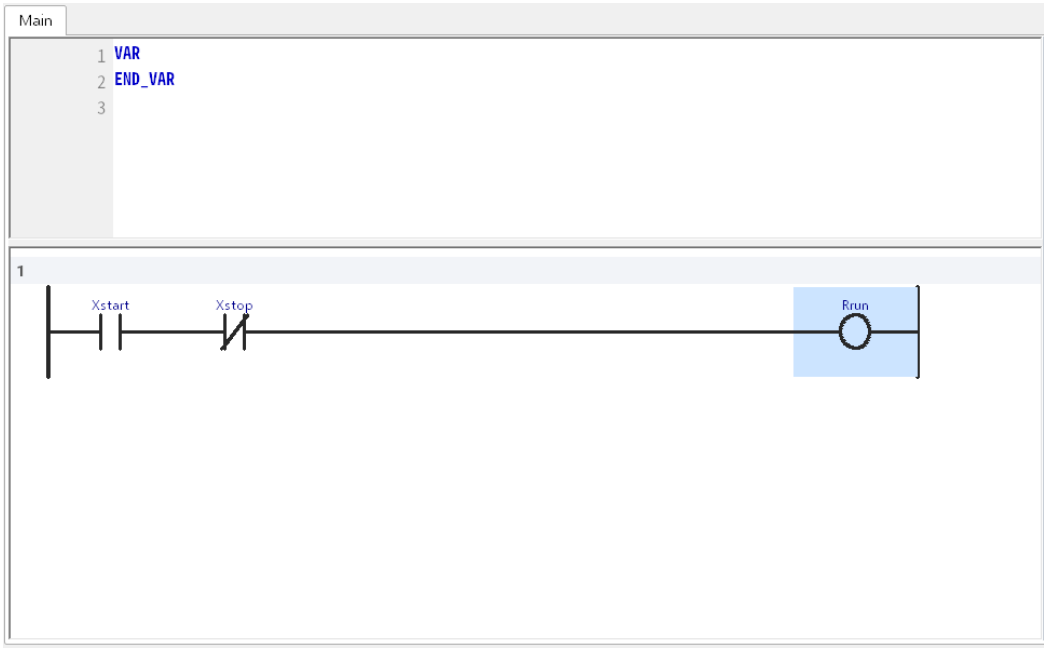


Figure 5-12 Xstart and Xstop in series drive the coil Rrun

*__Tip:__ If the stop button is wired normally closed (a broken wire then acts like pressing stop, which is safer), use the normally open contact `NO(Xstop)` in the program instead, or tick "Invert" in the I/O mapping in Step 9.

5.7 Step 7: Add an Rrun contact in parallel to form a latch circuit

After the start button is released, `Rrun` must keep the rung conducting through its own contact, so add a normally open `Rrun` contact in parallel below `Xstart`:

1. Click ← in "Select (the arrow keys)" in the side panel (or press ← on the keyboard) twice to move the selection to `Xstart`.
2. In "Insert the instructions", click **Below**.
3. Click **-| |-** (or **C**), choose the variable `Rrun` and click **OK**: the `Rrun` contact is placed in parallel below `Xstart`.
4. Set "Insert the instructions" back to **After**.

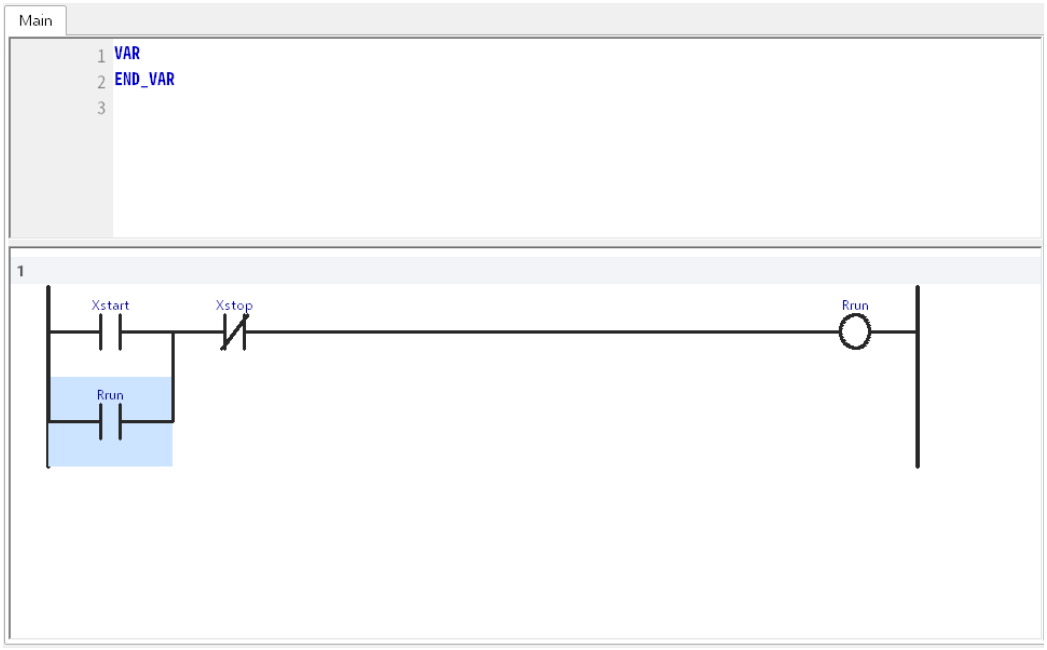


Figure 5-13 Rrun in parallel below Xstart: pressing Xstart latches Rrun, pressing Xstop releases it

*__Tip:__ Shortcut: after selecting `Xstart`, just press **B** on the keyboard to add a normally open contact in parallel below it (then enter the variable `Rrun`), without switching "Insert the instructions".

5.8 Step 8: Add network 2: a TON timer turns on the lamp after a 2-second delay

First declare an IEC TON timer instance in the declaration area above `Main`: between `VAR` and `END_VAR`, enter `tLamp : TON;`.

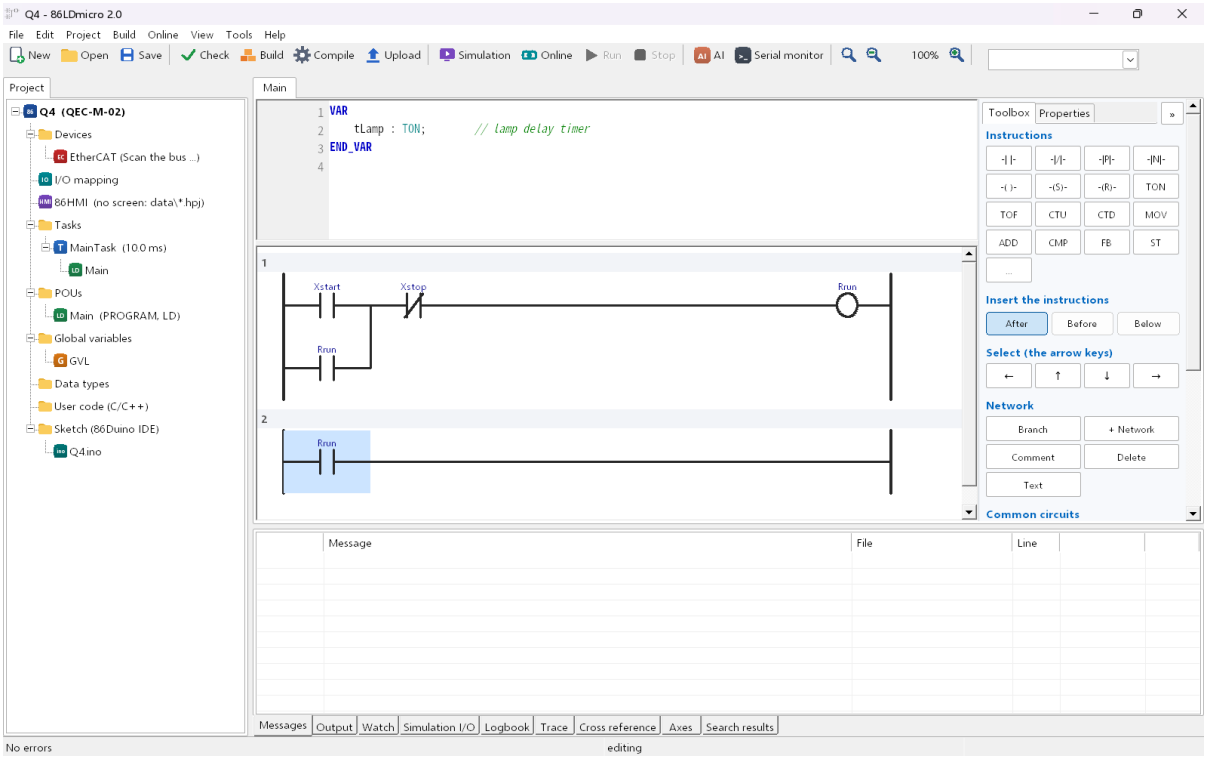


Figure 5-14 tLamp : TON (the lamp delay timer) added in the declaration area

1. Click the title bar of network 1 (the row numbered 1) to select the network, then click **+ Network** in "Network" in the side panel (or press **Ctrl+Enter**) to add network 2 below and select it.
2. Click **-| |-** (**C**) and choose the variable `Rrun`.
3. Click **FB** (function block, key **K**), enter `CALL(tLamp, IN := @, PT := T#2s, Q => @)` in the "Element" field and click **OK**. `@` stands for the power flow of the rung: the power flow coming in from the left goes to IN, and the output Q becomes the power flow going to the right.
4. Click **-( )-** (**L**) and choose the variable `Ylamp`.

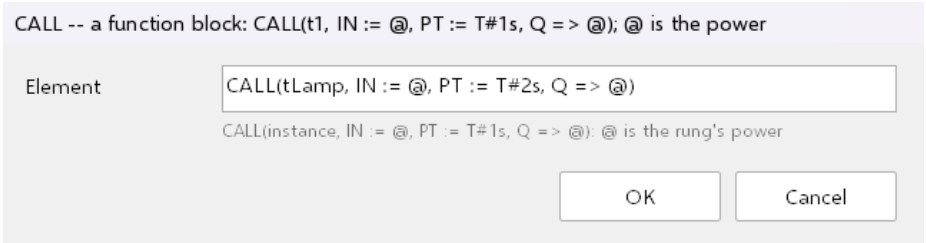


Figure 5-15 The parameters of the function block (CALL): IN takes the power flow, PT is 2 seconds, the Q output goes back to the power flow

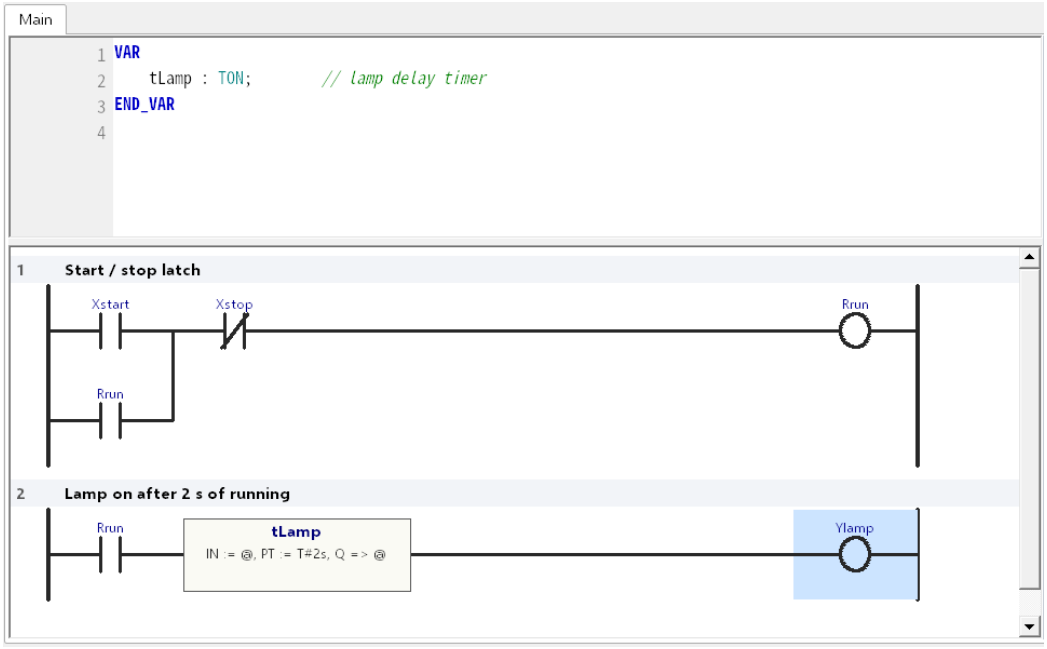


Figure 5-16 The two finished networks: Ylamp turns on after Rrun has been on for 2 seconds

Finally, give the networks titles: double-click the title bar of network 1, enter `Start / stop latch` in "Title" and click **OK**; for network 2, enter `Lamp on after 2 s of running`. "Body (text)" at the bottom of the window is the .lad text of this network.

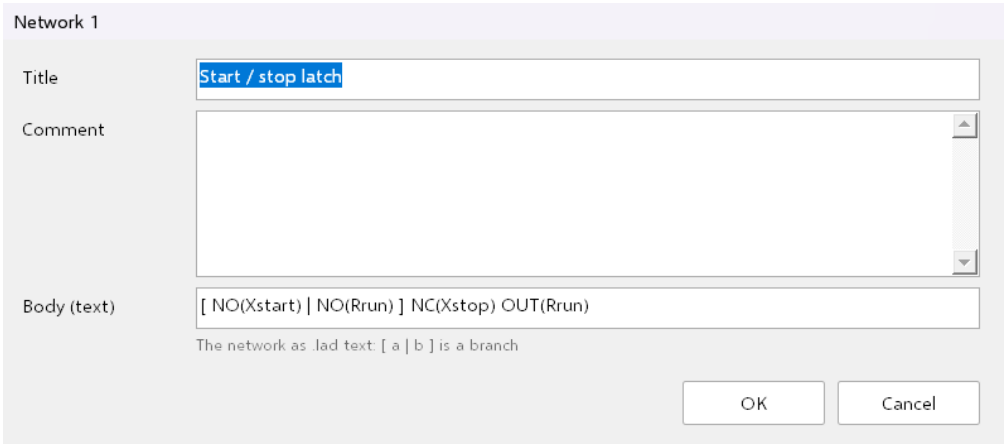


Figure 5-17 The title, comment and body (.lad text) of a network

Click **Text** in "Network" in the side panel (Ctrl+T) to see the .lad text of the whole `Main`; it should be the same as below (click **Back to the diagram** to return):

```
PROGRAM Main
VAR
    tLamp : TON;        // lamp delay timer
END_VAR
NETWORK 'Start / stop latch'
    [ NO(Xstart) | NO(Rrun) ] NC(Xstop) OUT(Rrun)
NETWORK 'Lamp on after 2 s of running'
    NO(Rrun) CALL(tLamp, IN := @, PT := T#2s, Q => @) OUT(Ylamp)
END_PROGRAM
```

5.9 Step 9: I/O mapping: connect the variables to the physical I/O of the QEC-M-02

For now, `Xstart`, `Xstop` and `Ylamp` are only variables in memory. They must be mapped to physical points of the controller, or the buttons and the lamp will not work after the upload.

1. In the project tree, double-click "**I/O mapping**" and click **Add...**.
2. For "Variable" choose `Xstart`, for "Channel" choose `LocalIO.X00`, and click **OK** (the direction of local I/O is set by the channel; X is an input).
3. Click **Add...** again: `Xstop` to `LocalIO.X01`.
4. Click **Add...** again: `Ylamp` to `LocalIO.Y00` (output).
5. Press **Ctrl+S** to save.

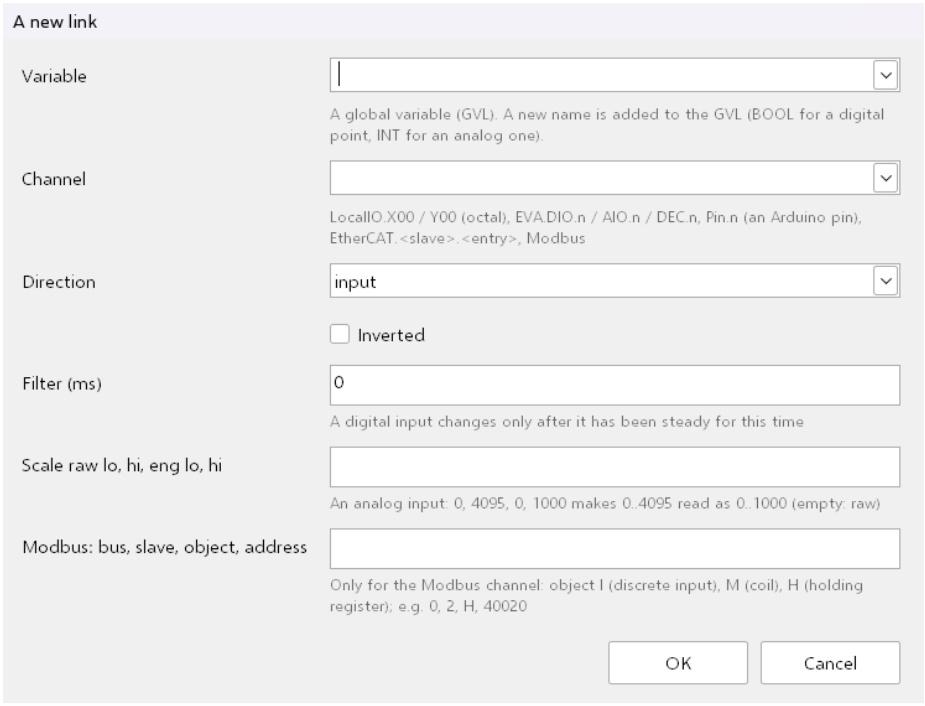


Figure 5-18 Adding an I/O mapping: variable, channel, direction, invert, filter, scaling

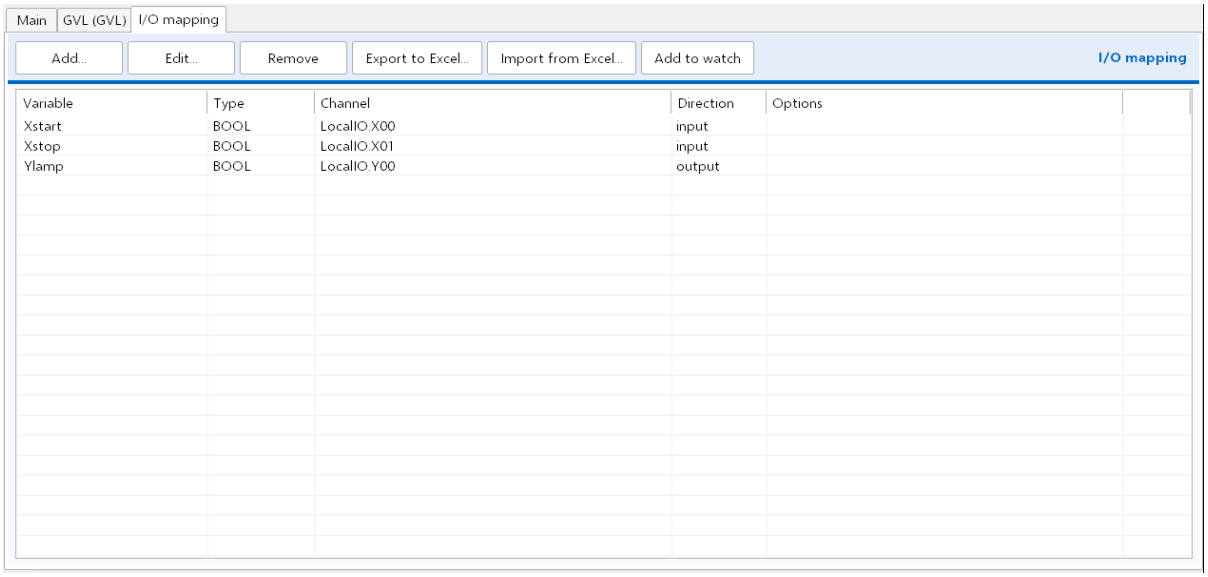


Figure 5-19 The I/O mapping page: each row maps a variable to a channel

- The local I/O channels of the QEC-M-02 are `LocalIO.X00` to `X17` (inputs) and `LocalIO.Y00` to `Y17` (outputs), numbered in **octal** (X07 is followed by X10).
- `Rrun` and `tLamp` are internal variables and need no mapping.
- You can also right-click a variable in the ladder diagram and choose "**I/O mapping of Xstart...**" to open the mapping window directly. For the channels of EtherCAT modules, see Chapter 11; for the full description of I/O mapping, see section 4.6.

5.10 Step 10: Check (Shift+F7) and simulate (F5)

Checking the program logic on the computer before uploading saves a lot of "flash, test, modify again" time.

1. Click **Check** on the toolbar (Shift+F7). Errors are listed in the "**Messages**" window at the bottom; double-click one to jump to the element at fault. When the status bar shows "No errors", you can continue.
2. Click **Simulate** on the toolbar (F5). The status bar shows "SIMULATION: RUN" and the scan count, and the program starts running on the computer.
3. Switch to the "**Simulation I/O**" window at the bottom: it lists the physical point of each I/O mapping. Double-click `LocalIO.X00` once (ON) and once more (OFF) to simulate a press of the start button.
4. Watch the ladder diagram: line segments with power flow turn green, and TRUE / FALSE is shown below the elements. `Rrun` becomes TRUE and stays TRUE; the `tLamp` box times, and after 2 seconds `Ylamp` becomes TRUE and `LocalIO.Y00` shows ON in "Simulation I/O".
5. Double-click `LocalIO.X01` twice (ON, OFF) to simulate pressing stop: `Rrun` and `Ylamp` return to FALSE.
6. Click **Simulate** (F5) again to end the simulation and return to editing.

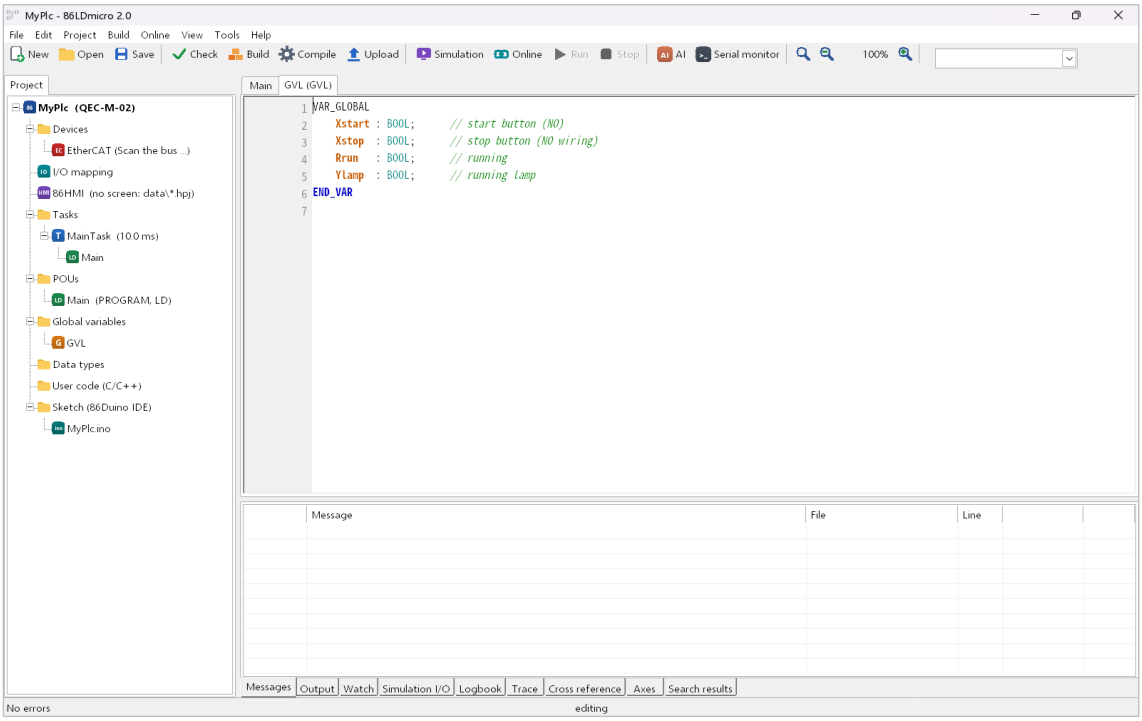


Figure 5-20 Check finished: no errors in the Messages window, and the status bar shows "No errors"

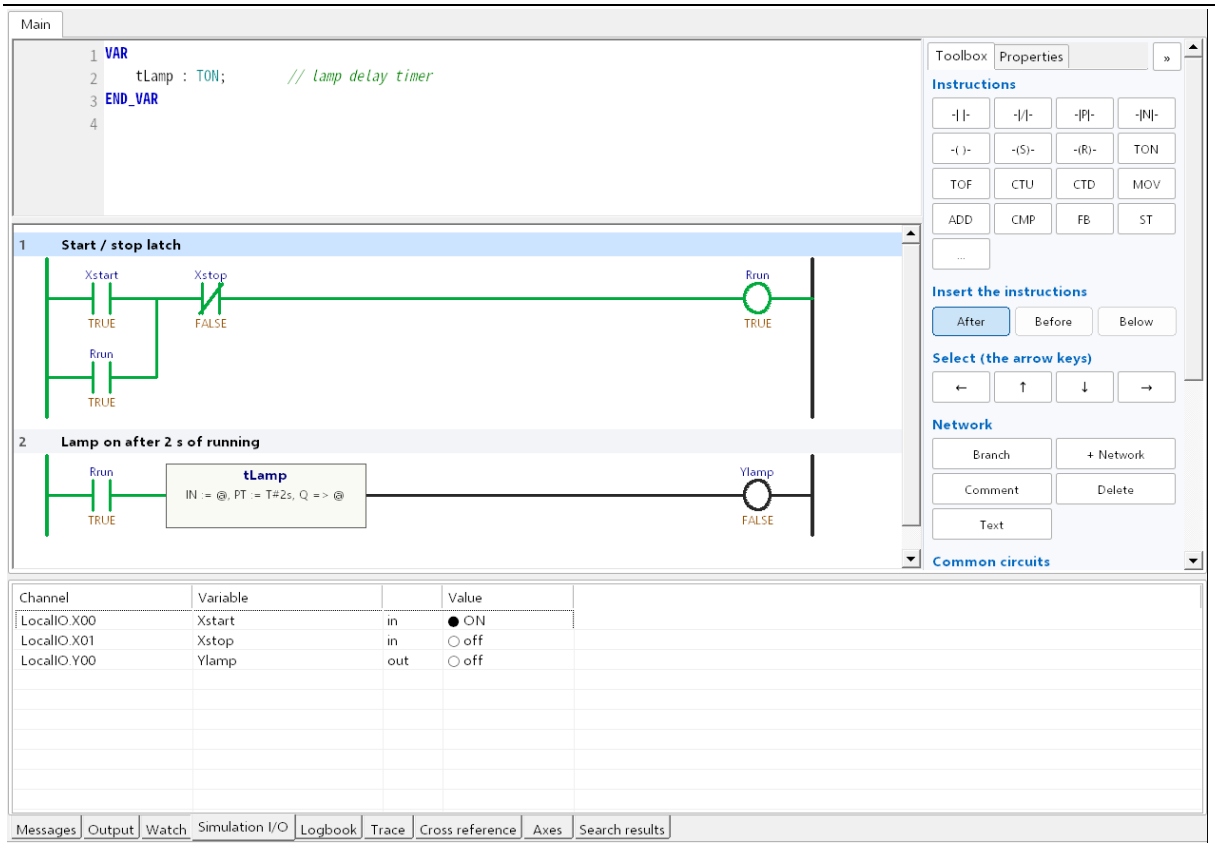


Figure 5-21 Simulation started: "Simulation I/O" lists the physical points such as LocalIO.X00 (Xstart)

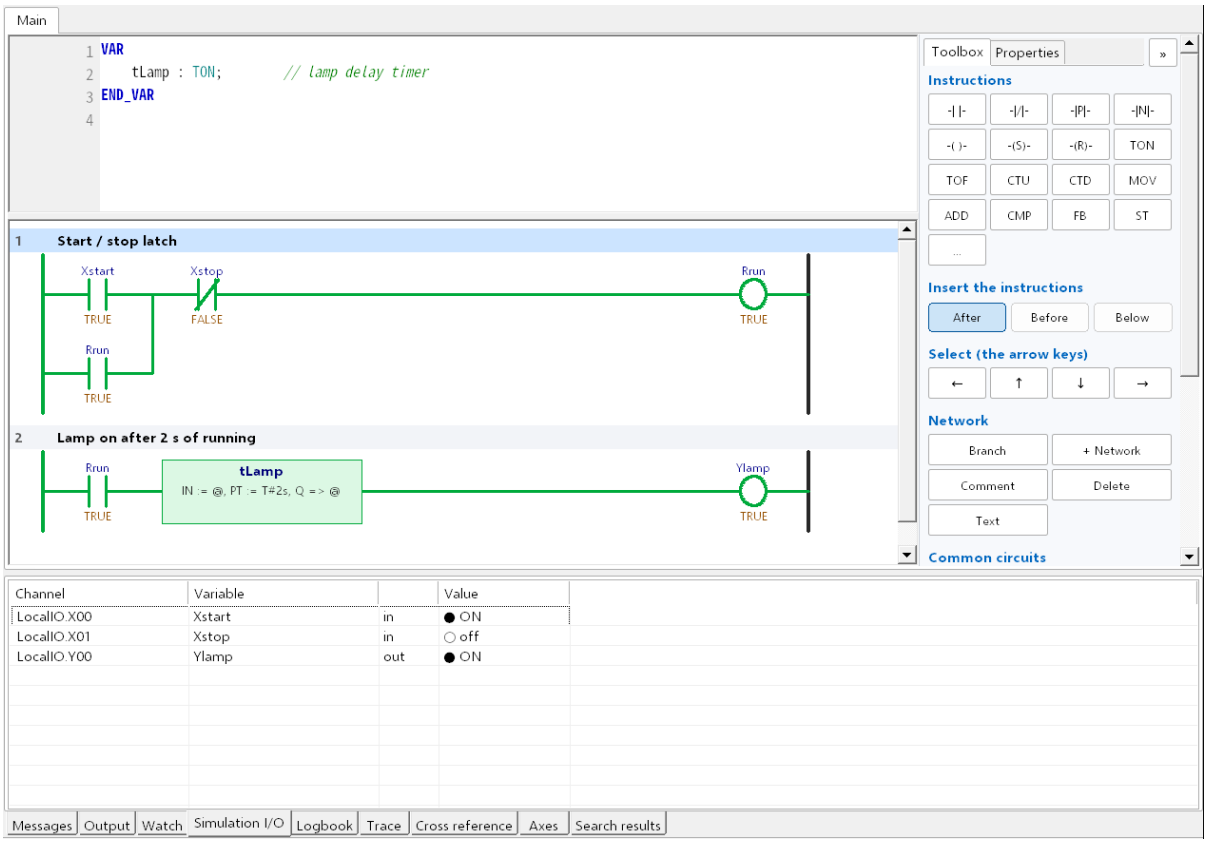


Figure 5-22 During simulation: green is power flow, the timer box shows the elapsed time, and LocalIO.Y00 is ON

To see several variables at once, switch to the "**Watch**" window, enter a variable name at the top (for example `Rrun` or `Main.tLamp.ET`) and click **Add** to see its live value (section 9.3).

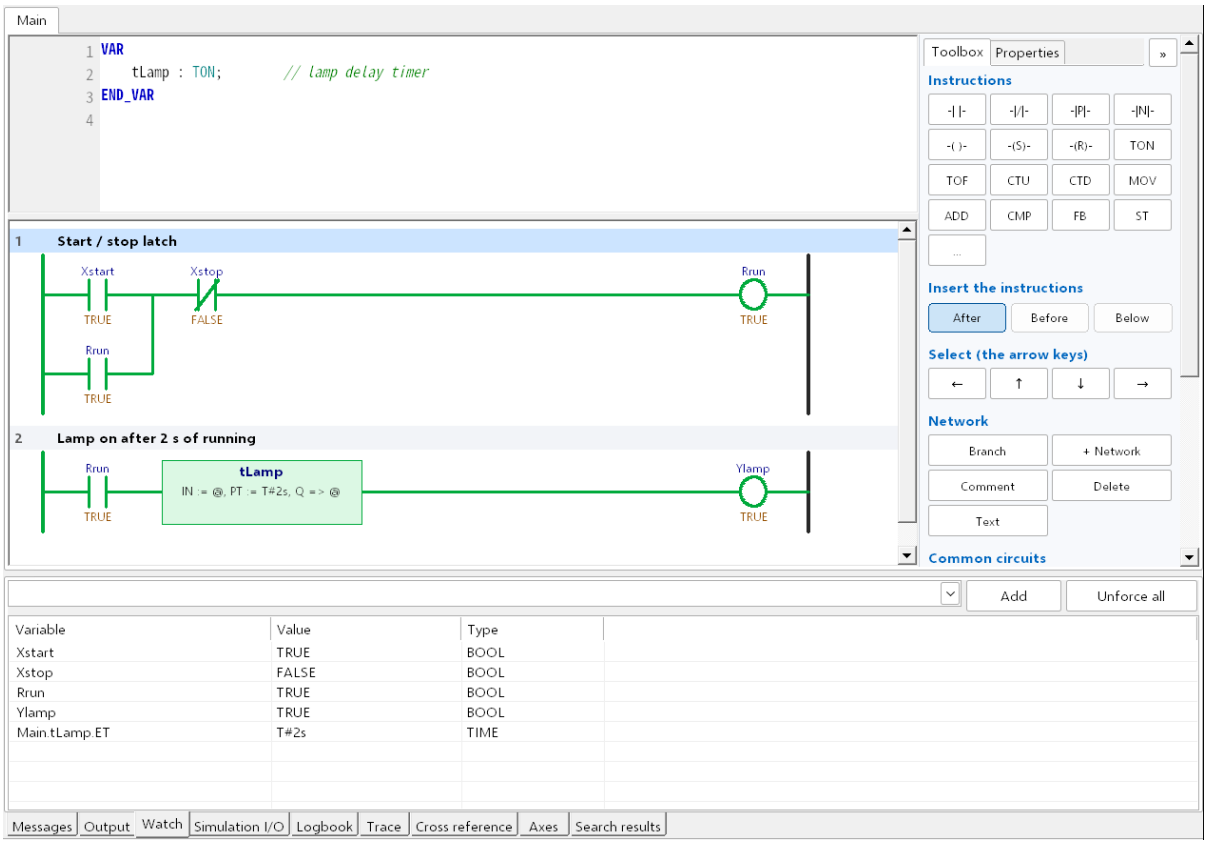


Figure 5-23 The Watch window: live values of the added variables

*__Note:__ During simulation, a variable mapped to a physical input (for example `Xstart` on `LocalIO.X00`) is set by "Simulation I/O"; if you write it directly in Watch, the input value overwrites it in the next scan. Switch inputs in "Simulation I/O" (section 9.2).

5.11 Step 11: Build, compile and upload (F8)

1. Make sure the QEC-M-02 is connected to the computer with a USB cable, and choose the controller's COM port in the serial port drop-down list at the right of the toolbar (when 86LDmicro is opened from the IDE, the port chosen in the IDE is already filled in).
2. Click **Upload** on the toolbar (F8). 86LDmicro then **builds** (writes the project into files of the sketch such as `plc86.cpp`), **compiles** (with the compiler of the 86Duino IDE) and **uploads**; the progress is shown in "**Build output**" of the "**Output**" window at the bottom.
3. The output shows in turn "Build: … files written to …", "== Compiled …", "== Uploading to COMx ..." with the progress, and finally "== Uploaded". The controller restarts automatically and starts running the PLC program.

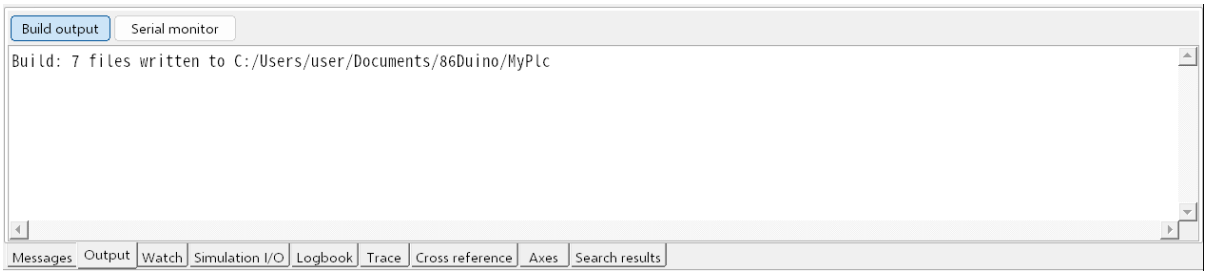


Figure 5-24 "Build output" in the Output window: the build, compile and upload progress is all shown here

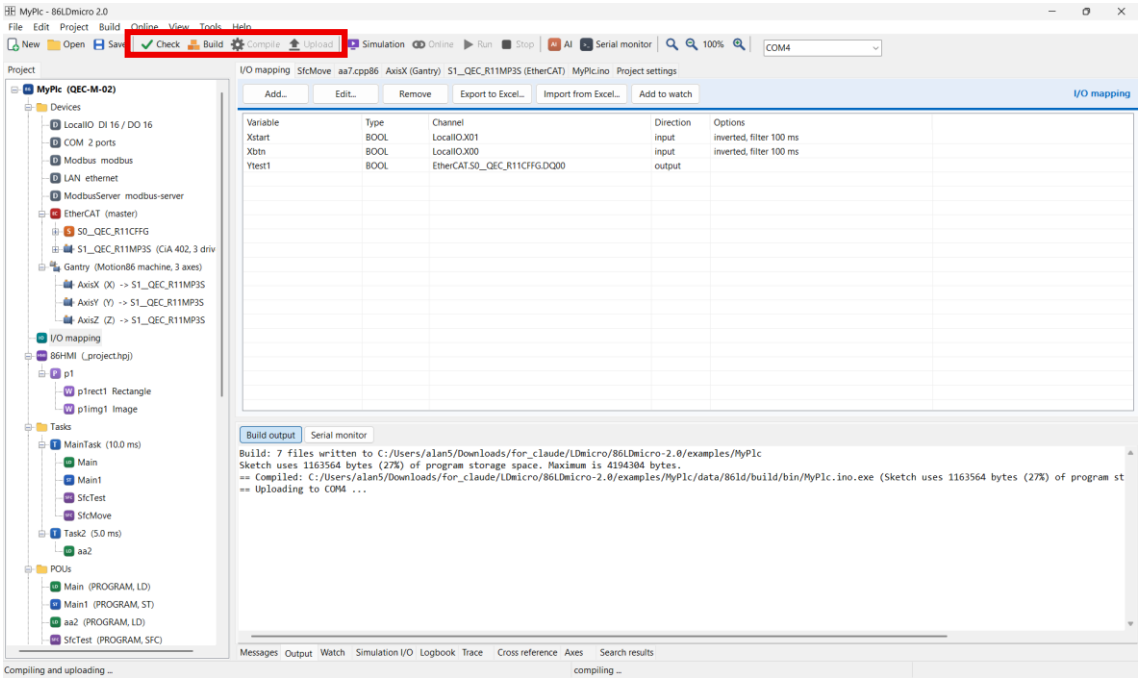


Figure 5-25 Uploading

After the build, an 86LDmicro opened from the IDE tells the IDE to reload the sketch: the IDE shows tabs such as `plc86.cpp` and `plc86.h`. From then on you can also compile and upload directly in the IDE.

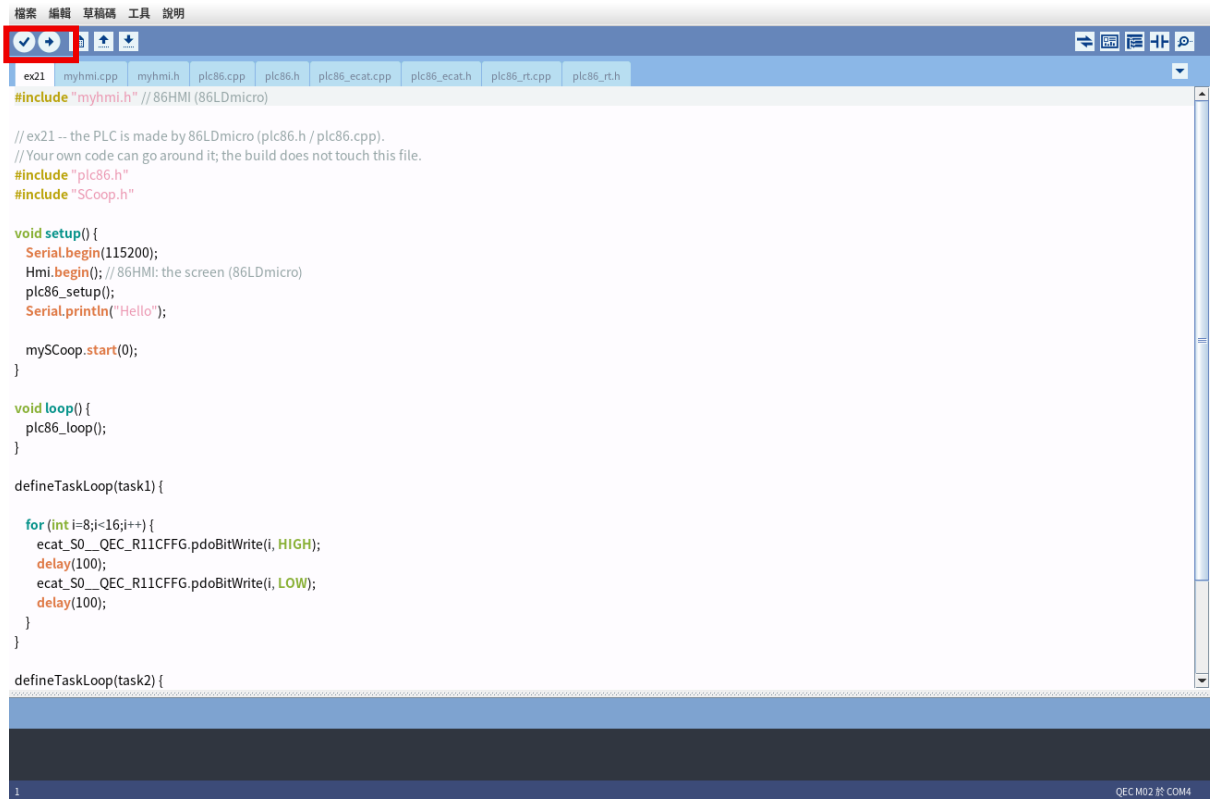


Figure 5-26 The 86Duino IDE after the build: the sketch has new tabs such as plc86.cpp and plc86.h

*__Note:__ Unsaved changes to the `.ino` in the IDE are overwritten when the sketch is reloaded after the build; save them in the IDE first.
If the upload fails with "Could not open COMx": close any other program that uses the same COM port (for example the Serial Monitor of the IDE), then press F8 again. For other messages and what to do, see section 10.1.

5.12 Step 12: Online view (Ctrl+F5)

1. After the upload, 86LDmicro goes online automatically ("Online again after an upload" in "Tools > Options", on by default); or click **Online** on the toolbar (Ctrl+F5, "Online > Login to the controller").
2. The status bar shows "ONLINE COMx: RUN" and the scan time, and the ladder diagram shows the live power flow and values on the controller.
3. Press the physical start button (X00): `Rrun` becomes TRUE, and the lamp on Y00 turns on 2 seconds later; press the stop button (X01) to turn it off.
4. To finish, choose "**Online > Logout**".

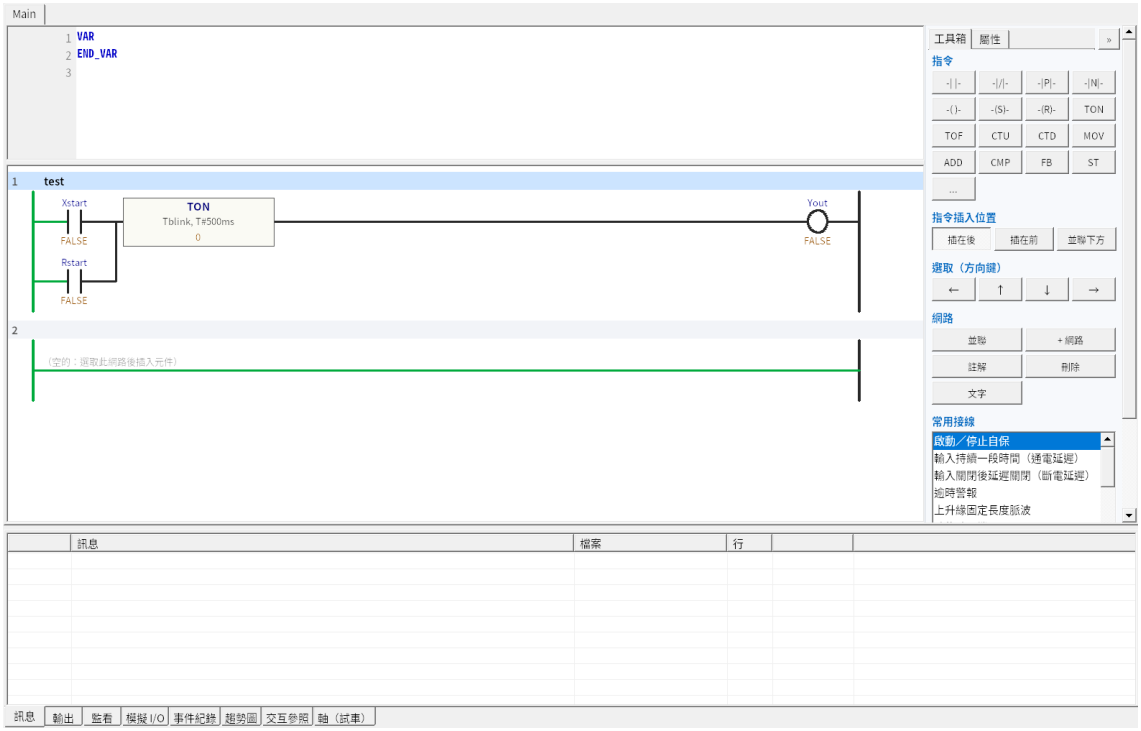


Figure 5-27 Online view: the ladder diagram shows the live state of the controller

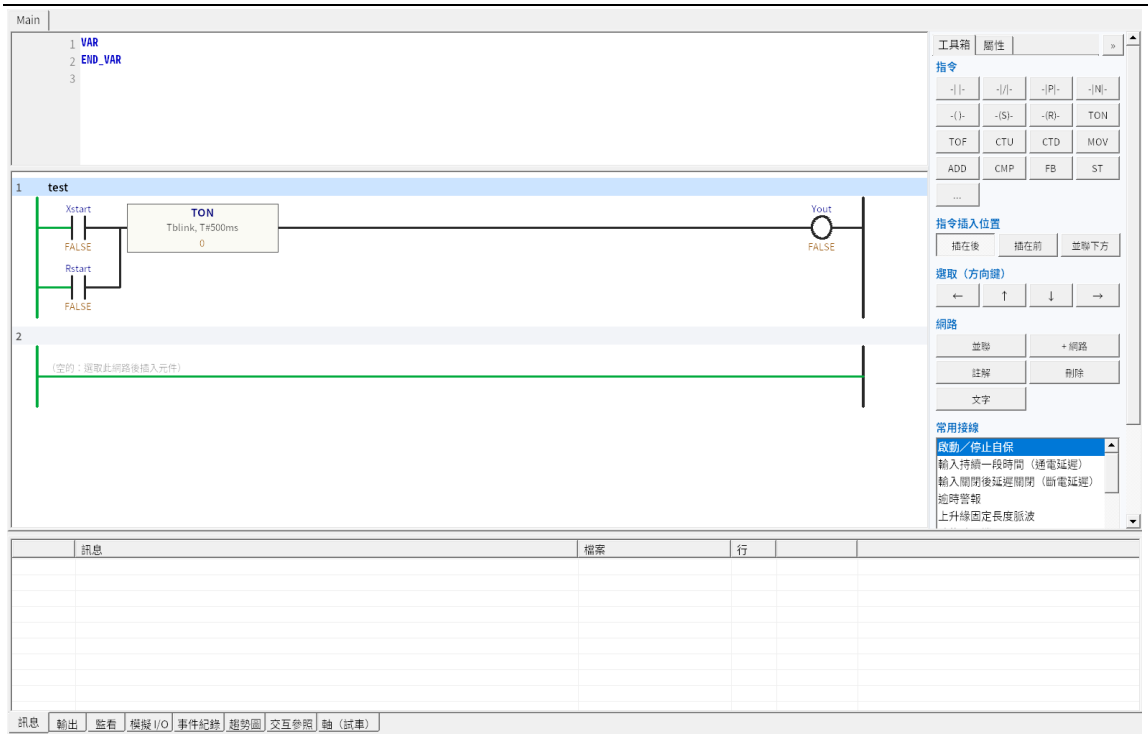


Figure 5-28 The status bar when online: ONLINE COMx: RUN and the scan time

Add variables in "Watch"; double-click a BOOL to toggle it, and right-click to force it (section 9.3). For the full description of the online view, RUN / STOP, the Logbook and more, see section 10.4.

> ***Tip:** Done! Chapter 6 gives more common examples with the same workflow (TOF, counting, blinking, ST, SFC, C/C++); Chapter 7 lists all the ladder instructions.

Chapter 6 Examples

The previous chapter went through the whole workflow once with "start / stop latch + delayed lamp". This chapter gives 7 more common examples, each in the same format: **scenario → variable table → program (.lad / ST / SFC / C++ text) → detailed steps table → expected result → screenshots**.

Each example is a separate sketch (its name is given in each section), and the workflow is the same as in Chapter 5: create the project → declare the variables → write the program → I/O mapping → check and simulate → upload. You can paste the program text directly into the text mode of the ladder diagram (Ctrl+T), or insert the elements one by one from the Toolbox following the steps table.

> ***Important:** The "I/O mapping" column of the variable tables gives suggested values; adjust them to your actual wiring. Every `X…` / `Y…` variable must be mapped to a physical channel in "I/O mapping" (Step 9 of Chapter 5), otherwise the I/O does not work after the upload.

6.0 General principles (read this first)

The four principles below apply to every example in this chapter, and they are also the concepts that need the most adjustment when you move from LDmicro 1.1. Once you understand them, you will hardly ever get stuck when compiling or running on the machine.

6.0.1 Principle ①: Every variable must be declared first; the declaration sets its type

LDmicro 1.1 creates variables automatically from the name prefix (X, Y, R, T, C) and the Source in the element window; 86LDmicro 2.0 follows IEC 61131-3: **every variable must be declared first, with a type**. The name prefix is only a naming convention and does not decide the kind of variable.

Where it is declared	How to write it	Who can use it	Suited for
The global variable list GVL ("Global variables" in the project tree)	`VAR_GLOBAL ... END_VAR`	All POUs, I/O mapping, 86HMI, C/C++ (`GVL(name)`)	Inputs, outputs, and flags and values shared between programs
The declaration area above a POU	`VAR ... END_VAR`	Only this POU (read from elsewhere as `program.variable`)	Local variables such as function block instances (`tLamp : TON;`), 1.x timers and step numbers
The retain area of the GVL	`VAR_GLOBAL RETAIN ... END_VAR`	Same as the GVL	Values that must be kept after a power loss (for example accumulated counts)

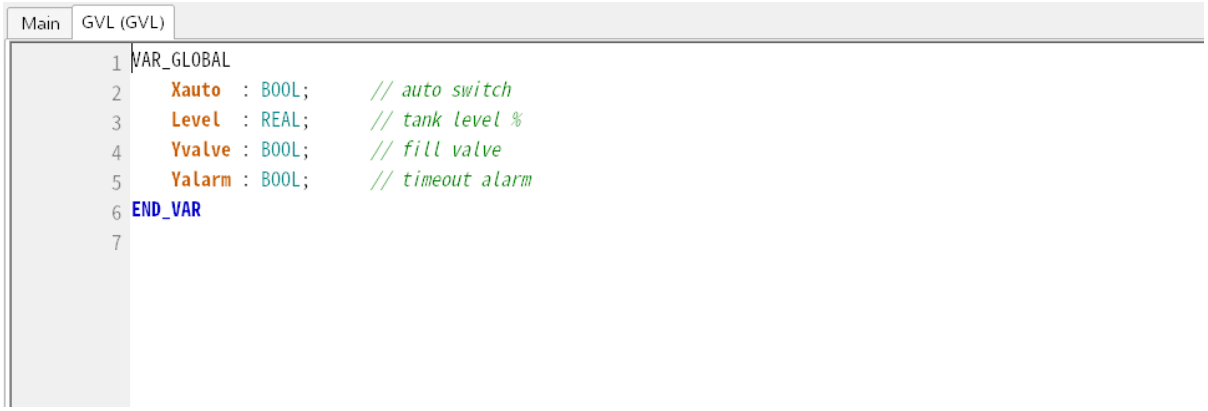


Figure 6-1 The GVL editor: one variable per line, the type after the colon, a comment after //

When the ladder diagram uses a name that is not declared (for example the typo `Xstrat`), the element is shown with a **red background**; after you click **Check** (Shift+F7), the "Messages" window lists "unknown variable Xstrat: declare it in VAR ... END_VAR, or in a global variable list (GVL)"; double-click the row to jump to the element.

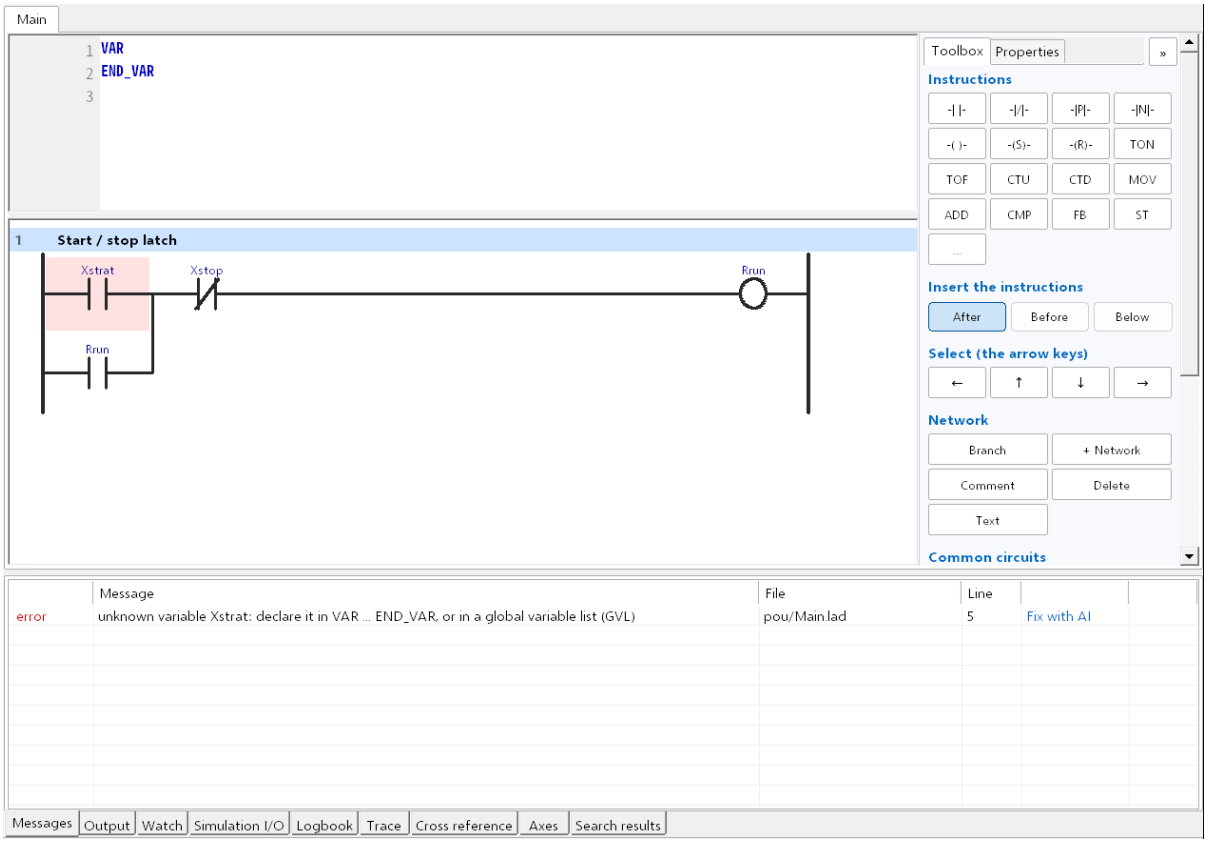


Figure 6-2 An undeclared variable: the element has a red background, and the build output shows "Build: the project has errors"

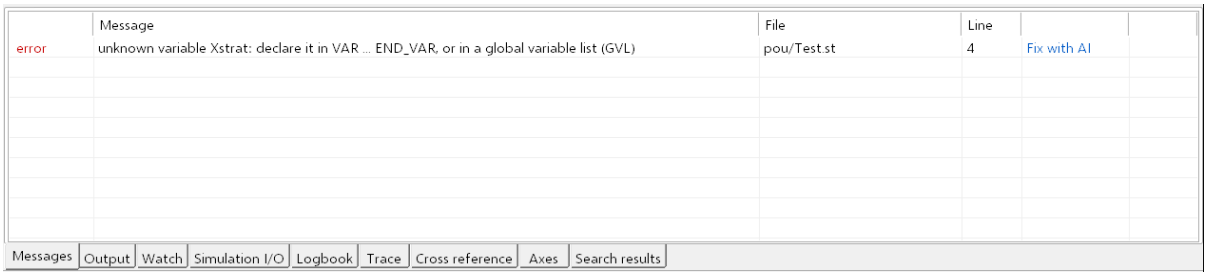


Figure 6-3 The "Messages" window: one error per row; double-click to jump to the element

6.0.2 Principle ②: I/O connects to physical points only through "I/O mapping"

A variable does not become a physical input just because its name is `X00` or `Xstart`. Only a variable given a channel (for example `LocalIO.X00`) in "I/O mapping" reads the physical input at the start of every scan and writes the physical output at the end of the scan.

Kind of variable	I/O mapping needed?	Description
Physical input (button, sensor)	Yes	Map to `LocalIO.X00` to `X17` (octal) or to an input of an EtherCAT module
Physical output (lamp, solenoid valve, motor)	Yes	Map to `LocalIO.Y00` to `Y17` or to an output of an EtherCAT module
Analog value (level, temperature)	Yes (analog channel)	Convert the raw value to engineering units with "Scaling" in the mapping window (section 4.6)
Internal flags and values (`R…`, count values)	No	Only in memory
Function block instances, 1.x timer / counter variables	No	Declaring them in the VAR of the POU is enough

When a contact in the ladder diagram reads a BOOL global variable that has no I/O mapping and is not written by any program or by the 86HMI, the build gives the warning "... is only read here: it has no I/O mapping and nothing writes it -- it is always FALSE", to remind you of the missing mapping.

*__Tip:__ A quick way to verify: after finishing the program, first click **Check** (Shift+F7) → **Simulate** (F5), double-click the input channels in "Simulation I/O", and see whether the output channels and the power flow of the ladder diagram behave as expected. Uploading only after the simulation works saves a lot of "flash, test, modify again" time.

6.0.3 Principle ③: Only one place writes a variable; mind the order of scans and networks

The PLC treats "read the inputs → run the programs in order → write the outputs" as one scan and repeats it at the interval of the task (every 10 ms in a new project). The networks of a POU run **from top to bottom**, and the programs in a task run in the order set in the task (section 4.3).

- Do not drive the same variable with two `OUT` coils: the later network overwrites the result of the earlier one. For example, with network 1 `NO(Xstart) OUT(Ymotor)` and network 2 `NO(Xstop) OUT(Ymotor)`, even when `Xstart` is TRUE, `Ymotor` is still decided by network 2 (FALSE). When you need several conditions, put them in parallel or in series before **the same** coil.
- Paired `SET` / `RST` are the exception: the one in the later network wins (for example, for stop priority, put `RST` after `SET`).
- To find which programs write a variable, right-click the variable and choose "**Where is <variable> used?**", or use "Cross reference" (Ctrl+Shift+F, section 9.5).
- Ladder diagrams, ST and C/C++ can share GVL variables; the same "one writer" rule applies, and all other places only read them (example 6.7).

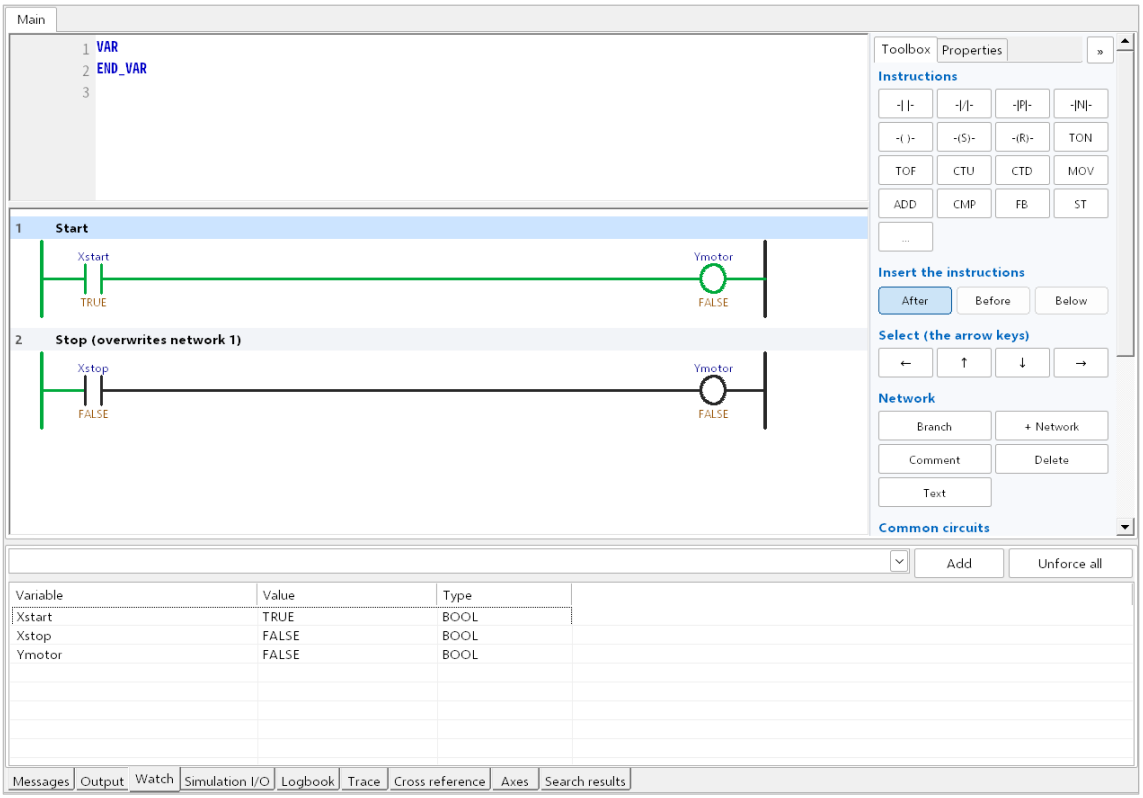


Figure 6-4 Two OUTs drive the same variable: the later network overwrites the result of the earlier one

6.0.4 Principle ④: Timers and counters: IEC function blocks or 1.x boxes

86LDmicro 2.0 has two kinds of timers and counters, and you can mix them:

	IEC function blocks (recommended)	**LDmicro 1.x compatible boxes**
Kinds	TON, TOF, TP, CTU, CTD, CTUD, R_TRIG, F_TRIG, SR, RS	TON, TOF, RTO, RTL, TCY, THI, TLO, CTU, CTD, CTC, CTR...
How to insert	**FB** (key K): `CALL(instance, IN := @, PT := T#2s, Q => @)`	**TON** (O), **TOF** (F), **CTU** (U), **CTD** (D) in the Toolbox; the others under "..."
Declaration	Instance declared in VAR: `tLamp : TON;`	Timer / counter variable declared in VAR: `t1 : DINT;`, `c1 : INT;`
Time	Real time, independent of the task interval	Counted in PLC cycles (delay / interval, rounded), as in 1.1
Reading the result	Members can be read anywhere: `NO(tLamp.Q)`, `CMP(cBox.CV >= 10)`, in ST `tLamp.ET`	Read the value of the count variable (for example `GEQ(c1, 10)`)

LDmicro 1.1 cannot use a timer or a counter directly as a contact; it must first drive an `R_…` internal relay as a "bridge". In 2.0, the outputs of IEC function blocks (`.Q`, `.CV`, `.ET`) are variables and can be used directly in contacts or comparisons in any network.

6.1 Start / stop latch

The most typical PLC circuit, equivalent to a self-holding relay: two buttons control one output; after start is pressed, the output stays ON even when the button is released, until stop is pressed.

Scenario: Control a motor (or a lamp) with two buttons, start and stop; one press of start keeps it running. Sketch name: `Ex61_Latch`.

Variable	Type	I/O mapping	Purpose
`Xstart`	BOOL	`LocalIO.X00`	Start button (normally open)
`Xstop`	BOOL	`LocalIO.X01`	Stop button (wired normally open)
`Ymotor`	BOOL	`LocalIO.Y00`	Motor / run lamp

Program (Main, .lad text)

```
PROGRAM Main
VAR
END_VAR
NETWORK 'Start / stop latch'
    // Press Xstart to start and latch, press Xstop to stop
    [ NO(Xstart) | NO(Ymotor) ] NC(Xstop) OUT(Ymotor)
END_PROGRAM
```

Detailed steps

#	Action
1	Create the sketch `Ex61_Latch` and a new project as in Steps 1 to 3 of Chapter 5.
2	Double-click "Global variables > GVL", declare `Xstart`, `Xstop` and `Ymotor` (BOOL), and press Ctrl+S to save.
3	Open `Main` and click network 1 to select it; press **C** → variable `Xstart` → OK; press **N** → `Xstop`; press **L** → `Ymotor`.
4	Press ← twice to select `Xstart`, press **B** (normally open contact in parallel below) → variable `Ymotor` → OK.
5	Double-click the title bar of network 1, enter the title `Start / stop latch` and the comment `Press Xstart to start and latch, press Xstop to stop`.
6	In "I/O mapping", add: `Xstart` → `LocalIO.X00`, `Xstop` → `LocalIO.X01`, `Ymotor` → `LocalIO.Y00`.
7	Shift+F7 to check; F5 to simulate: in "Simulation I/O", double-click `LocalIO.X00` twice (ON, OFF), then double-click `LocalIO.X01` twice, and confirm the latch and the stop.
8	F8 to upload.

Expected result: After one press of X00, Y00 = ON and stays ON; after one press of X01, Y00 = OFF. When both buttons are held down, Y00 = OFF (stop has priority).

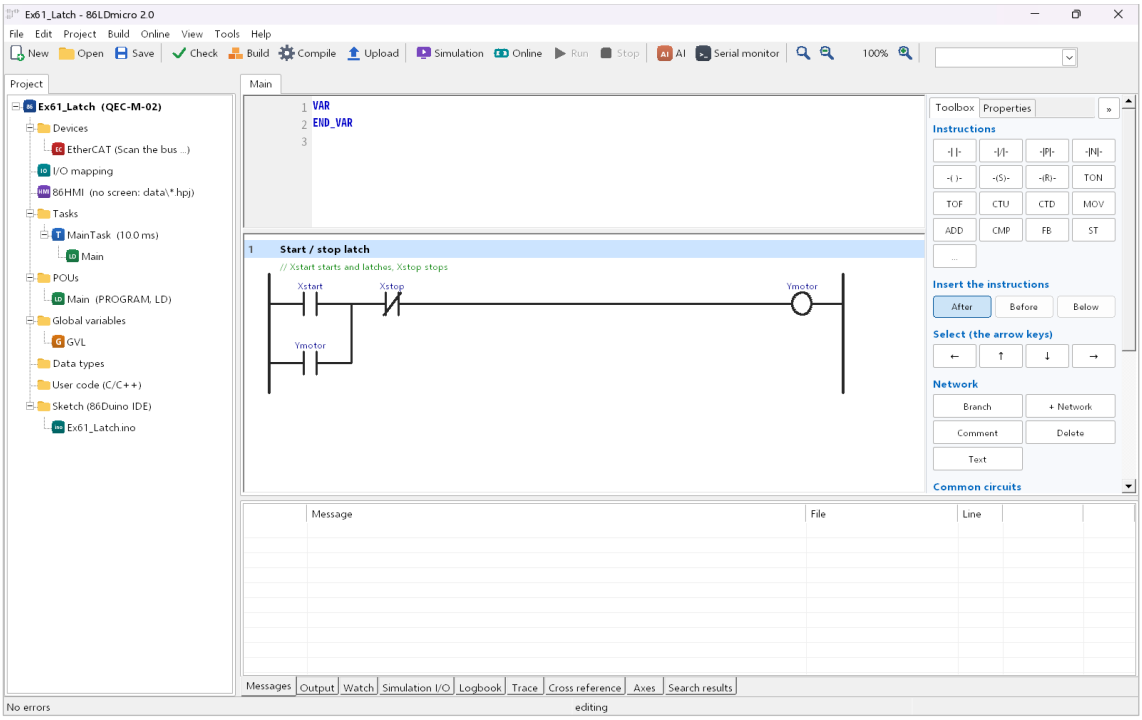


Figure 6-5 The ladder diagram of 6.1: Xstart in parallel with the latch contact Ymotor, in series with Xstop, drives Ymotor

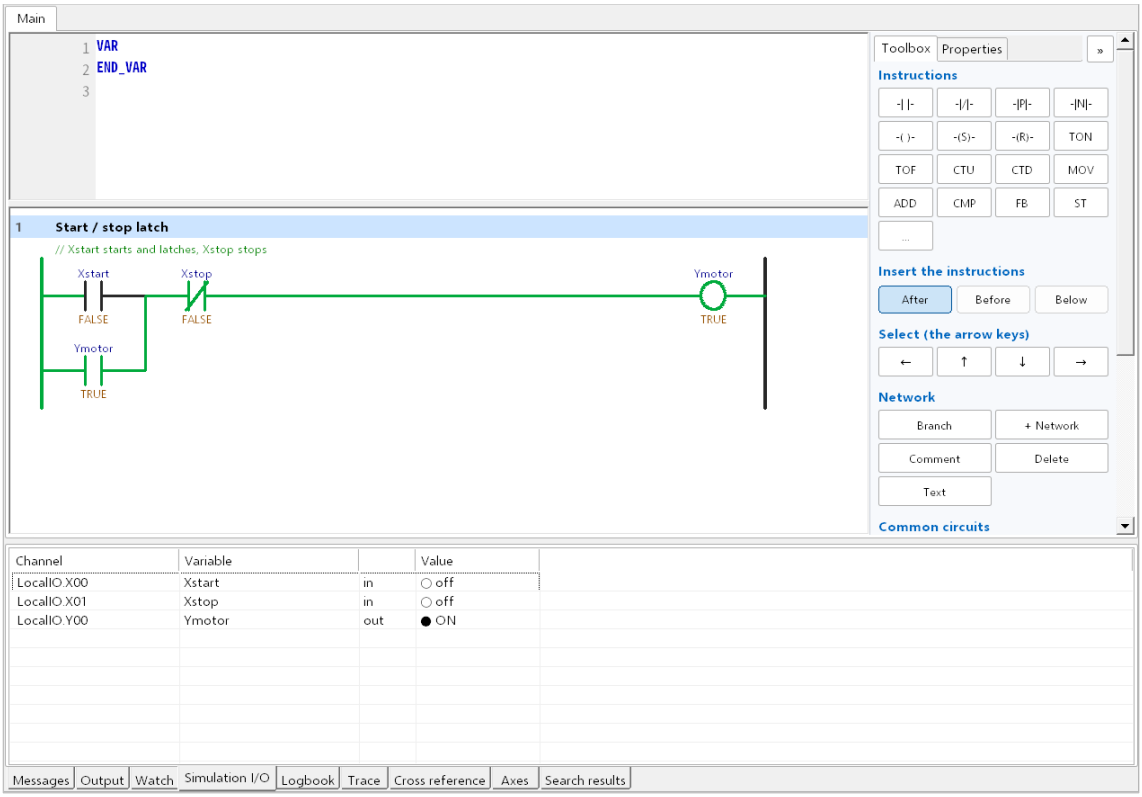


Figure 6-6 Simulation of 6.1: Ymotor is still TRUE after the start button is released (latch)

Another way: Common circuits. In "Common circuits" in the side panel, choose **Start / stop latch**, click **Insert**, and fill in the names of the start button, the stop button and the output: the same network is inserted, and the new variables are declared in the GVL automatically.

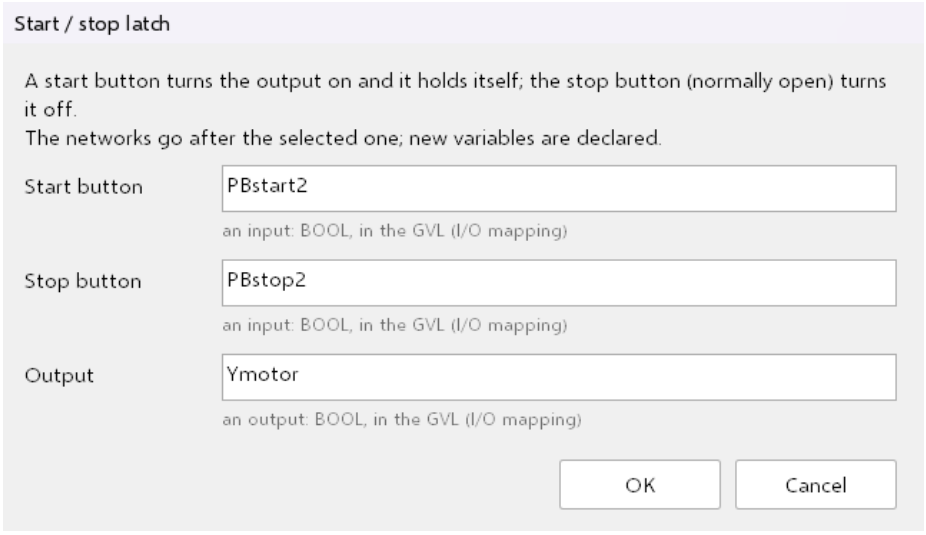


Figure 6-7 The common circuit "Start / stop latch": fill in the names of the buttons and the output

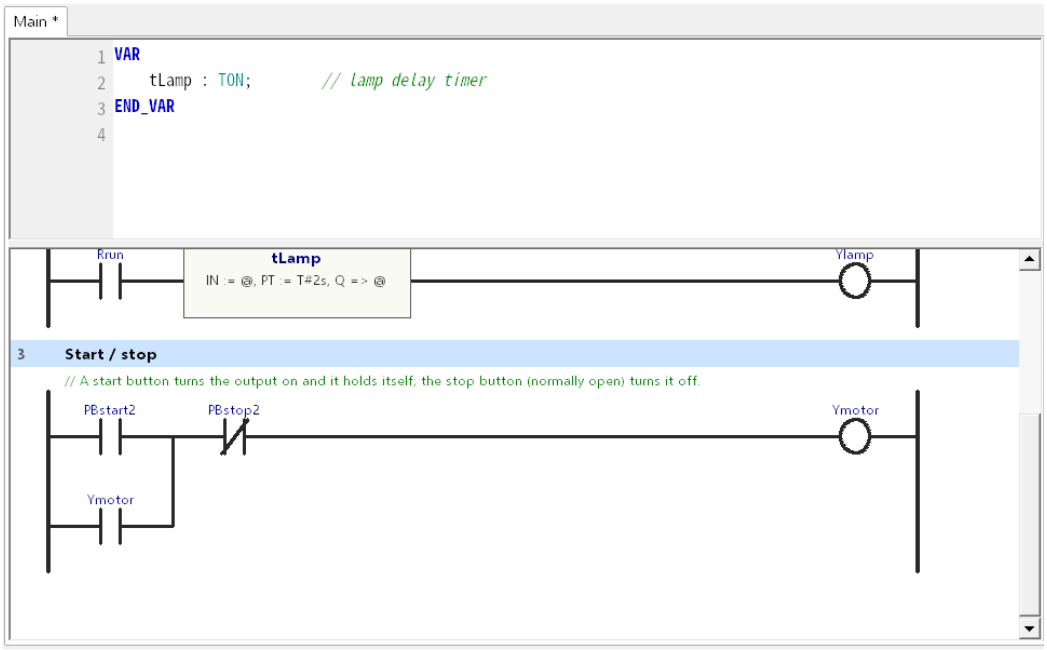


Figure 6-8 After inserting: the latch network made by the template; the new variables are already added to the GVL

Variation: SET / RST. The same function can also be written as two networks with set and reset coils; `RST` is written later, so stop has priority when both buttons are pressed at the same time:

```
NETWORK 'Start: set'
    NO(Xstart) SET(Ymotor)
NETWORK 'Stop: reset (written later, so stop has priority)'
    NO(Xstop) RST(Ymotor)
```

6.2 Delayed light off (TOF)

When the switch is turned on, the light turns on at once; after the switch is turned off, the light stays on for 5 more seconds before it goes off. Common for corridor lights, restroom lights and meeting room lighting.

Scenario: Switch ON → light ON at once; switch OFF → light OFF after a 5-second delay. Sketch name: `Ex62_Tof`.

Variable	Type	I/O mapping	Purpose
`Xsw`	BOOL	`LocalIO.X00`	Light switch
`Ylight`	BOOL	`LocalIO.Y00`	Light
`tOff`	TOF (IEC function block, in the VAR of Main)	—	5-second off delay

Program (Main, .lad text)

```
PROGRAM Main
VAR
    tOff : TOF;         // 5-second off delay
END_VAR
NETWORK 'Delayed light off'
    // Switch ON: the light turns on at once; after switch OFF it stays on for 5 more seconds
    NO(Xsw) CALL(tOff, IN := @, PT := T#5s, Q => @) OUT(Ylight)
END_PROGRAM
```

Detailed steps

#	Action
1	Create the sketch `Ex62_Tof` and a new project; declare `Xsw` and `Ylight` (BOOL) in the GVL.
2	Open `Main`; in the declaration area at the top, between `VAR` and `END_VAR`, enter `tOff : TOF;`.
3	Click network 1 to select it, press **C** → variable `Xsw`.
4	Press **K** (FB) → in "Element" enter `CALL(tOff, IN := @, PT := T#5s, Q => @)` → OK.
5	Press **L** → variable `Ylight`.
6	"I/O mapping": `Xsw` → `LocalIO.X00`, `Ylight` → `LocalIO.Y00` (`tOff` needs no mapping).
7	F5 to simulate: double-click `LocalIO.X00` to set it ON, and `Ylight` is TRUE at once; double-click it again to set it OFF, add `Main.tOff.ET` in "Watch" to see the elapsed time; `Ylight` becomes FALSE after 5 seconds.
8	F8 to upload.

Expected result: When `Xsw` = 1, `Ylight` = 1 at once; after `Xsw` goes from 1 to 0, `Ylight` stays 1 for 5 seconds and then becomes 0 (`tOff.ET` counts from 0 to `T#5s`). If the switch is turned on again within the 5 seconds, the timing restarts and the light stays on.

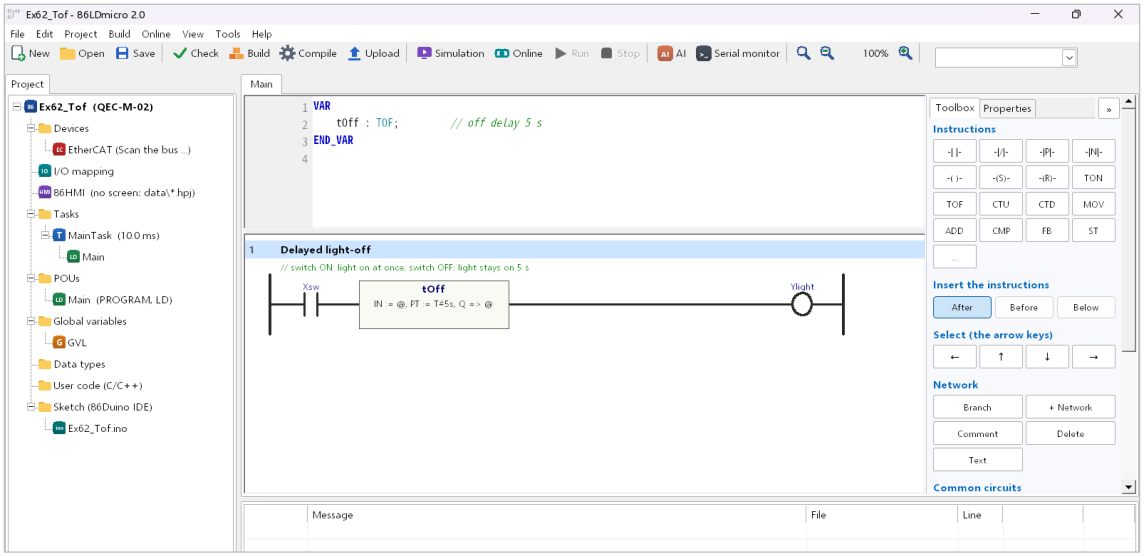


Figure 6-9 The ladder diagram of 6.2: Xsw → TOF function block tOff (PT := T#5s) → Ylight

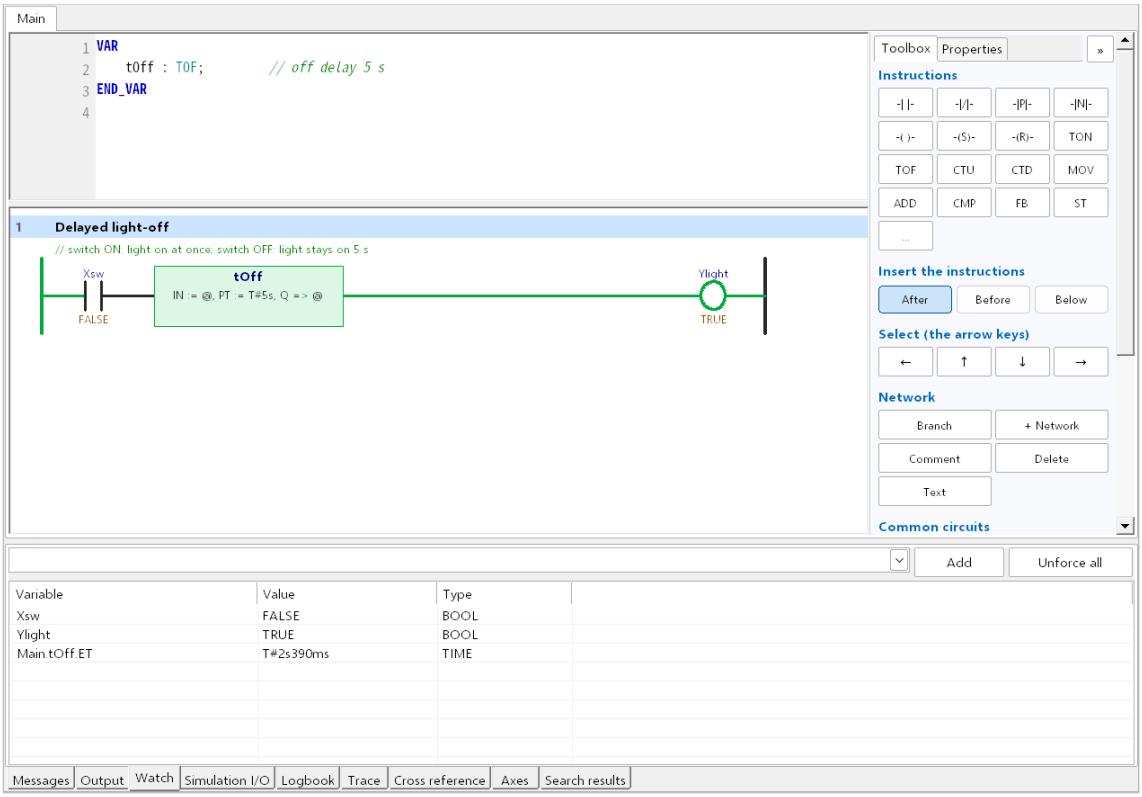


Figure 6-10 Simulation of 6.2: the switch is already OFF, the TOF is timing, Ylight is still TRUE

*Tip: Variation 1: on for N seconds per press (an auto-off button): replace `tOff : TOF` with `tOn : TP` and write `NO(Xsw) CALL(tOn, IN := @, PT := T#5s, Q => @) OUT(Ylight)`. On a rising edge of its input, TP outputs a pulse of a fixed 5 seconds, however long the button is pressed.

*Variation 2: a 1.x box: click **TOF** (F) in the Toolbox, write `NO(Xsw) TOF(t1, T#5s) OUT(Ylight)`, and declare `t1 : DINT;` in VAR (it times in PLC cycles, section 7.4).

6.3 Packing counter (CTU + compare)

Each product the sensor detects adds 1; at 10 products, the boxing mechanism is driven for 1 second, and the count resets automatically and goes on. This example shows how to read the count value with a compare instruction and then drive the following networks with the internal flag `Rdone`.

Scenario: Counting on a production line, 10 per box, with automatic boxing and automatic reset in a cycle. Sketch name: `Ex63_Counter`.

Variable	Type	I/O mapping	Purpose
`Xsensor`	BOOL	`LocalIO.X00`	Product sensor
`Ybox`	BOOL	`LocalIO.Y00`	Boxing mechanism
`Count`	INT	—	Current count (can be shown in Watch and on the 86HMI)
`Rdone`	BOOL	—	Flag: 10 reached
`cBox`	CTU (in the VAR of Main)	—	Product counter
`tBox`	TP (in the VAR of Main)	—	1-second boxing pulse

Program (Main, .lad text)

```
PROGRAM Main
VAR
    cBox : CTU;         // product counter
    tBox : TP;          // 1-second boxing pulse
END_VAR
NETWORK 'Count products'
    // +1 on every rising edge of the sensor; reset when Rdone is TRUE
    NO(Xsensor) CALL(cBox, CU := @, R := Rdone, PV := 10, CV => Count)
NETWORK 'Compare: 10 reached'
    CMP(Count >= 10) OUT(Rdone)
NETWORK 'Boxing for 1 s'
    NO(Rdone) CALL(tBox, IN := @, PT := T#1s, Q => @) OUT(Ybox)
END_PROGRAM
```

Detailed steps

#	Action
1	Create the sketch `Ex63_Counter` and a new project; declare `Xsensor`, `Ybox`, `Rdone` (BOOL) and `Count` (INT) in the GVL.
2	`Main`: in the declaration area, enter `cBox : CTU;` and `tBox : TP;`.
3	Network 1: press **C** → `Xsensor`; press **K** → in "Element" enter `CALL(cBox, CU := @, R := Rdone, PV := 10, CV => Count)`.
4	Ctrl+Enter to add network 2: press **E** (condition contact CMP) → enter `Count >= 10` in "Condition"; press **L** → `Rdone`.
5	Ctrl+Enter to add network 3: press **C** → `Rdone`; press **K** → `CALL(tBox, IN := @, PT := T#1s, Q => @)`; press **L** → `Ybox`.
6	"I/O mapping": `Xsensor` → `LocalIO.X00`, `Ybox` → `LocalIO.Y00` (`Count`, `Rdone`, `cBox` and `tBox` need no mapping).
7	F5 to simulate: in "Simulation I/O", double-click `LocalIO.X00` repeatedly (ON, OFF is one product), 10 times in all; add `Count` in "Watch" to see the count.
8	F8 to upload.

How it works

1. On every rising edge of `Xsensor`, `cBox` adds 1, and `Count` is updated with it.
2. When `Count` reaches 10, the condition `Count >= 10` of network 2 is true, and `Rdone` = TRUE.
3. Network 3: the rising edge of `Rdone` triggers `tBox` (TP), and `Ybox` = TRUE for a fixed 1 second.
4. In the next scan, `R := Rdone` in network 1 resets the count (`Count` = 0), and `Rdone` in network 2 returns to FALSE; `Ybox` is still held by the TP until the 1 second is over.
5. The cycle repeats, counting the next box.

Expected result: For every 10 pulses received, Ybox is energized for 1 second, and the count resets and starts again. Precondition: the next 10 products do not all arrive within the 1 second of boxing (TP accepts no new trigger during its pulse, so the Rdone of the next box would be ignored).

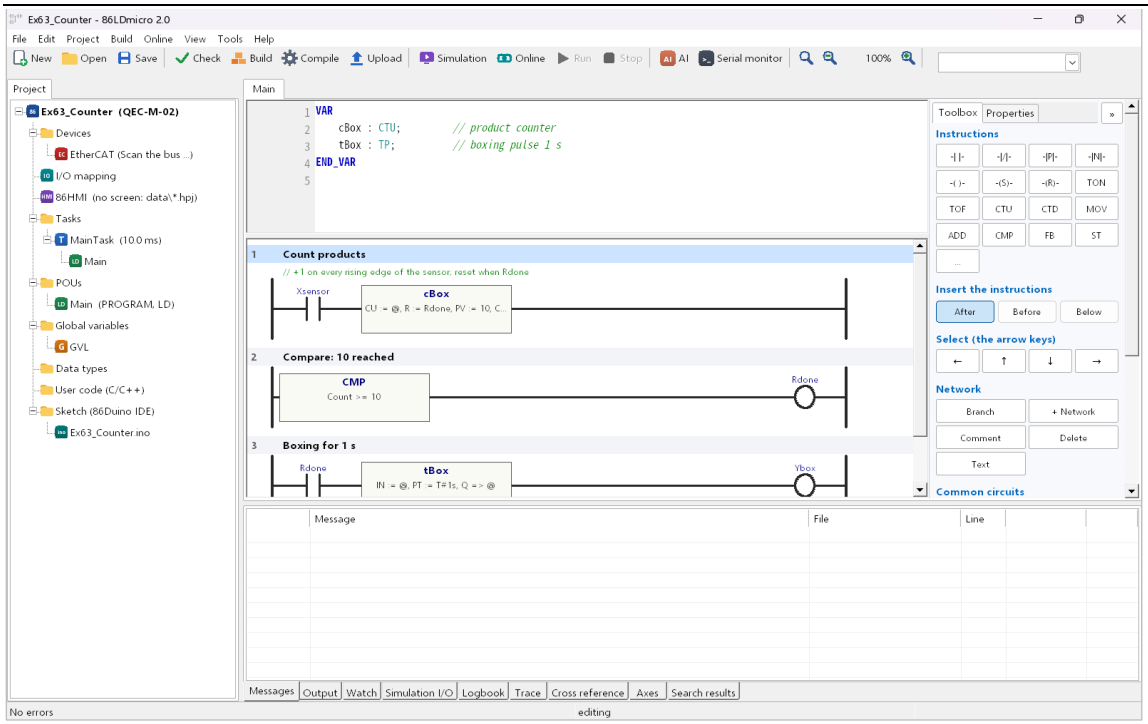


Figure 6-11 The ladder diagram of 6.3: three networks for CTU counting, the CMP comparison and the TP boxing pulse

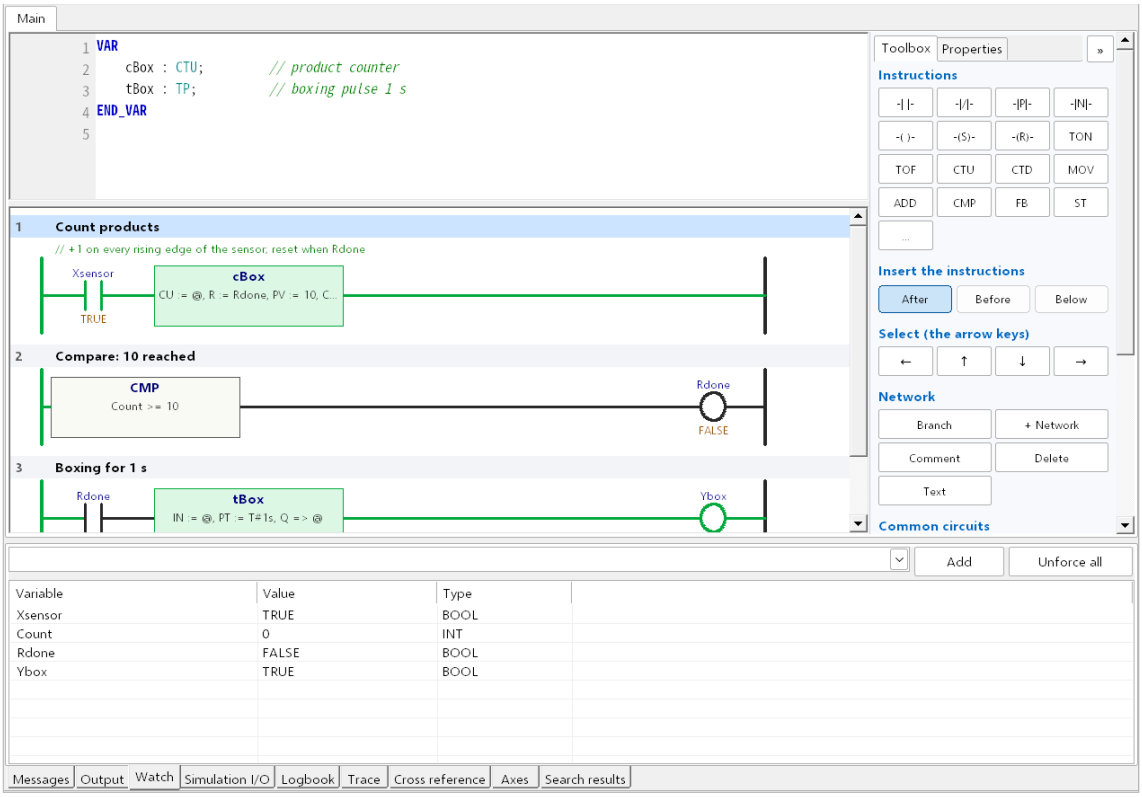


Figure 6-12 Simulation of 6.3: after the 10th product, Ybox is energized and Count is already reset

***Tip:** The comparison can also read the output of the function block directly: `CMP(cBox.CV >= 10)`, or the contact `NO(cBox.Q)` (TRUE when CV reaches PV).
If the sensor signal bounces, set "Filter (ms)" in the I/O mapping, or use the common circuit "Debounce an input".

***Note: When products come very fast, add a busy interlock (for example, stop the conveyor while Ybox is boxing), or switch to a design that accumulates the number of products waiting to be boxed, so that no box is missed.**

If a rising edge of the sensor comes in the very scan in which the count resets, that product is not counted (R of the CTU has priority); when every product must be counted, use the accumulating design above.

6.4 Running lights (alternate blinking)

Two lamps blink alternately at 500 ms: while one is on, the other is off, and they switch together every 500 ms. This example uses the LDmicro 1.x compatible cycle timer **TCY** to make a square wave, stores it in the internal bit `Rstate`, and then drives the two lamps from two networks, one with a normally open and one with a normally closed contact.

Scenario: When the switch is ON, the two lamps blink alternately in opposite phase; when the switch is OFF, both are off. Sketch name: `Ex64_Blink`.

Variable	Type	I/O mapping	Purpose
`Xrun`	BOOL	`LocalIO.X00`	Blink switch
`Rstate`	BOOL	—	Square wave state (shared between networks)
`Yled1`	BOOL	`LocalIO.Y00`	Same as `Rstate` (in phase)
`Yled2`	BOOL	`LocalIO.Y01`	Opposite of `Rstate` (inverted)
`tClock`	DINT (in the VAR of Main)	—	Timer variable of the TCY [1.x]

Program (Main, .lad text)

```
PROGRAM Main
VAR
    tClock : DINT;      // TCY timer [1.x]
END_VAR
NETWORK '500 ms square wave'
    NO(Xrun) TCY(tClock, T#500ms) OUT(Rstate)
NETWORK 'Lamp 1: in phase with the square wave'
    NO(Xrun) NO(Rstate) OUT(Yled1)
NETWORK 'Lamp 2: opposite to the square wave'
    NO(Xrun) NC(Rstate) OUT(Yled2)
END_PROGRAM
```

Detailed steps

#	Action
1	Create the sketch `Ex64_Blink` and a new project; declare `Xrun`, `Rstate`, `Yled1` and `Yled2` (BOOL) in the GVL.
2	`Main`: in the declaration area, enter `tClock : DINT;`.
3	Network 1: press **C** → `Xrun`; click **...** in "Instructions" in the side panel → "Boxes" → **TCY**, fill in `tClock` for "timer" and `T#500ms` for "delay" (leave "adjust" empty) → OK; press **L** → `Rstate`.
4	Ctrl+Enter to add network 2: **C** → `Xrun`; **C** → `Rstate`; **L** → `Yled1`.
5	Ctrl+Enter to add network 3: **C** → `Xrun`; **N** → `Rstate`; **L** → `Yled2`.
6	"I/O mapping": `Xrun` → `LocalIO.X00`, `Yled1` → `LocalIO.Y00`, `Yled2` → `LocalIO.Y01`.
7	F5 to simulate: double-click `LocalIO.X00` to set it ON, and watch `LocalIO.Y00` and `Y01` alternate ON / OFF every 500 ms.
8	F8 to upload.

Expected result: When `Xrun` = 1, `Yled1` and `Yled2` switch, in opposite states, about every 500 ms (a period of about 1 second); when `Xrun` = 0, both are 0.

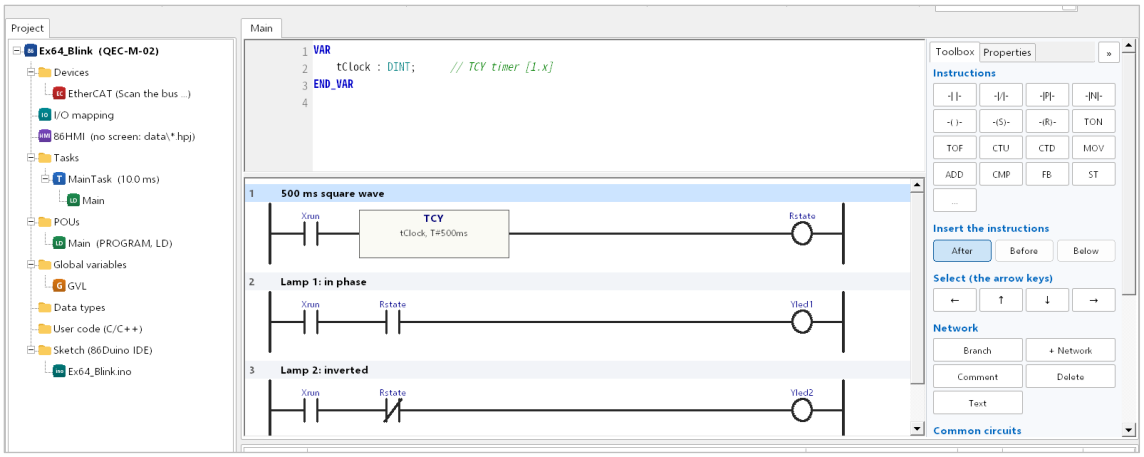


Figure 6-13 The ladder diagram of 6.4: TCY makes the square wave Rstate, and two networks drive the two lamps with NO / NC

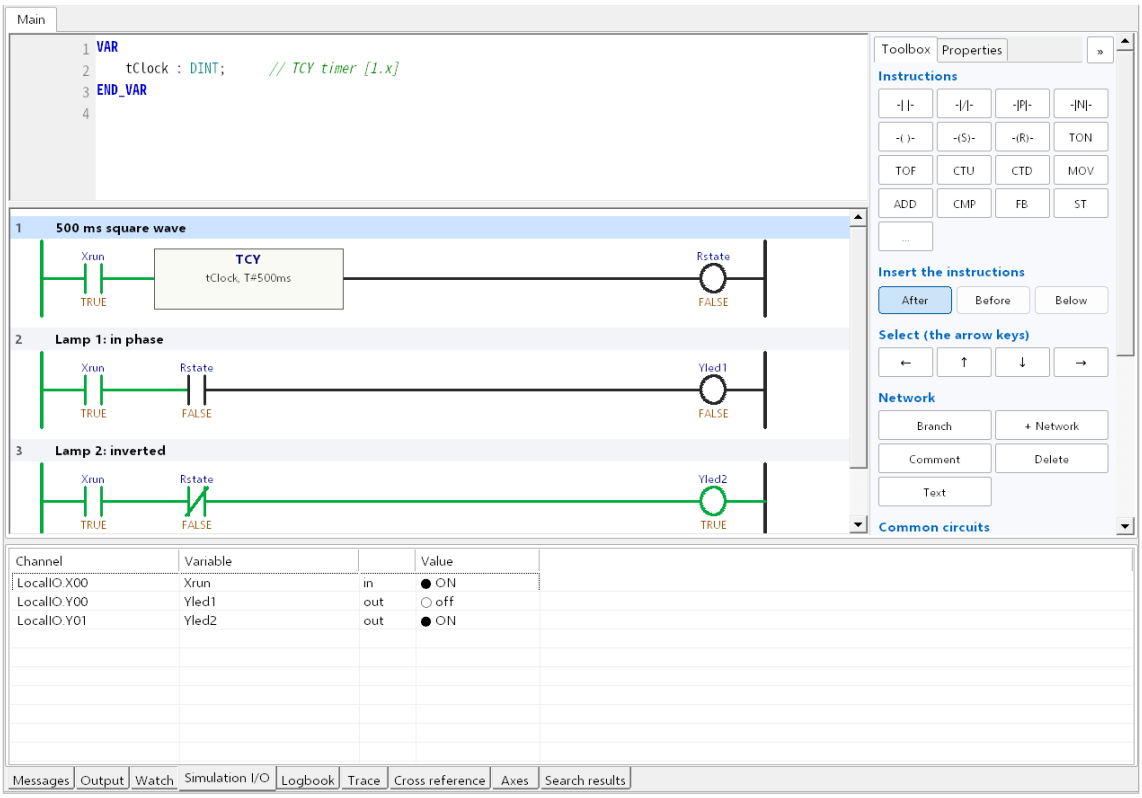


Figure 6-14 Simulation of 6.4: Yled1 and Yled2 turn on and off alternately

*__Tip:__ TCY and the other 1.x timers count in PLC cycles: with a task interval of 10 ms, `T#500ms` is 50 cycles. A delay shorter than the task interval cannot be timed (Check gives a warning).
You can also do without 1.x instructions: **Blink while enabled** in "Common circuits" in the side panel lets you set the on and off times separately; or build your own blink function block with TON in ST (section 8.3).

6.5 ST: Fill control

Write a fill control with steps in Structured Text (ST): when the auto switch is on and the level is below 20 %, open the valve to fill, and close it at 80 %; if 80 % is still not reached after more than 30 seconds of filling, treat it as a fault and output an alarm. ST suits logic with many conditions, calculations or steps.

Scenario: Automatic refilling of a water tank, with a timeout alarm; turning off the auto switch stops it and resets the alarm. Sketch name: `Ex65_Fill`.

Variable	Type	I/O mapping	Purpose
`Xauto`	BOOL	`LocalIO.X00`	Auto fill switch
`Level`	REAL	Analog input (depends on the hardware, section 4.6)	Level (%)
`Yvalve`	BOOL	`LocalIO.Y00`	Fill valve
`Yalarm`	BOOL	`LocalIO.Y01`	Fill timeout alarm
`tFill`	TON (in the VAR of Filling)	—	Fill timeout of 30 s
`step`	INT (in the VAR of Filling)	—	0 idle, 1 filling, 2 timeout

Program (Filling, ST)

```
PROGRAM Filling
VAR
    tFill : TON;        // fill timeout
    step  : INT;        // 0 idle, 1 filling, 2 timeout
END_VAR
CASE step OF
    0:  (* Idle: start filling when in auto and the level is below 20 % *)
        IF Xauto AND Level < 20.0 THEN
            step := 1;
        END_IF;
    1:  (* Filling: stop at 80 %; more than 30 s is a fault *)
        IF Level >= 80.0 OR NOT Xauto THEN
            step := 0;
        ELSIF tFill.Q THEN
            step := 2;
        END_IF;
    2:  (* Timeout: turning off the auto switch resets it *)
        IF NOT Xauto THEN
            step := 0;
        END_IF;
END_CASE;
tFill(IN := (step = 1), PT := T#30s);   // called in every scan
Yvalve := (step = 1);
Yalarm := (step = 2);
END_PROGRAM
```

Detailed steps

#	Action
1	Create the sketch `Ex65_Fill` and a new project; declare `Xauto`, `Yvalve`, `Yalarm` (BOOL) and `Level` (REAL) in the GVL.
2	"Project > Add a PROGRAM (ST)...", enter the name `Filling` → OK. The new PROGRAM is added automatically to the first task, MainTask, and the ST editor opens.
3	Enter the program above (typing the first few letters of a variable name lists the completions); press **Ctrl+T** to format it automatically, and Ctrl+S to save. You can keep the empty `Main`.
4	"I/O mapping": `Xauto` → `LocalIO.X00`, `Yvalve` → `LocalIO.Y00`, `Yalarm` → `LocalIO.Y01`. Leave `Level` unmapped for now so that you can write it directly in the simulation; on the machine, map it to the actual analog input channel and set "Scaling" (for example, raw values 0 to 4095 to 0 to 100).
5	Shift+F7 to check; F5 to simulate: in "Simulation I/O", double-click `LocalIO.X00` to set it ON; in "Watch", add `Level`, `Filling.step` and `Yvalve`; double-click `Level` and enter `10` → `Yvalve` = TRUE; enter `85` → `Yvalve` = FALSE.
6	Then change `Level` to `10` and wait 30 seconds: `Filling.step` = 2, `Yalarm` = TRUE; set `LocalIO.X00` to OFF, and the alarm resets.
7	F8 to upload.

Expected result: When the auto switch is ON and the level is < 20 %, the fill valve opens; it closes when the level reaches 80 %. If 80 % is not reached after 30 seconds of filling, the valve closes and the alarm is output until the auto switch is OFF.

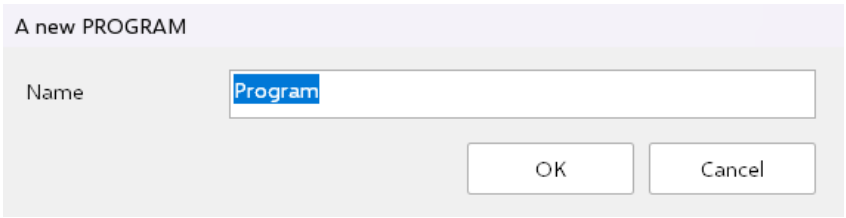


Figure 6-15 "Project > Add a PROGRAM (ST)...": enter the program name

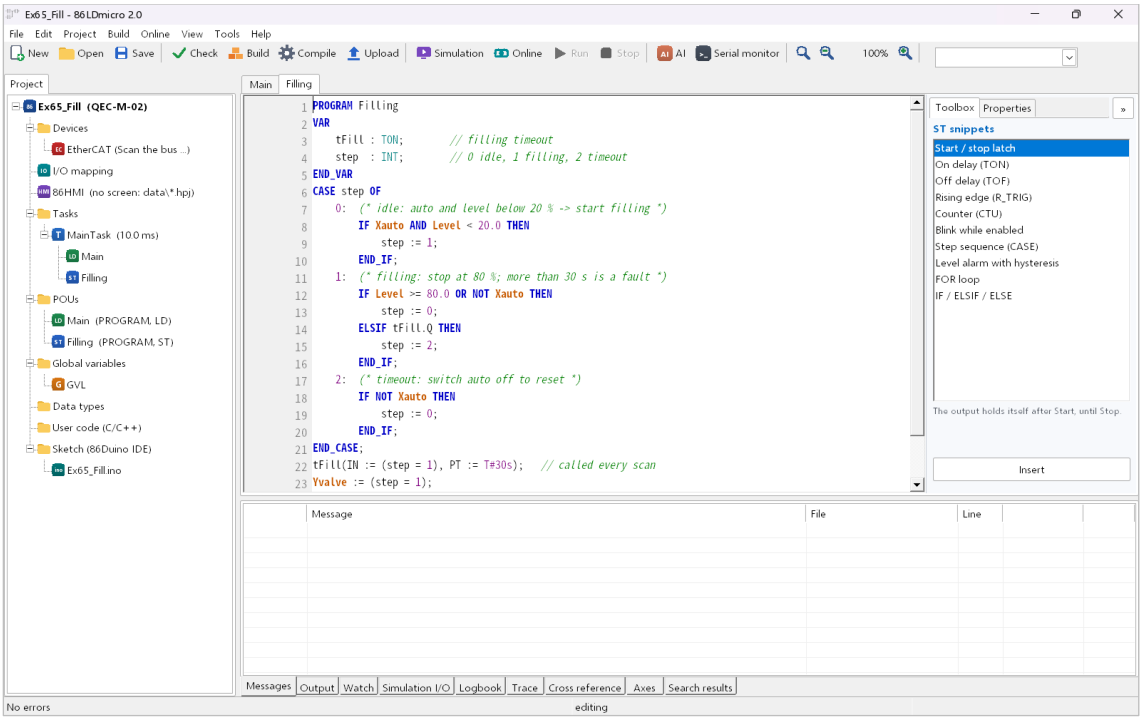


Figure 6-16 The ST program Filling of 6.5: CASE steps and the TON timeout

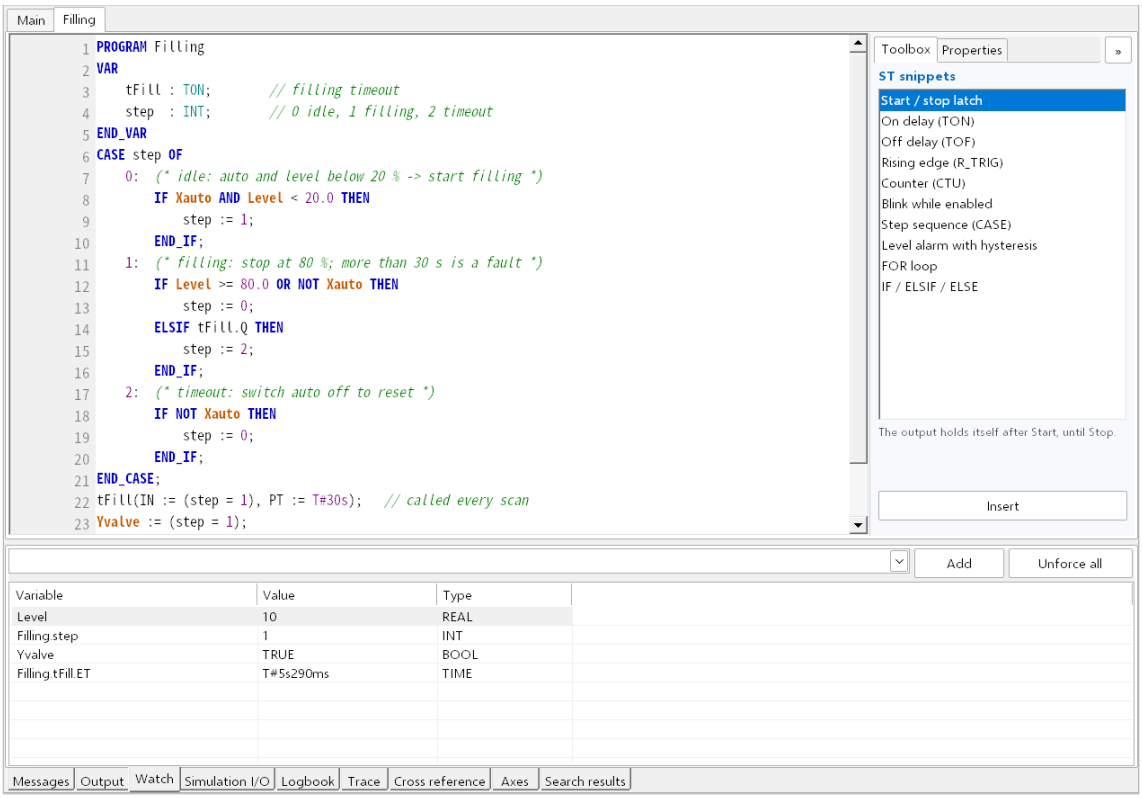


Figure 6-17 Simulation of 6.5: Level = 10, step = 1, the fill valve is open

***Tip:** A function block instance must be called in every scan for its timer to accumulate time. This example puts `tFill(...)` outside the CASE and calls it with `IN := (step = 1)`; when step 1 is left, IN becomes FALSE and the timer resets automatically. For the ST syntax, see section 8.2.

6.6 SFC: Sequence control

Describe a fixed "fill → mix → drain" sequence with a Sequential Function Chart (SFC). SFC shows the flow with **steps** and **transitions**; only the active steps perform their actions at a time, which suits the automatic sequences of machines.

Scenario: After start is pressed: the inlet valve opens until the full level is reached or 10 seconds pass; mix for 5 seconds; drain for 3 seconds; when done, return to idle and wait for the next start. Sketch name: `Ex66_Seq`.

Variable	Type	I/O mapping	Purpose
`Xstart`	BOOL	`LocalIO.X00`	Start button
`Xfull`	BOOL	`LocalIO.X01`	Full level switch
`Yvalve`	BOOL	`LocalIO.Y00`	Inlet valve
`Ymixer`	BOOL	`LocalIO.Y01`	Mixer motor
`Ydrain`	BOOL	`LocalIO.Y02`	Drain valve
`cycles`	INT (in the VAR of Seq)	—	Number of completed cycles

Program (Seq, .sfc text)

```
PROGRAM Seq
VAR
    cycles : INT;       // completed cycles
END_VAR
SFC
STEP Idle INITIAL
    Ydrain := FALSE;
END_STEP
TRANSITION Xstart
STEP Fill
    Yvalve := TRUE;
END_STEP
TRANSITION Xfull OR Fill.T >= T#10s
STEP Mix
    Yvalve := FALSE;
    Ymixer := TRUE;
END_STEP
TRANSITION Mix.T >= T#5s
STEP Drain
    Ymixer := FALSE;
    Ydrain := TRUE;
END_STEP
TRANSITION Drain.T >= T#3s
STEP Done
    cycles := cycles + 1;
END_STEP
TRANSITION TRUE
JUMP Idle
END_SFC
END_PROGRAM
```

Detailed steps

#	Action
1	Create the sketch `Ex66_Seq` and a new project; declare `Xstart`, `Xfull`, `Yvalve`, `Ymixer` and `Ydrain` (BOOL) in the GVL.
2	"Project > Add a PROGRAM (SFC, a sequence)...", name `Seq` → OK. The new chart already has `Init` → `TRUE` → `Step1` → `Step1.T >= T#1s` → jump to `Init`. Enter `cycles : INT;` in the declaration area at the top.

#	Action
3	Double-click `Init`: change the name to `Idle`, enter `Ydrain := FALSE;` in "Action" → OK (the JUMP to it is renamed along with it).
4	Double-click the transition after `Idle`: enter `Xstart` in "Condition (ST)".
5	Double-click `Step1`: name `Fill`, action `Yvalve := TRUE;`. Double-click the transition after it and change it to `Xfull OR Fill.T >= T#10s`.
6	Select this transition and click **+ Step** (Ins) in "Chart" in the side panel: a step and a `TRUE` transition are added after it. Double-click the new step: name `Mix`, actions `Yvalve := FALSE; Ymixer := TRUE;`; change the transition to `Mix.T >= T#5s`.
7	In the same way, select the last transition and click **+ Step**: step `Drain` (actions `Ymixer := FALSE; Ydrain := TRUE;`), transition `Drain.T >= T#3s`.
8	Click **+ Step** once more: step `Done` (action `cycles := cycles + 1;`), keep the transition `TRUE`; after it comes the existing "jump to Idle".
9	"I/O mapping": map X00, X01 and Y00 to Y02 as in the variable table.
10	F5 to simulate: double-click `LocalIO.X00` twice (ON, OFF) and watch the active step (green) move along in order, with its active time shown on the right; you can also double-click `LocalIO.X01` during Fill to simulate the full level.
11	F8 to upload.

Expected result: One press of X00: `Yvalve` ON (at most 10 seconds, ending early when X01 is ON) → `Ymixer` ON for 5 seconds → `Ydrain` ON for 3 seconds → back to `Idle`, `Ydrain` OFF, `cycles` + 1.

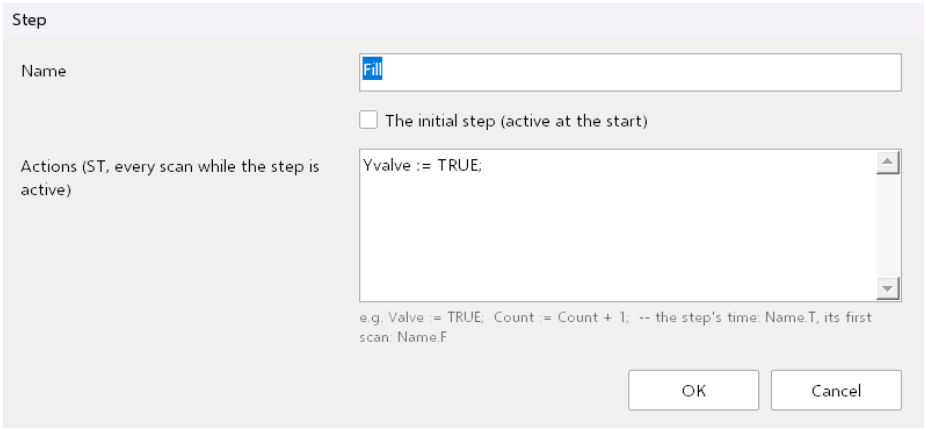


Figure 6-18 The step settings window: name, initial step, actions (ST)



Figure 6-19 The transition settings window: condition (ST) and comment

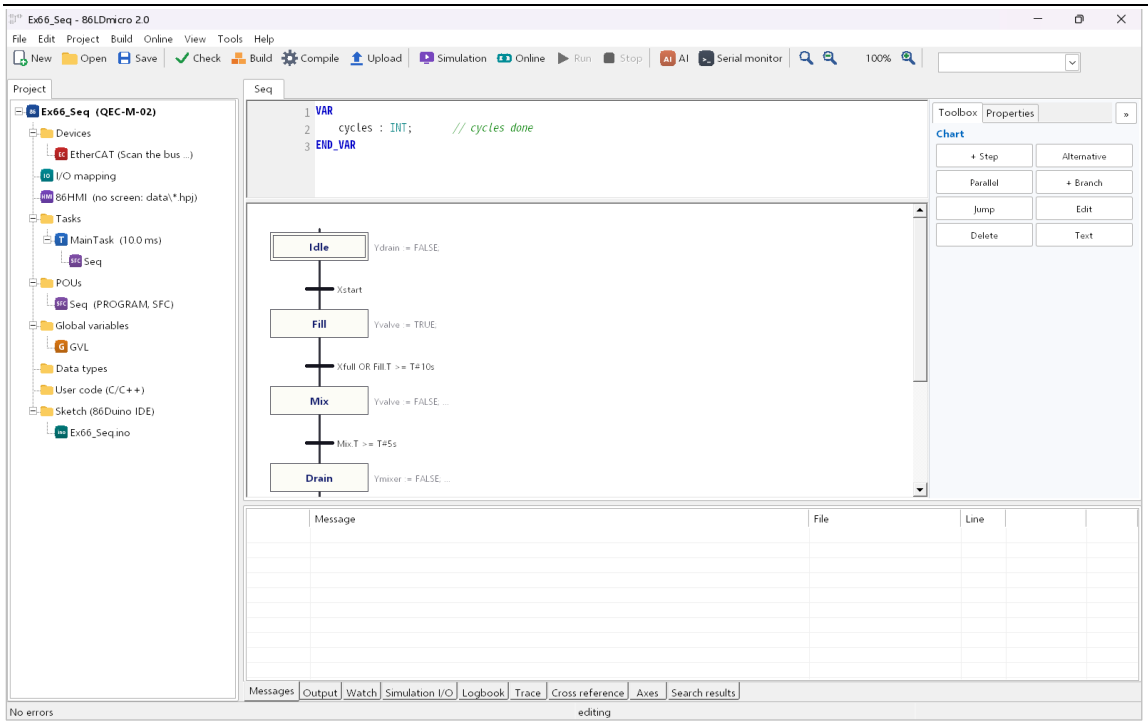


Figure 6-20 The SFC chart of 6.6: Idle → Fill → Mix → Drain → Done → jump to Idle

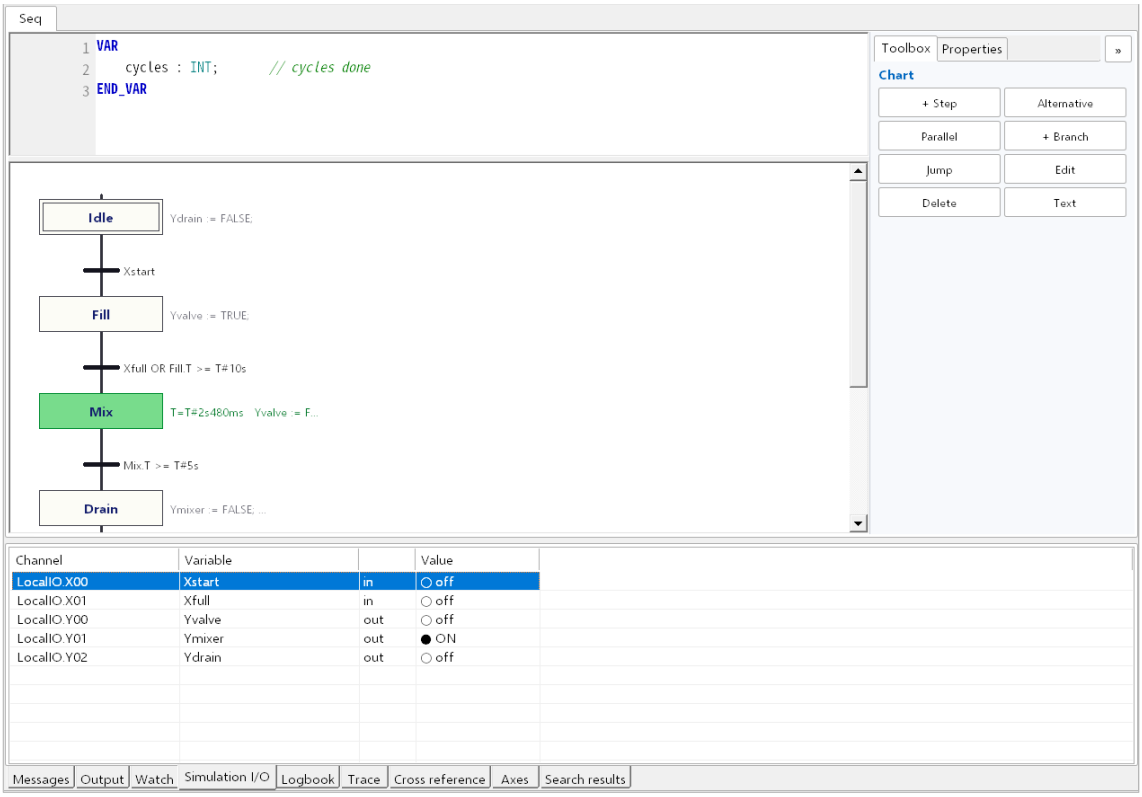


Figure 6-21 Simulation of 6.6: the Mix step is active (green), with its active time shown on the right

*__Tip:__ Outputs set in an SFC step are **held**; to turn them off, write `:= FALSE` in a later step (in this example, Mix closes the inlet valve, Drain stops the mixer, and Idle closes the drain valve).
You can also click **Text** in "Chart" (Ctrl+T) to enter the .sfc text above directly, then click **Back to the diagram**. For branches, jumps and the execution semantics, see sections 8.4 to 8.6.

6.7 C/C++: Sending a message on Serial1

When a button is pressed, user code (C/C++) sends a line of text from `Serial1` (COM1 of the controller), for example to a barcode printer, a host computer or another device. The ladder diagram handles the button, C/C++ handles the communication, and the two share GVL variables.

Scenario: Each press of the button makes `Serial1` send `Button pressed, count = N` at 9600 bps. Sketch name: `Ex67_Serial`.

Variable	Type	I/O mapping	Purpose
`Xbtn`	BOOL	`LocalIO.X00`	Send button
`Nmsg`	DINT	—	Number of presses (message number), written only by the ladder diagram

Program 1 (Main, .lad text): the rising edge of the button adds 1 to `Nmsg`.

```
PROGRAM Main
VAR
END_VAR
NETWORK 'One press: message number + 1'
    PT(Xbtn) ADD(Nmsg, Nmsg, 1)
END_PROGRAM
```

Program 2 (user code SendMsg.cpp86): when `Nmsg` changes, it sends a line of message.

```
// SendMsg: when Nmsg changes, send a line of message from Serial1

void plc86_user_setup(void) {
    Serial1.begin(9600);               // COM1, 9600 bps
}

// Every pass of loop(): only work that never blocks (no delay())
void plc86_user_loop(void) {
    static int32_t sent = 0;           // number already sent
    int32_t n = GVL(Nmsg);
    if(n != sent) {
        sent = n;
        Serial1.print("Button pressed, count = ");
        Serial1.println(n);
    }
}
```

Detailed steps

#	Action
1	Create the sketch `Ex67_Serial` and a new project; declare `Xbtn : BOOL;` and `Nmsg : DINT;` in the GVL.
2	Open `Main`, network 1: press **P** (positive transition contact PT) → `Xbtn`; press **A** (ADD) → "dest" `Nmsg`, "a" `Nmsg`, "b" `1` → OK.
3	"Project > Add a user code file (C/C++)...", name `SendMsg` → OK. The template already lists the hook functions: change `plc86_user_setup()` and `plc86_user_loop()` to the content above, delete the unused `plc86_user_before()` and `plc86_user_after()`, and press Ctrl+S to save.
4	"I/O mapping": `Xbtn` → `LocalIO.X00`.
5	Shift+F7 to check; F5 to simulate: double-click `LocalIO.X00` repeatedly in "Simulation I/O"; `Nmsg` in "Watch" adds 1 with each press.

#	Action
6	Connect COM1 to the computer (for example with a USB to RS-232 / 485 adapter), then F8 to upload.
7	In "Output" at the bottom, switch to **Serial monitor**, choose the serial port of the adapter and the baud rate `9600`, and click **Connect**; you receive a line of message each time you press X00.

Expected result: Each press of X00 adds 1 to `Nmsg`, and `Serial1` sends a line `Button pressed, count = 1`, `= 2`, ...

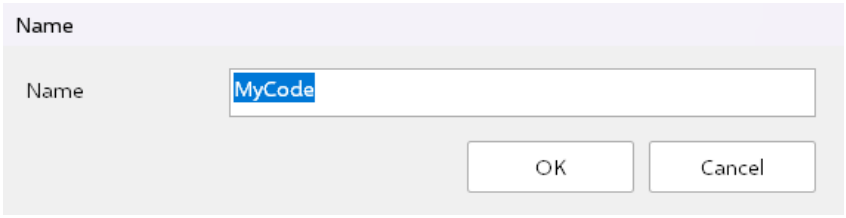


Figure 6-22 "Project > Add a user code file (C/C++)...": enter the file name

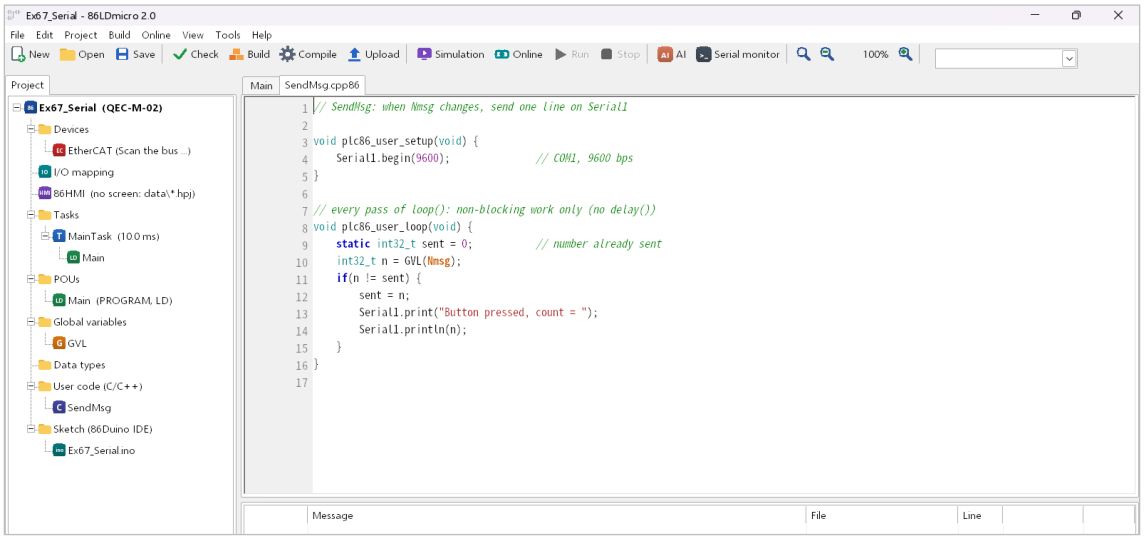


Figure 6-23 The user code SendMsg.cpp86 of 6.7: plc86_user_setup() and plc86_user_loop()

***Note:** The USB `Serial` is used for uploading and for the online view; use `Serial1` to `Serial3` to communicate with external devices.
In C/C++, a PLC variable must be written as `GVL(Nmsg)`, not by its name alone. `plc86_user_loop()` runs in every `loop()`; do not use `delay()`.
In this example the ladder diagram writes `Nmsg` and C/C++ only reads it, following principle ③ of 6.0. For the hook functions and other ways of writing, see Chapter 14.

Chapter 7 Ladder diagram instruction reference

The ladder diagram of 86LDmicro 2.0 has 90 instructions in all (the same as those listed by the command line `86ldmicro-cli instructions`); this chapter describes them one by one in 10 groups by purpose. Each section contains an introduction, screenshots of the menu or parameter window, and an instruction table: **Instruction** (the name in .lad), **.lad syntax** (an example that you can paste directly into text mode), **Key** (the key to press directly after selecting an element or a network) and **Description**.

- Instructions marked **[1.x]** are LDmicro 1.x compatible instructions: they behave as in 1.1, and their timers count in PLC cycles. For new programs, the IEC function blocks (section 7.8) are recommended.
- A parameter can be a variable, a constant or an ST expression; write times as IEC time constants such as `T#500ms` and `T#2s`.
- For instructions with no button in the Toolbox, click **...** in "Instructions" in the side panel and choose from the "Contacts", "Coils", "Boxes", "Program flow" and "Function blocks" submenus (the menus show a description of each instruction).
- After any instruction is inserted, a parameter window opens with one field per parameter; for "variable" fields, you can choose a declared variable from the drop-down list.

7.1 Editing

The ladder diagram tab has the declaration area (`VAR ... END_VAR`) at the top, the networks in the middle, and the Toolbox in the side panel on the right. This section describes how to insert, select, move and delete elements and networks, as well as the text mode and Common circuits.

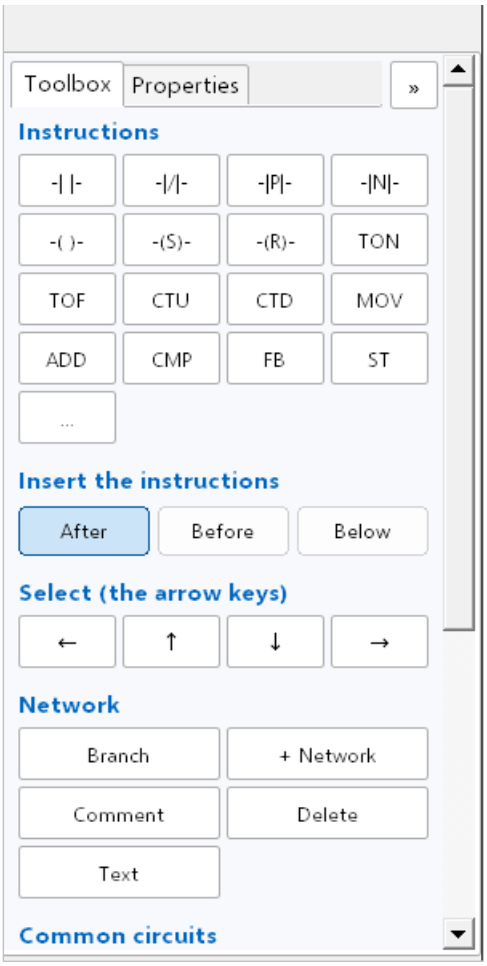


Figure 7-1 The Toolbox in the side panel of the ladder diagram: Instructions, Insert the instructions, Select (the arrow keys), Network, Common circuits

Toolbox section	Contents
Instructions	Buttons for common elements; hover the mouse over one to see the description and key of the instruction; **...** lists all instructions by category
Insert the instructions	**After**: after the selected element (in series; when a whole network is selected, before the coils); **Before**: before the selected element; **Below**: a parallel branch below the selected element
Select (the arrow keys)	← →: the element to the left / right; ↑ ↓: the element above / below, or the previous / next network
Network	**Branch** (adds a parallel branch below the selected element, same as the B key), **+ Network**, **Comment** (the network title and comment), **Delete**, **Text** (edits the whole POU as .lad text, Ctrl+T)
Common circuits	Ready-made network templates: select one and click **Insert** (or double-click), fill in the names, and it is inserted

7.1.1 Inserting elements

First select an element (the insertion point is based on it) or a network (click the network's title bar), then insert in any of the following ways:

- **Buttons**: click a button in the "Instructions" section of the Toolbox. Hold **Shift** while clicking a button to insert before the selected element.
- **Keys**: press a key from the table below; **Shift+key** inserts before the selected element; **B** adds a normally open contact in parallel below the selected element.
- **The "..." menu**: click **...** and choose any instruction by category.

Key	Instruction	Key	Instruction	Key	Instruction
C	Normally open contact NO	S	Set coil SET	U	Up counter CTU [1.x]
N	Normally closed contact NC	R	Reset coil RST	D	Down counter CTD [1.x]
P	Rising edge contact PT	O	On delay TON [1.x]	M	Move MOV
L	Coil OUT	F	Off delay TOF [1.x]	A	Add ADD
E	Condition contact CMP	K	Function block CALL (FB)	T	ST box
B	Normally open contact in parallel below	Shift+key	Insert before the selected element		

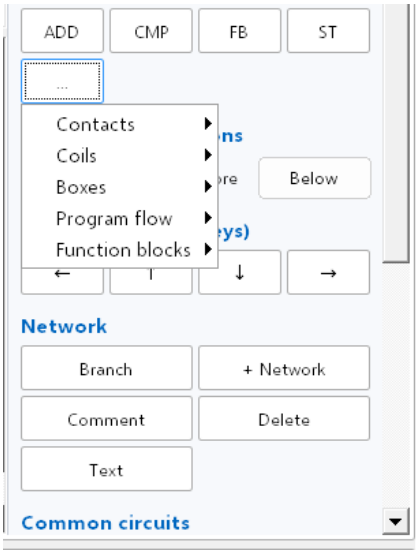


Figure 7-2 "..." lists all instructions by category: Contacts, Coils, Boxes, Program flow, Function blocks

The insertion point is set by "Insert the instructions". For example, inserting a normally closed contact while `Xstart` is selected:

Insert the instructions	Result (.lad)
After	`NO(Xstart) NC(Xstop)`
Before (or Shift+N)	`NC(Xstop) NO(Xstart)`
Below	`[ NO(Xstart) | NC(Xstop) ]`

Coils (OUT, SET, RST...) always go on the right side of the network; several coils in a row at the end of a network all sit next to the right rail and are driven by the same power flow.

7.1.2 Selecting, changing, moving and deleting

Action	How
Select an element	Click the element, or use ← → ↑ ↓
Select a network	Click the network's title bar (the row with its number); Home / End selects the first / last network
Change an element's parameters	Double-click the element, or select it and press Enter
Move an element	Ctrl+← / Ctrl+→: swap places with the element to the left / right in the same series
Move a network	Select the network, then Ctrl+↑ / Ctrl+↓
Delete	Del (or Backspace), or **Delete** in "Network"
Cut, copy, paste	Ctrl+X / Ctrl+C / Ctrl+V (an element or a whole network)
Undo, redo	Ctrl+Z / Ctrl+Y

Right-click an element: besides Edit, Cut, Copy, Paste and Delete, it offers commands for its variable: **I/O mapping of <variable>...**, **86HMI binding of <variable>...**, **Add <variable> to the watch window**, **Add to the trace**, **Where is <variable> used?**, **Its declaration**, **Properties**; in simulation or online, also **Write**, **Force**, **Unforce**.

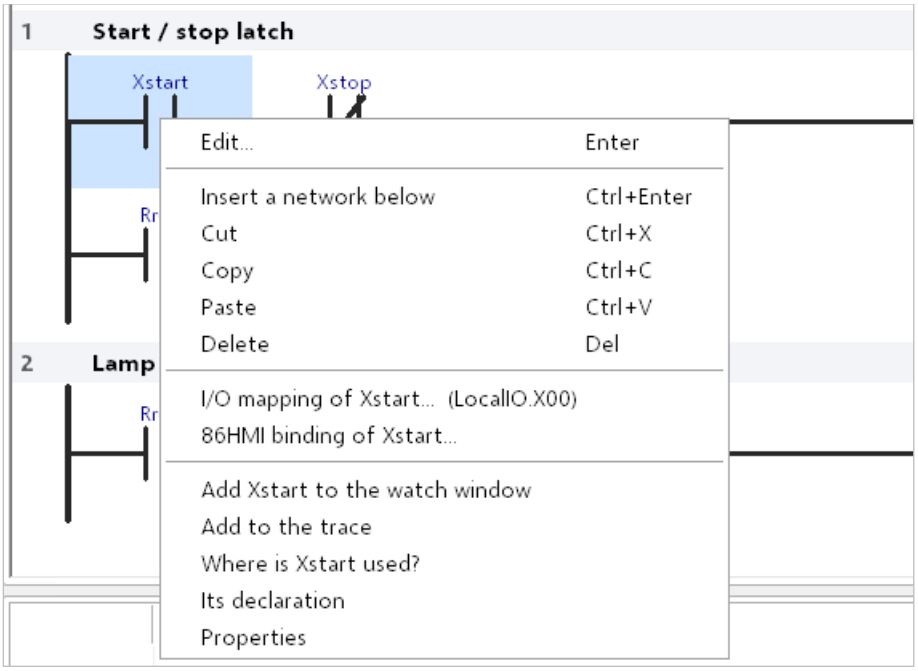


Figure 7-3 The context menu of an element: edit, network, clipboard, and the variable's I/O mapping, watch and cross reference

*__Note:__ The ladder diagram cannot be edited in simulation or online (the status bar shows "Stop the simulation / go offline to edit"); double-clicking a contact or coil then toggles the value of its variable.

7.1.3 Networks

Action	How
Add a network	**+ Network** in "Network" or Ctrl+Enter: inserts after the selected network; Ctrl+Shift+Enter: before it
Title and comment	Double-click the network's title bar; or right-click "Network title and comment..."; or **Comment** in "Network"
Delete a network	Select the network, then Del or **Delete** in "Network"
Empty network	Shows "(empty: select the network and insert an element)"; select it and insert

The network window has three fields: **Title** (shown above the network), **Comment** (several lines, shown below the title) and **Body (text)** (this network as .lad text: edit it directly or paste a whole piece of logic; the syntax is checked when you click OK).

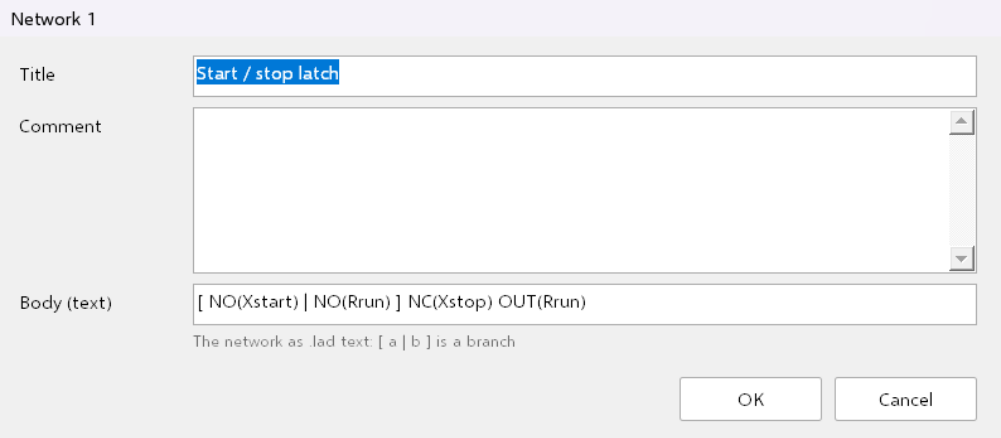


Figure 7-4 The network's title, comment and body (.lad text)

7.1.4 Text mode

Click **Text** in "Network" (or press Ctrl+T) to switch the whole POU to .lad text editing, which suits large changes, copy and paste, or comparing versions with version control; the format is checked when you click **Back to the diagram** at the top. When the file has a syntax error, it opens in text mode and shows where the error is.

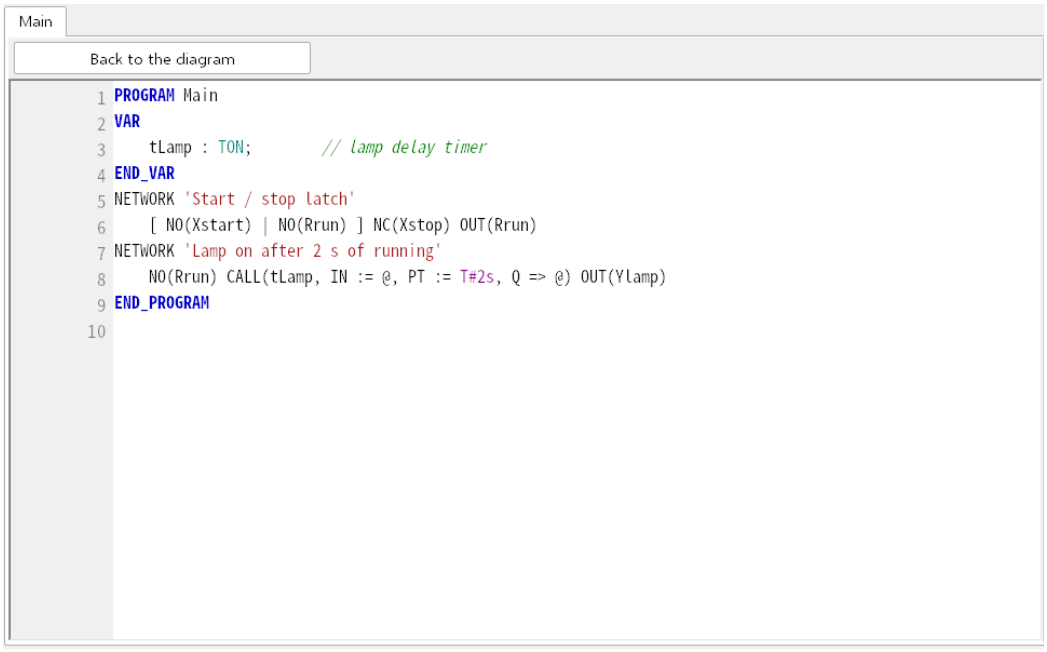


Figure 7-5 Text mode: the whole POU as .lad text, with "Back to the diagram" at the top

```
PROGRAM Main
VAR
    tLamp : TON;
END_VAR
NETWORK 'Start / stop latch'
    // Comment
    [ NO(Xstart) | NO(Rrun) ] NC(Xstop) OUT(Rrun)
NETWORK
    NO(Rrun) CALL(tLamp, IN := @, PT := T#2s, Q => @) OUT(Ylamp)
END_PROGRAM
```

- It starts with `PROGRAM` / `FUNCTION_BLOCK` and ends with the matching `END_…`; the declaration section is written as in ST.
- `NETWORK 'title'` starts a network; the `//` lines after it are comments.
- A network is a **series** of elements separated by spaces; `[ a | b | c ]` are **parallel** branches, and each branch can itself be a series or contain nested parallel branches.
- An element is written `instruction(parameter, ...)`; an instruction without parameters is just its name (for example `OSR`, `MCR`). Function block parameters are written `name := value` and outputs `name => target`; `@` stands for the power flow.

7.1.5 Common circuits

"Common circuits" in the side panel offers ready-made network templates: select one (its description appears below), click **Insert** (or double-click) and fill in the variable names and times in the form; the network is inserted after the selected network, and new variables are declared automatically (inputs, outputs and internal bits in the GVL, timers and counters in this POU).

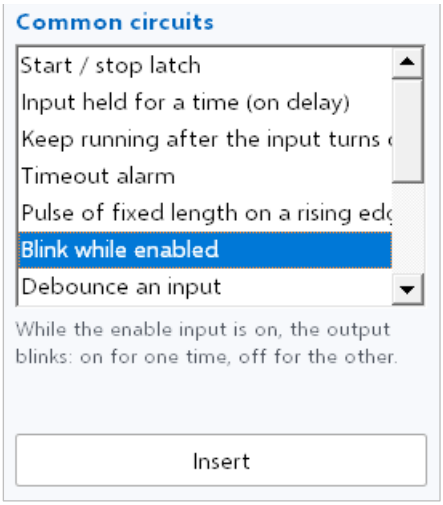


Figure 7-6 The Common circuits list: select one to see its description below, then click "Insert"

Template	Network created (.lad, default names)
Start / stop latch	`[ NO(Xstart) | NO(Ymotor) ] NC(Xstop) OUT(Ymotor)`
Input held for a time (on delay)	`NO(Xsensor) TON(Tconfirm, T#2000ms) OUT(Yconfirmed)`
Delayed off after the input goes off (off delay)	`NO(Xheater) TOF(Tafter, T#5000ms) OUT(Yfan)`
Timeout alarm	`NO(Rstep) NC(Xdone) TON(Ttimeout, T#10000ms) SET(Yalarm); NO(Xreset) RST(Yalarm)`
Pulse of fixed length on a rising edge	`PT(Xtrigger) SET(Ypulse); NO(Ypulse) TON(Tpulse, T#500ms) RST(Ypulse)`
Blink while enabled	Two TONs SET / RST the output in turn; RST when the enable is OFF
Debounce an input	`NO(Xswitch) TON(Tdb_on, T#50ms) SET(Rswitch); NC(Xswitch) TON(Tdb_off, T#50ms) RST(Rswitch)`
Count to N	`NO(Xpart) CTU(Cparts, 10) OUT(Ydone); NO(Xreset) RES(Cparts)`
Forward / reverse with interlock	One latch network each for forward and reverse, interlocked through each other's normally closed contacts
Level alarm with hysteresis	`GRT(Alevel, 3000) SET(Yhigh); LES(Alevel, 2800) RST(Yhigh)`
One button on / off (toggle)	`NO(Xbutton) TOGGLE(Ylight)`
Square wave (cyclic timer)	`NO(Xrun) TCY(Tcycle, T#1000ms) OUT(Yclock)`
IEC timer block (TON)	`NO(Xstart) CALL(tonStart, IN := @, PT := T#1000ms, Q => @) OUT(Ydelayed)`

7.2 Contacts and coils

A contact reads a BOOL value and decides whether power flow passes; a coil writes a BOOL variable according to the power flow. A contact's variable can be a BOOL variable in the GVL or in VAR, or a BOOL output of a function block (for example `NO(tLamp.Q)`).

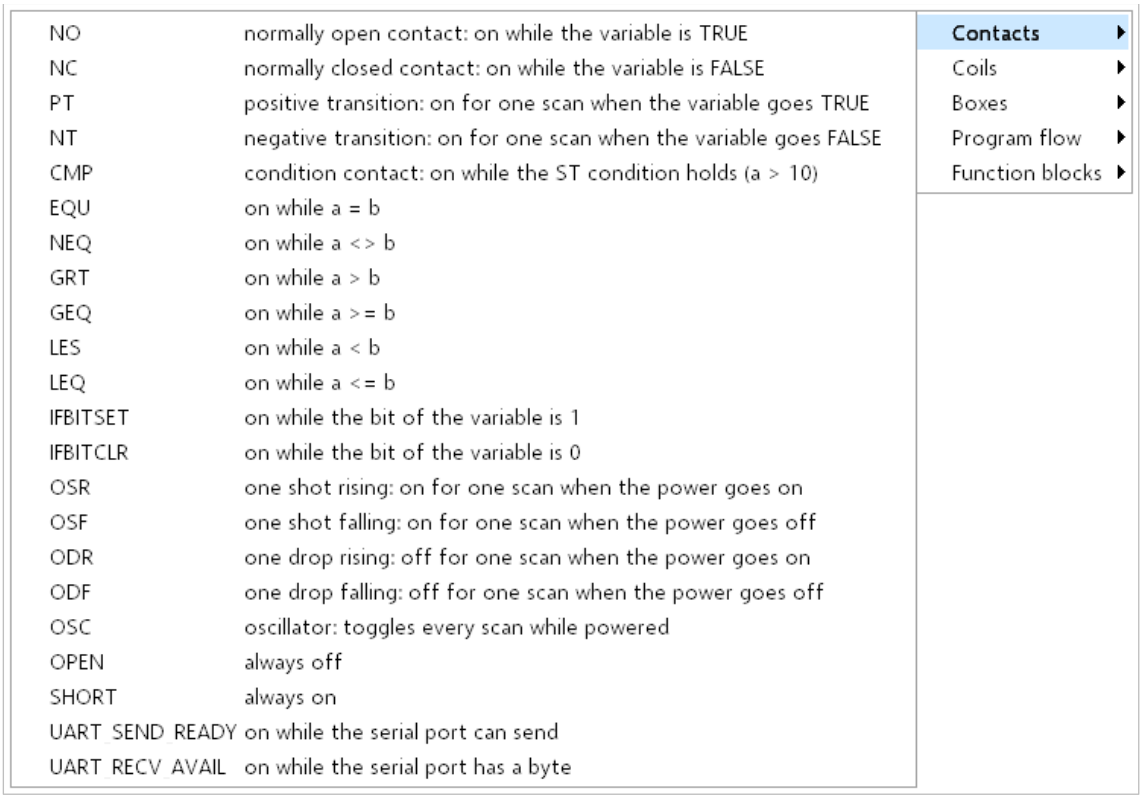


Figure 7-7 "..." > "Contacts": all contact instructions

Instruction	.lad syntax	Key	Description
NO	`NO(Xstart)`	C	Normally open contact: on when the variable is TRUE
NC	`NC(Xstop)`	N	Normally closed contact: on when the variable is FALSE
OPEN	`OPEN`	—	Always open (for example to cut part of a circuit temporarily)
SHORT	`SHORT`	—	Always closed (for example to skip a condition temporarily)
OUT	`OUT(Ylamp)`	L	Coil: the variable follows the power flow (TRUE with power flow, FALSE without)
OUTN	`OUTN(Yready)`	—	Negated coil: the variable is the inverse of the power flow
SET	`SET(Yrun)`	S	Set coil: with power flow, sets the variable to TRUE and keeps it (no change without power flow)
RST	`RST(Yrun)`	R	Reset coil: with power flow, sets the variable to FALSE and keeps it
TOGGLE	`TOGGLE(Ylight)`	—	Toggle coil: inverts the variable on each rising edge of the power flow [1.x]

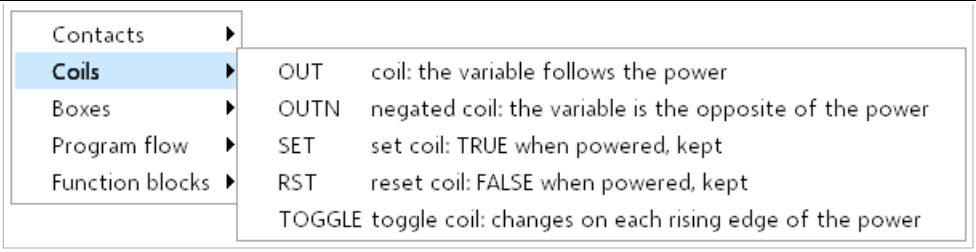


Figure 7-8 "..." > "Coils": OUT, OUTN, SET, RST, TOGGLE

***Tip:** Drive each variable with only one `OUT` coil; when several conditions are needed, put them in parallel or in series before that one coil (section 6.0, principle ③). When `SET` / `RST` are used as a pair, the later network wins.

7.3 Edges and pulses

Edge instructions are on only during the one scan in which the state changes, for actions done "once per press" (counting, adding 1, sending a record). `PT` and `NT` detect a variable; `OSR`, `OSF`, `ODR` and `ODF` detect the power flow coming in from the left.

Instruction	.lad syntax	Key	Description
PT	`PT(Xstart)`	P	Rising edge contact: on for one scan when the variable goes from FALSE to TRUE
NT	`NT(Xstart)`	—	Falling edge contact: on for one scan when the variable goes from TRUE to FALSE
OSR	`NO(Xa) OSR`	—	One shot rising: on for one scan when the power flow goes on [1.x]
OSF	`NO(Xa) OSF`	—	One shot falling: on for one scan when the power flow goes off [1.x]
ODR	`NO(Xa) ODR`	—	Rising edge off: off for one scan when the power flow goes on [1.x]
ODF	`NO(Xa) ODF`	—	Falling edge off: off for one scan when the power flow goes off [1.x]
OSC	`NO(Xrun) OSC OUT(Yclk)`	—	Oscillator: toggles every scan while powered (the period is 2 scans) [1.x]

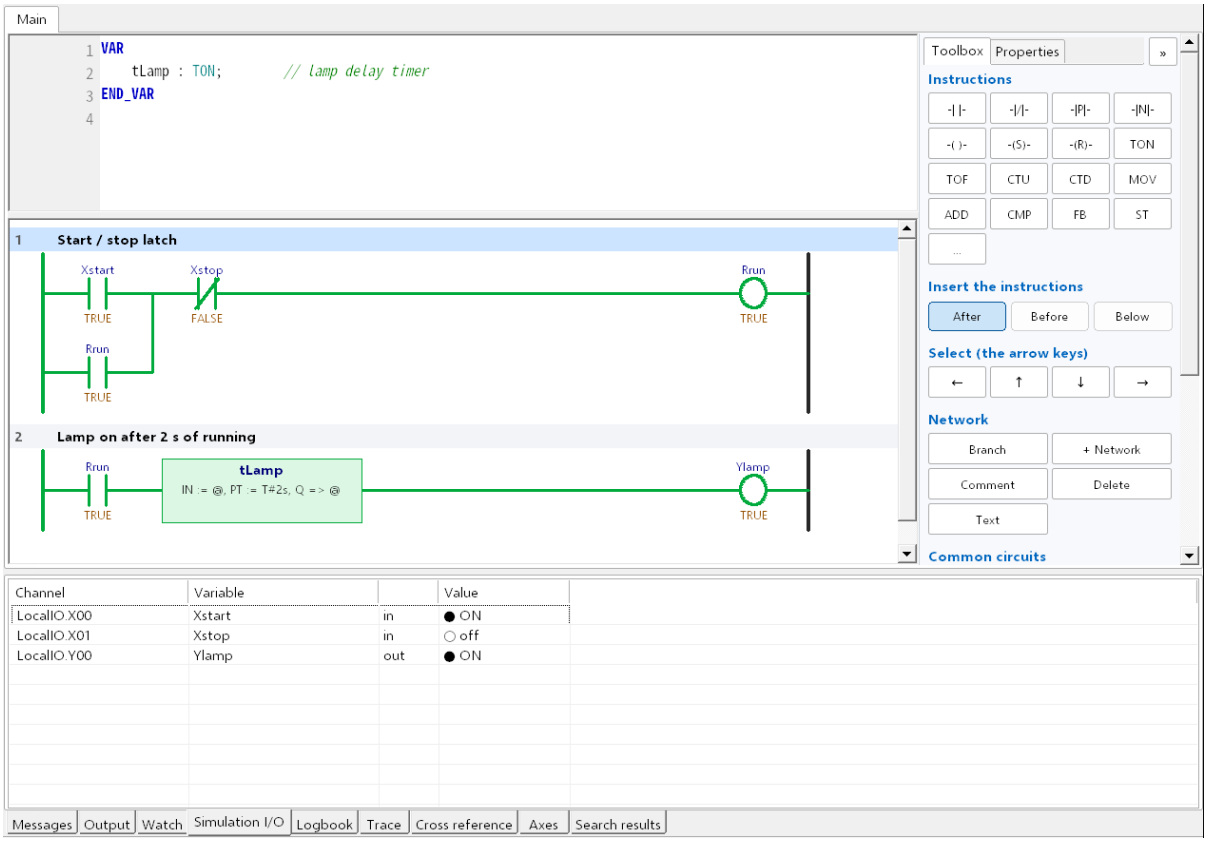


Figure 7-9 An edge contact example (network 3): each press of PT(Xstart) makes ADD add 1 to Count

Pulse output: the following 1.x instructions generate pulse trains on a digital output. The output parameter is a BOOL variable mapped to a physical output in the I/O mapping. For positioning that needs acceleration and deceleration, use a Motion86 axis instead (Chapter 12).

Instruction	.lad syntax	Key	Description
NPULSE	`PT(Xgo) NPULSE(100, 1000, Ypls)`	—	On a rising edge of the power flow, sends N pulses on the output; parameters: count, frequency (Hz), output variable [1.x]
NPULSE_OFF	`NO(Xstop) NPULSE_OFF`	—	Stops the N pulses while there is power flow [1.x]
PULSER	`NO(Xa) PULSER(10, 10, 0, pc, Ypls)`	—	Outputs a pulse train on the coil variable; parameters: P1, P0 (ON and OFF lengths), acceleration, counter, coil [1.x]
STEPPER	`NO(Xb) STEPPER(stp, Nsteps, 1, 1, 0, Ystep)`	—	Stepper motor pulses; parameters: name, maximum (steps), P, n, curve, coil. Acceleration tables are not supported (error when n > 1 and curve > 0; use NPULSE or Motion86 instead) [1.x]

***Tip:** The IEC edge detection function blocks `R_TRIG` and `F_TRIG` are used with CALL, for example `NO(Xa) CALL(rStart, CLK := @, Q => @) ADD(Count, Count, 1)` (section 7.8).

7.4 Timers

There are two kinds of timers: **IEC function blocks** (TON, TOF, TP, timed in real time and inserted with CALL, section 7.8) and **1.x timer boxes** (the table in this section). The first parameter of a 1.x timer is the timer variable (declared as an integer such as `DINT`; it holds the number of PLC cycles elapsed), and it times by counting PLC cycles: delay ÷ task interval, as an integer.

- **Delay**: write `T#500ms`, `T#2s`. A number without a unit is taken as microseconds, as in 1.x, and the check gives a warning. A delay shorter than the task interval cannot be timed (warning).
- **Adjust** (optional): the number of PLC cycles added to the delay (may be negative), as in 1.x.
- TON, TOF, TCY, THI and TLO reset automatically when the input changes; RTO and RTL accumulate time and are cleared with `RES` (section 7.5).

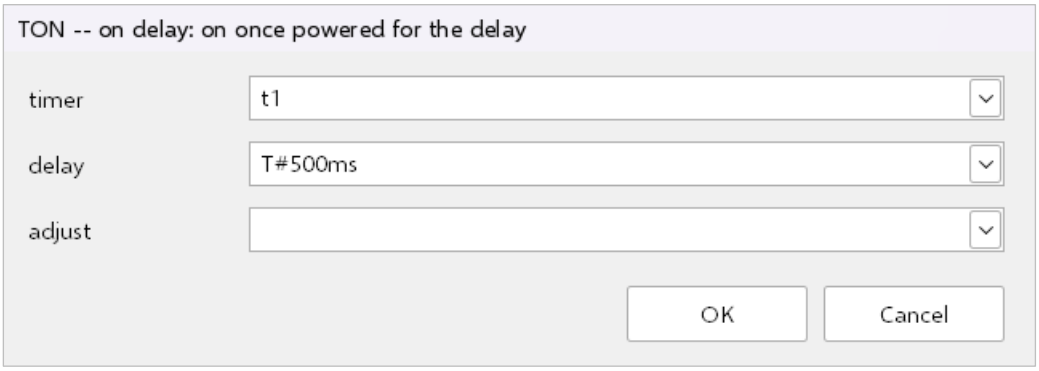


Figure 7-10 The parameter window of a 1.x TON: timer, delay, adjust

Instruction	.lad syntax	Key	Description
TON	`NO(Xa) TON(t1, T#500ms) OUT(Ya)`	O	On delay: on once powered for the delay time [1.x]
TOF	`NO(Xa) TOF(t2, T#1s) OUT(Ya)`	F	Off delay: stays on for the delay time after the power goes off [1.x]
RTO	`NO(Xa) RTO(t3, T#2s) OUT(Yb)`	—	Retentive on delay: accumulates the powered time and turns on when it reaches the delay (cleared by RES) [1.x]
RTL	`NO(Xa) RTL(t4, T#2s) OUT(Yb)`	—	Retentive off delay: accumulates the unpowered time [1.x]
TCY	`NO(Xrun) TCY(t5, T#500ms) OUT(Yclk)`	—	Cyclic timer: toggles ON / OFF each time the delay elapses while powered [1.x]
THI	`NO(Xa) THI(t6, T#1s) OUT(Yp)`	—	On a rising edge, outputs a pulse as long as the delay [1.x]
TLO	`NO(Xa) TLO(t7, T#1s) OUT(Yp)`	—	On a falling edge, outputs a low pulse as long as the delay [1.x]
T2CNT	`NO(Xa) T2CNT(Dcycles, T#1s)`	—	Converts the delay into a number of PLC cycles and stores it in a variable [1.x]

IEC timer function blocks (recommended): declare the instance in VAR (for example `tLamp : TON;`) and insert it with **FB** (K).

Function block	.lad syntax	Behavior
TON	`NO(Xa) CALL(tLamp, IN := @, PT := T#2s, Q => @) OUT(Ya)`	Q = TRUE once IN has stayed TRUE for PT; when IN goes FALSE, Q and ET return to zero at once
TOF	`NO(Xa) CALL(tOff, IN := @, PT := T#5s, Q => @) OUT(Yb)`	Q = TRUE while IN is TRUE; after IN goes FALSE, Q stays on for PT more
TP	`NO(Xa) CALL(tPulse, IN := @, PT := T#1s, Q => @) OUT(Yc)`	On a rising edge of IN, Q = TRUE for exactly PT; changes of IN during that time have no effect

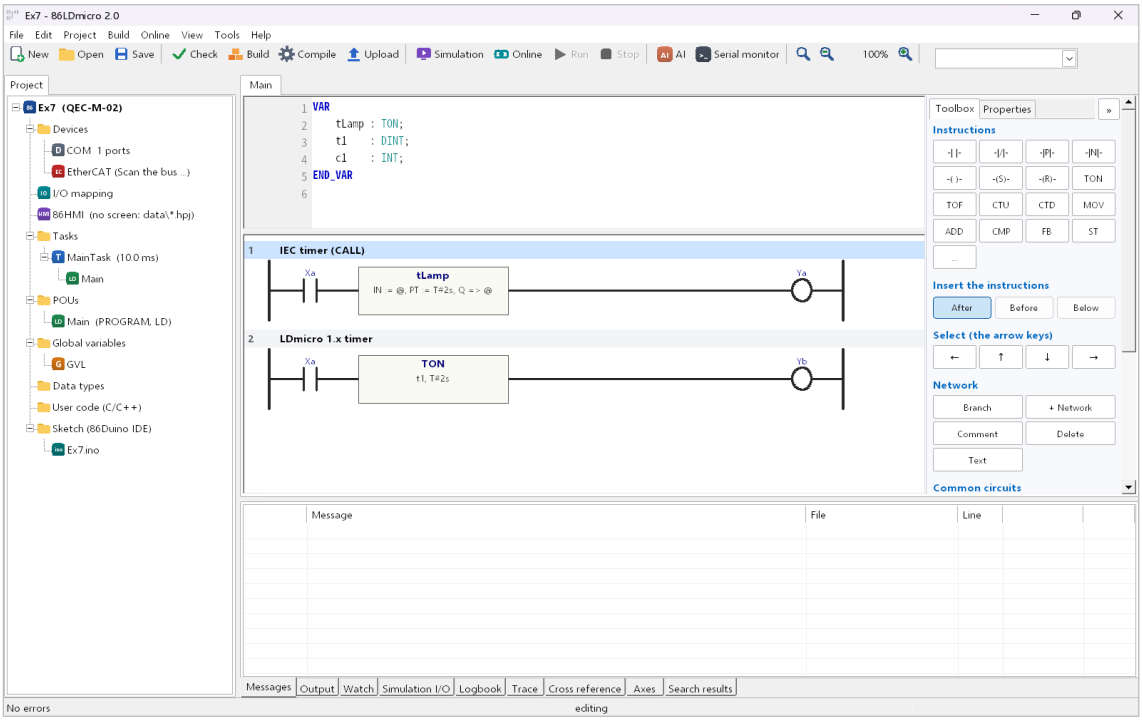


Figure 7-11 An IEC TON (top) and a 1.x TON (bottom) in simulation: the function block shows its parameters, the 1.x box shows the cycles counted

7.5 Counters

The first parameter of a 1.x counter is the counter variable (declared as an integer such as `INT`); its value is the current count and can be used directly in compare instructions. For the IEC counter function blocks (CTU, CTD, CTUD), see section 7.8.

- Parameters: **Counter** (the variable), **Max value** / **Min value**, **Initial value** (optional, 0 by default), **Input** (optional, when to count: `RISING` (default, rising edge), `FALLING` (falling edge), `HIGH` (every scan while powered), `LOW` (every scan while unpowered)).
- `RES` clears a timer, counter or variable to 0; `RES` is a box and goes at the far right of the network.

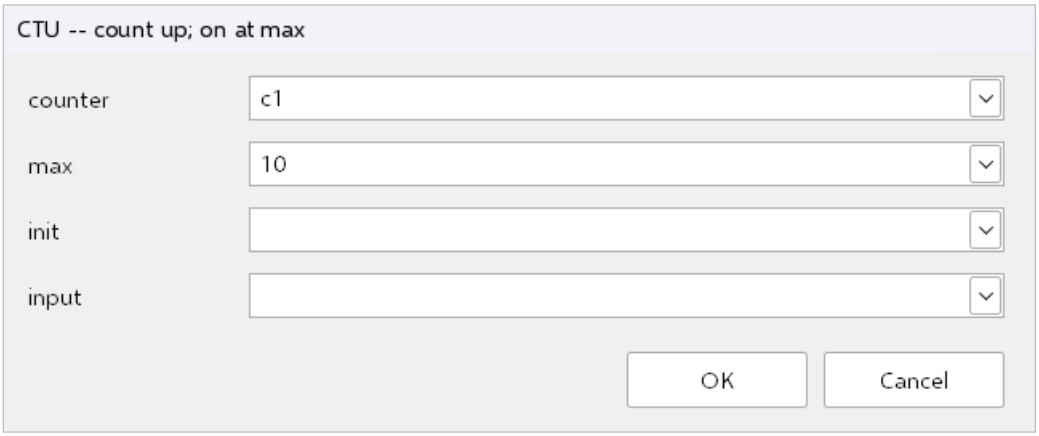


Figure 7-12 The parameter window of a 1.x CTU: counter, max value, initial value, input

Instruction	.lad syntax	Key	Description
CTU	`NO(Xa) CTU(c1, 10) OUT(Ya)`	U	Counts up; on when the count ≥ the max value [1.x]
CTD	`NO(Xb) CTD(c2, 0, 10) OUT(Yb)`	D	Counts down (initial value 10 in this example); on while above the min value [1.x]
CTC	`NO(Xb) CTC(c3, 9, 0, FALLING)`	—	Circular counter: when the count passes the max value it goes back to the initial value, and it is on for that count (counts on falling edges in this example) [1.x]
CTR	`NO(Xb) CTR(c4, 0, 9)`	—	Circular down counter: when the count drops below the min value it goes back to the initial value, and it is on for that count [1.x]
RES	`NO(Xreset) RES(c1)`	—	Resets a timer / counter / variable to 0 [1.x]

7.6 Comparison

Compare instructions are all **contacts**: on when the condition holds. We recommend **CMP** (the condition contact, key E), which takes an ST comparison expression directly, can compare any type (INT, REAL, TIME, function block members...) and can combine AND / OR; the 1.x EQU to LEQ can still be used.

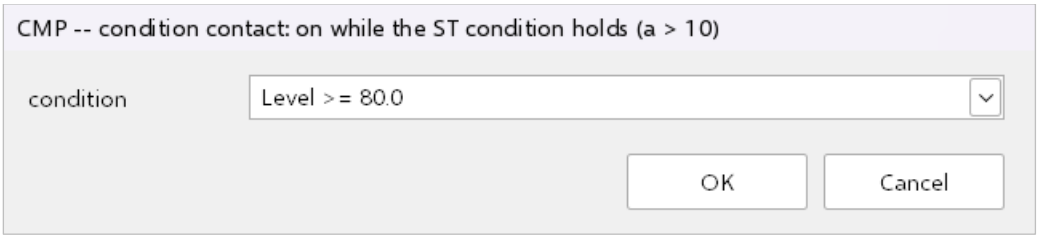


Figure 7-13 The parameter window of the CMP condition contact: enter an ST condition

Instruction	.lad syntax	Key	Description
CMP	`CMP(Level >= 80.0) OUT(Yhigh)`	E	Condition contact: on while the ST condition holds, for example `Count = 5`, `tLamp.ET > T#1s`, `(a > 10) AND NOT b`
EQU	`EQU(Count, 10)`	—	On when a = b [1.x]
NEQ	`NEQ(Count, 0)`	—	On when a <> b [1.x]
GRT	`GRT(Level, 5)`	—	On when a > b [1.x]
GEQ	`GEQ(Level, 5)`	—	On when a >= b [1.x]
LES	`LES(Level, 90)`	—	On when a < b [1.x]
LEQ	`LEQ(Level, 90)`	—	On when a <= b [1.x]
IFBITSET	`IFBITSET(Nbits, 3)`	—	On when bit n of the variable is 1 (bit 0 is the lowest bit) [1.x]
IFBITCLR	`IFBITCLR(Nbits, 0)`	—	On when bit n of the variable is 0 [1.x]

7.7 Arithmetic and move

Arithmetic and move instructions are all **boxes**: they compute once whenever there is power flow (that is, every scan; to do it only once, put an edge contact such as PT before it). The first parameter is the destination variable of the result. In the four basic operations, if either operand is a REAL the calculation is done in floating point, and an integer destination receives the nearest integer. For complex calculations, an ST box (section 7.8) or an ST program is recommended.

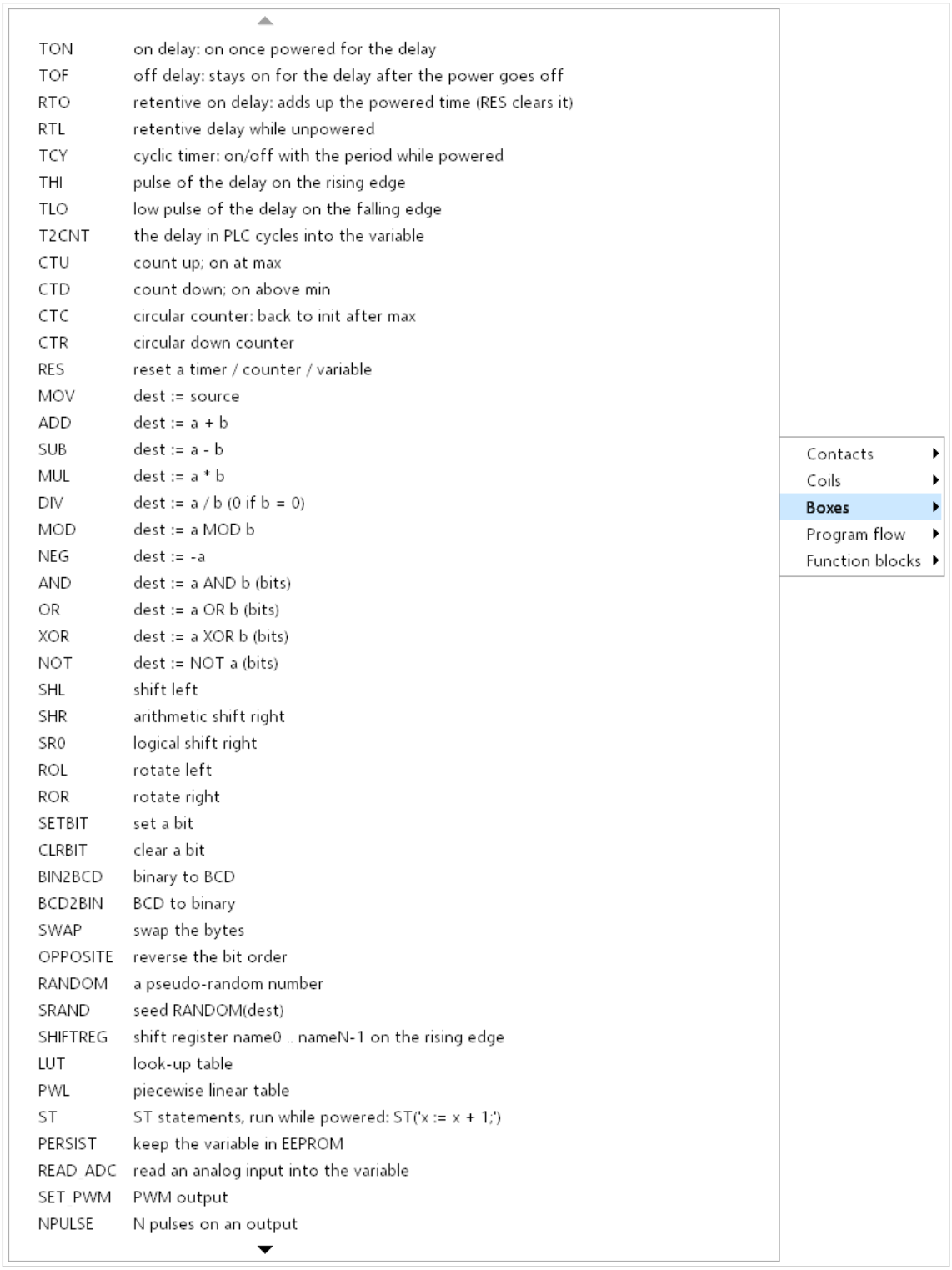


Figure 7-14 "..." > "Boxes": timer, counter, arithmetic, move and I/O boxes

7.7.1 Move and the four basic operations

Instruction	.lad syntax	Key	Description
MOV	`MOV(Val, 100)`	M	dest := source
ADD	`ADD(Count, Count, 1)`	A	dest := a + b
SUB	`SUB(Val, Val, 1)`	—	dest := a - b
MUL	`MUL(Val, Val, 2)`	—	dest := a * b
DIV	`DIV(Val, Val, 3)`	—	dest := a / b (an integer division by 0 gives 0)
MOD	`MOD(Val, Val, 7)`	—	dest := a MOD b (remainder)
NEG	`NEG(Val, Val)`	—	dest := -a

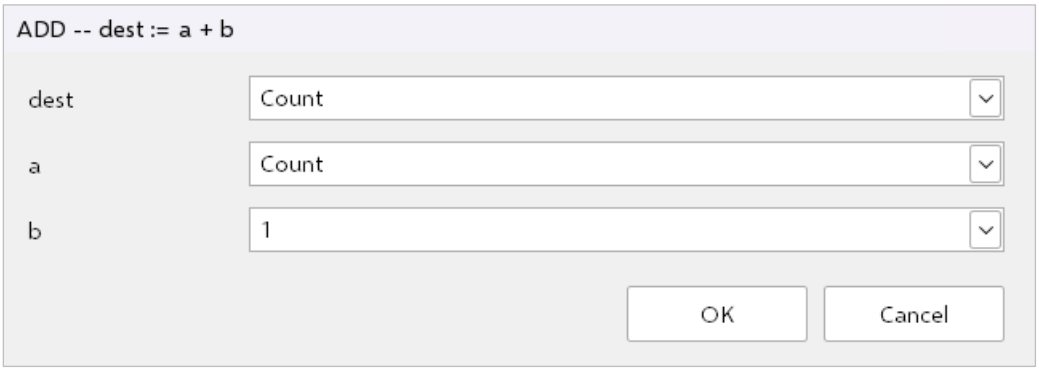


Figure 7-15 The parameter window of ADD: dest, a, b

7.7.2 Bit operations

Instruction	.lad syntax	Key	Description
AND	`AND(Nbits, Nbits, 16#0F)`	—	dest := a AND b (bitwise) [1.x]
OR	`OR(Nbits, Nbits, 1)`	—	dest := a OR b (bitwise) [1.x]
XOR	`XOR(Nbits, Nbits, 2)`	—	dest := a XOR b (bitwise) [1.x]
NOT	`NOT(Nbits, Nbits)`	—	dest := NOT a (bitwise inversion) [1.x]
SHL	`SHL(Nbits, Nbits, 1)`	—	Shift left by n bits [1.x]
SHR	`SHR(Nbits, Nbits, 1)`	—	Arithmetic shift right (keeps the sign) [1.x]
SR0	`SR0(Nbits, Nbits, 1)`	—	Logical shift right (fills with 0 on the left) [1.x]
ROL	`ROL(Nbits, Nbits, 1)`	—	Rotate left [1.x]
ROR	`ROR(Nbits, Nbits, 1)`	—	Rotate right [1.x]
SETBIT	`SETBIT(Nbits, 2)`	—	Sets bit n of the variable to 1 [1.x]
CLRBIT	`CLRBIT(Nbits, 5)`	—	Clears bit n of the variable to 0 [1.x]

7.7.3 Conversion, random numbers and tables

Instruction	.lad syntax	Key	Description
BIN2BCD	`BIN2BCD(Nbcd, Val)`	—	Binary to BCD [1.x]
BCD2BIN	`BCD2BIN(Val, Nbcd)`	—	BCD to binary [1.x]
SWAP	`SWAP(Nraw, Val)`	—	Swaps the bytes [1.x]
OPPOSITE	`OPPOSITE(Nraw, Val)`	—	Reverses the bit order [1.x]
RANDOM	`RANDOM(Val)`	—	Stores a pseudo-random number in dest [1.x]
SRAND	`SRAND(Val, 1234)`	—	Sets the seed of RANDOM [1.x]
SHIFTREG	`NO(Xclk) SHIFTREG(sreg, 4)`	—	Shift register: on a rising edge of the power flow, shifts `sreg0`→`sreg1`→…→`sreg3` in order; declare `sreg0` to `sreg3` yourself (same type) [1.x]
LUT	`LUT(Nout, Nidx, 10, 20, 40, 80)`	—	Look-up table: dest := the value at "index" (the index starts at 0) [1.x]
PWL	`PWL(Nout, Nraw, 0, 0, 100, 50, 200, 200)`	—	Piecewise linear table: converts the index into dest along the polyline (x0, y0), (x1, y1)...; x must increase, the values are integers, and when the values are too large the check suggests making them smaller [1.x]

7.8 IEC function blocks (CALL) and the ST instruction

CALL puts any function block in a network: the IEC standard function blocks, the PLCopen / Motion86 motion control function blocks (section 7.11) and the FUNCTION_BLOCKs you write in the project (section 8.3). Declare the function block instance in VAR (or the GVL) first.

1. Declare the instance in the declaration section, for example `tLamp : TON;`, `cParts : CTU;`, `rStart : R_TRIG;`.
2. Select an element or a network and click **FB** (K).
3. In the "Element" field, enter `CALL(instance, parameter := value, output => target)` and click OK.

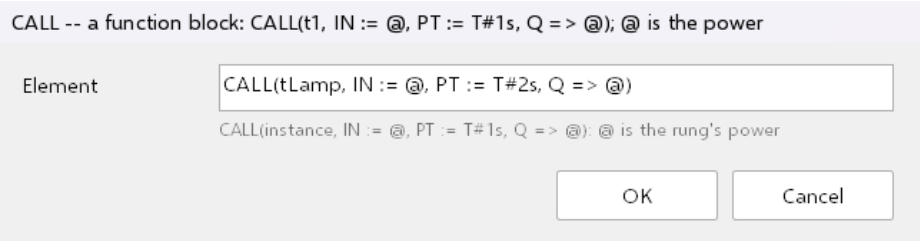


Figure 7-16 The parameter window of CALL: enter the whole CALL in the "Element" field

- `IN := @`: connects the power flow coming in from the left to this input; `Q => @`: this output becomes the power flow to the right (to the next elements).
- Other inputs can take a variable or a constant (`R := Xreset`, `PV := 10`), and outputs can be stored in a variable (`CV => Count`). Simply leave out the parameters you do not use.
- Function block outputs can also be read directly in other networks: `NO(tLamp.Q)`, `CMP(cParts.CV >= 5)`.

Function block	Inputs	Outputs	.lad example
TON / TOF / TP	IN, PT	Q, ET	`CALL(tLamp, IN := @, PT := T#2s, Q => @)`
CTU (CTU_DINT)	CU, R, PV	Q, CV	`CALL(cParts, CU := @, R := Xreset, PV := 10, Q => @, CV => Count)`
CTD (CTD_DINT)	CD, LD, PV	Q, CV	`CALL(cDown, CD := @, LD := Xload, PV := 5, Q => @)`
CTUD (CTUD_DINT)	CU, CD, R, LD, PV	QU, QD, CV	`CALL(cUD, CU := @, CD := Xdown, R := Xreset, PV := 10, QU => @)`
R_TRIG / F_TRIG	CLK	Q	`CALL(rStart, CLK := @, Q => @)`
SR (set dominant)	S1, R	Q1	`CALL(sr1, S1 := @, R := Xstop, Q1 => @)`
RS (reset dominant)	S, R1	Q1	`CALL(rs1, S := @, R1 := Xstop, Q1 => @)`

ST box: executes a piece of ST code while there is power flow (click **ST** or press T). Write a line break in a string as `$N`. It suits a small calculation inside a ladder diagram; write longer logic as an ST program (Chapter 8).

Instruction	.lad syntax	Key	Description
CALL	`NO(Xa) CALL(tLamp, IN := @, PT := T#2s, Q => @) OUT(Ya)`	K	Function block: `@` is the power flow
ST	`NO(Xa) ST('Count := Count + 1; Level := Count * 2;')`	T	ST statements, executed while there is power flow

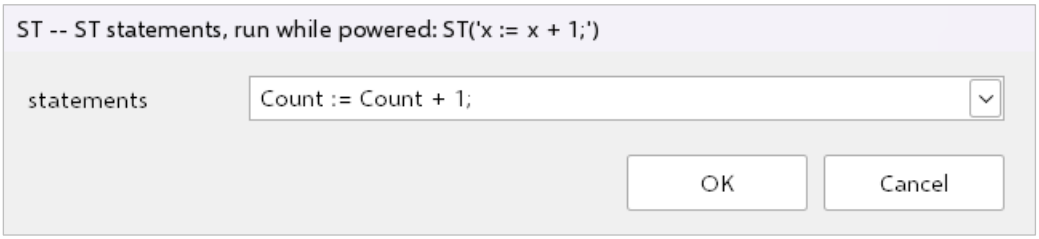


Figure 7-17 The parameter window of the ST box: enter ST statements

***Tip:** The "..." > "Function blocks" submenu contains only CALL; which function block is inserted is decided by the type of the first parameter of CALL (the instance).

7.9 Program flow

Program flow instructions change the order in which networks run, as in LDmicro 1.x. Ordinary programs rarely need them; for a process in steps, use SFC instead (section 8.4), and write logic you reuse as a FUNCTION_BLOCK.

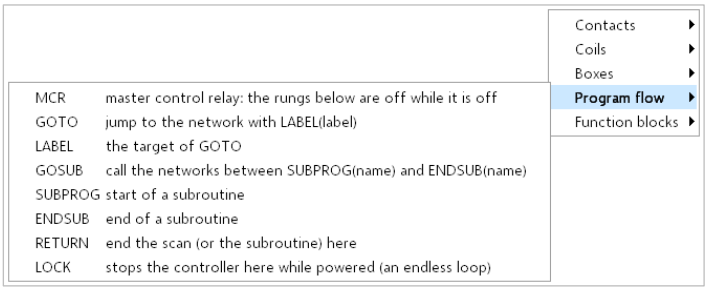


Figure 7-18 "..." > "Program flow": MCR, GOTO, LABEL, GOSUB, SUBPROG, ENDSUB, RETURN, LOCK

Instruction	.lad syntax	Key	Description
MCR	`NO(Xauto) MCR … MCR`	—	Master control relay: when the condition is OFF, all power flow in the networks below it is cut until the next `MCR` [1.x]
GOTO	`NO(Xskip) GOTO(skip)`	—	Jumps to the network that contains `LABEL(skip)` (a network number also works) [1.x]
LABEL	`LABEL(skip)`	—	The target of GOTO; goes at the start of a network [1.x]
GOSUB	`NO(Xa) GOSUB(Blink)`	—	Calls the networks between `SUBPROG(Blink)` and `ENDSUB(Blink)` [1.x]
SUBPROG	`SUBPROG(Blink)`	—	Start of a subroutine (a network of its own) [1.x]
ENDSUB	`ENDSUB(Blink)`	—	End of a subroutine (a network of its own) [1.x]
RETURN	`NO(Xb) RETURN`	—	Ends this scan here (in a subroutine: returns to the caller) [1.x]
LOCK	`NO(Xa) NO(Xb) NO(Xc) LOCK`	—	When powered, stops the controller here (an endless loop) [1.x]

```
NETWORK 'Run the next two networks only in auto mode'
    NO(Xauto) MCR
NETWORK
    NO(Xb) OUT(Ya)
NETWORK
    NO(Xc) OUT(Yb)
NETWORK 'End of master control'
    MCR
NETWORK
    NO(Xa) GOSUB(Blink)
NETWORK
    SUBPROG(Blink)
NETWORK
    NO(Xa) OUTN(Yc)
NETWORK
    ENDSUB(Blink)
```

***Safety:** When powered, `LOCK` stops the controller in an endless loop with no more scans; it is kept only for migrating 1.x programs. Do not use it in new programs.

7.10 I/O, serial port and others

This section covers the hardware-related instructions of LDmicro 1.x. In new projects, I/O usually goes directly through "I/O mapping" (section 4.6), and serial communication is best done in C/C++ user code (Chapter 14); the instructions below are mainly for migrating 1.1 programs.

- **COM port parameter**: write `COM0`, `COM1`..., which map in order to the serial ports under "Devices > COM" in the project (`Serial`, `Serial1`...). `Serial` (USB) is reserved for uploading and the online view; use `Serial1` to `Serial3` to talk to external devices.
- **Analog and PWM**: the variable of `READ_ADC` must be mapped to an analog input in the I/O mapping; the pin parameter of `SET_PWM` is a variable mapped to `Pin.n` (an Arduino pin) in the I/O mapping.

Instruction	.lad syntax	Key	Description
READ_ADC	`NO(Xa) READ_ADC(Nadc)`	—	Reads an analog input into the variable (the variable must be mapped to an analog input in the I/O mapping, for example `Pin.n`) [1.x]
SET_PWM	`NO(Xa) SET_PWM(Nduty, 1000, Npwm)`	—	PWM output; parameters: duty cycle, frequency, pin (the variable mapped to `Pin.n`), resolution (optional) [1.x]
PERSIST	`NO(Xa) PERSIST(Count)`	—	Keeps the variable in EEPROM: read back in the first scan after power-on, then written when the value changes [1.x]
UART_SEND	`PT(Xb) UART_SEND(Val, COM1)`	—	Sends a variable from a serial port; parameters: variable, COM, number of bytes (optional), wait (optional) [1.x]
UART_RECV	`UART_RECV_AVAIL(COM1)` `UART_RECV(Val, COM1)`	—	Receives one byte from a serial port into the variable (usually after UART_RECV_AVAIL); parameters: variable, COM, number of bytes (optional), wait (optional) [1.x]
UART_SEND_READY	`UART_SEND_READY(COM1)`	—	Contact: on when the serial port can send [1.x]
UART_RECV_AVAIL	`UART_RECV_AVAIL(COM1)`	—	Contact: on when the serial port has data [1.x]
FMTSTR	`PT(Xc) FMTSTR(Count, 'Count = \5\r\n', COM1)`	—	On a rising edge of the power flow, sends a formatted string to the serial port (one character per scan); `\1` to `\5` give the number of digits of the variable, `\-5` includes the sign [1.x]
STRING	`NO(Xa) STRING(sMsg, Count, 'N=%d')`	—	Stores a formatted string in a STRING variable (C printf format) [1.x]
DELAY	`NO(Xa) DELAY(10)`	—	Busy wait (microseconds); lengthens the scan time [1.x]

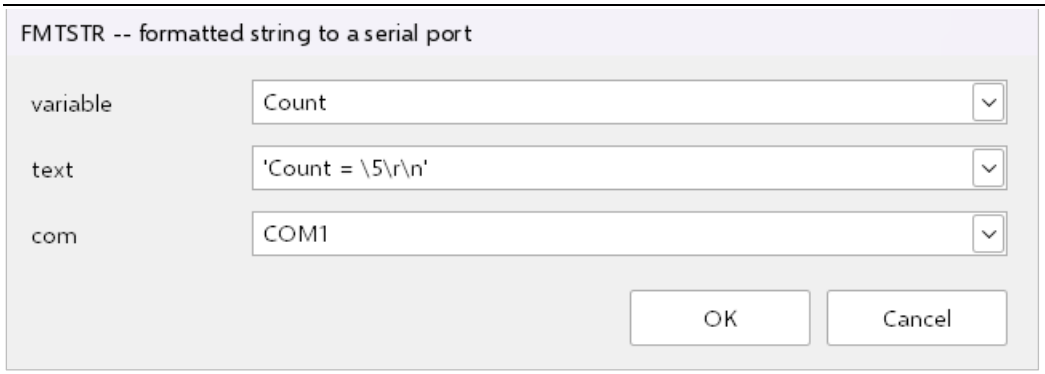


Figure 7-19 The parameter window of FMTSTR: variable, text, COM

*__Note:__ Writes by `PERSIST` to EEPROM (flash memory) wear it out; use it only for settings that rarely change. Declare ordinary variables that must survive a power loss in `VAR_GLOBAL RETAIN`.
`DELAY` busy-waits within the scan, which lengthens the scan time and can trigger the watchdog (section 10.5).

7.11 Motion control instructions

There are two ways to do motion control in a ladder diagram:

- **PLCopen MC and Motion86 function blocks** (recommended): `MC_Power`, `MC_MoveAbsolute`, `MC_Home`... and the machine blocks `M86_*`, inserted with **CALL**, with the power flow connected to Enable / Execute. Fill `Axis :=` with the axis name and `Machine :=` with the machine name (Chapter 12).
- **The 1.x MC element**: `MC(function, axis, a1, a2, a3, a4, done, busy, error)`, for programs migrated from LDmicro 1.1; the axis is a variable bound to an 86EVA servo, and unused parameters are filled with `-`.

```
VAR
    pwrX  : MC_Power;
    homeX : MC_Home;
    mvX   : MC_MoveAbsolute;
END_VAR
NETWORK 'Power on'
    NO(Xenable) CALL(pwrX, Axis := AxisX, Enable := @, Status => @) OUT(RpowerOk)
NETWORK 'Homing'
    NO(RpowerOk) NO(Xhome) CALL(homeX, Axis := AxisX, Execute := @, Done => @) OUT(RhomeDone)
NETWORK 'Move to 100 mm'
    NO(RpowerOk) NO(Xgo) CALL(mvX, Axis := AxisX, Execute := @, Position := 100.0, Velocity := 50.0,
Done => @) OUT(RmoveDone)
```

Instruction	.lad syntax	Key	Description
CALL (MC_* / M86_*)	`NO(Xgo) CALL(mvX, Axis := AxisX, Execute := @, Position := 100.0, Velocity := 50.0, Done => @)`	K	PLCopen / Motion86 function blocks; for the list of function blocks and their parameters, see sections 12.7 and 12.8
MC	`PT(Xgo) MC(MC_MoveAbsolute, Naxis, 100, 50, -, -, Rdone, Rbusy, Rerr)`	—	PLCopen motion box for CiA 402 servos (LDmicro 1.x); the functions are MC_Power, MC_MoveAbsolute, MC_MoveRelative, MC_MoveVelocity, MC_Home, MC_Stop, MC_Halt, MC_Reset, MC_ReadActualPosition, MC_ReadStatus, MC_ReadAxisError; fill unused parameters with - [1.x]

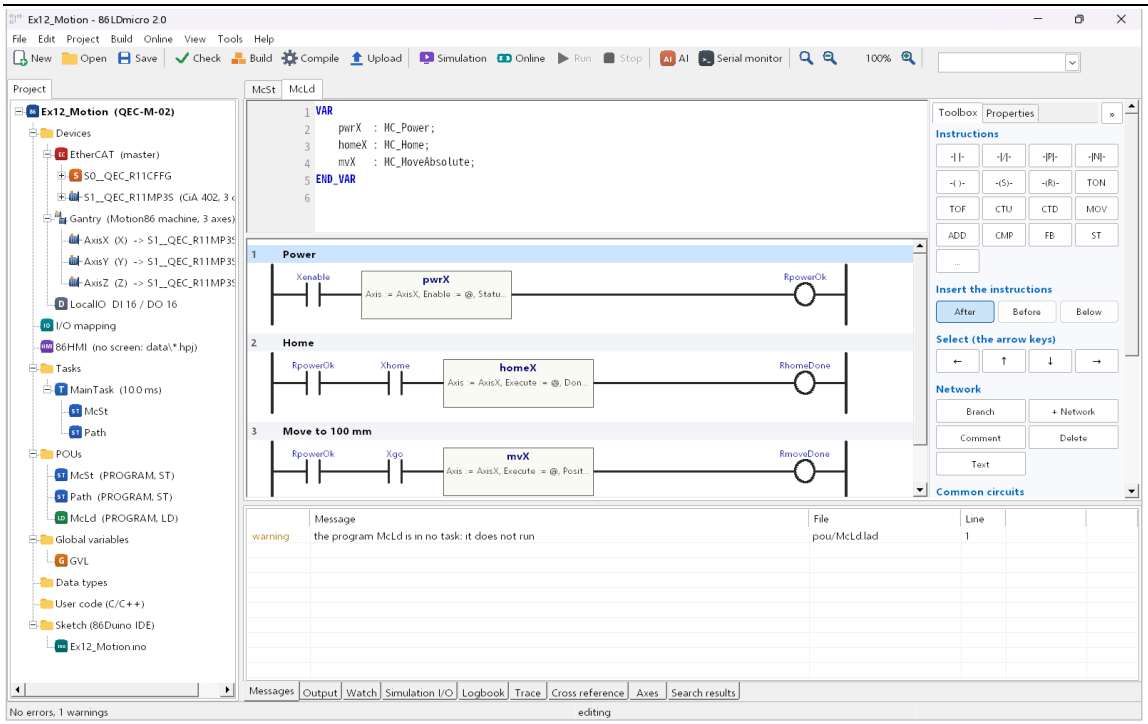


Figure 7-20 PLCopen function blocks in a ladder diagram: MC_Power, MC_Home, MC_MoveAbsolute

*__Safety:__ Execute-type function blocks (MoveAbsolute, Home...) send their command on the **rising edge** of Execute; to send it again, let the power flow go off first.
The function block instances must be called in every scan (do not let MCR or GOTO skip the network), or Done / Busy will not be updated. For the first trial run, first check the direction and limits at low speed with "Online" on the machine page and the axis page (sections 12.5 and 12.6).

Chapter 8 ST and SFC

This chapter describes the editors and syntax of Structured Text (ST) and Sequential Function Chart (SFC). ST suits calculations, comparisons, loops and data processing; SFC suits processes with steps in sequence. Both are IEC 61131-3 languages and can be placed in the same project, and in the same task, as ladder diagrams (LD), running in order. For complete examples, see section 6.5 (ST: filling control) and section 6.6 (SFC: sequence control).

8.1 The ST editor

Create an ST program with "Project > Add a PROGRAM (ST)...", or by right-clicking POU in the project tree. Enter a name and click **OK**: the file `pou/<name>.st` opens at once, and the new PROGRAM is added to the first task automatically.

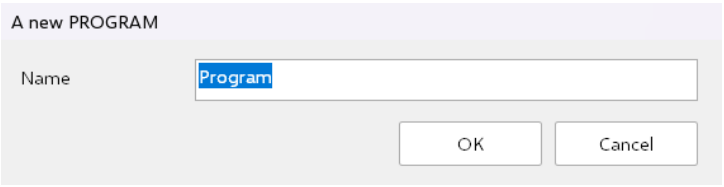


Figure 8-1 Add a PROGRAM (ST): enter the program name

The ST editor has two areas: on the left, the program text (the declaration section VAR … END_VAR and the program body are in the same file); in the side panel, **ST snippets** in the "Toolbox".

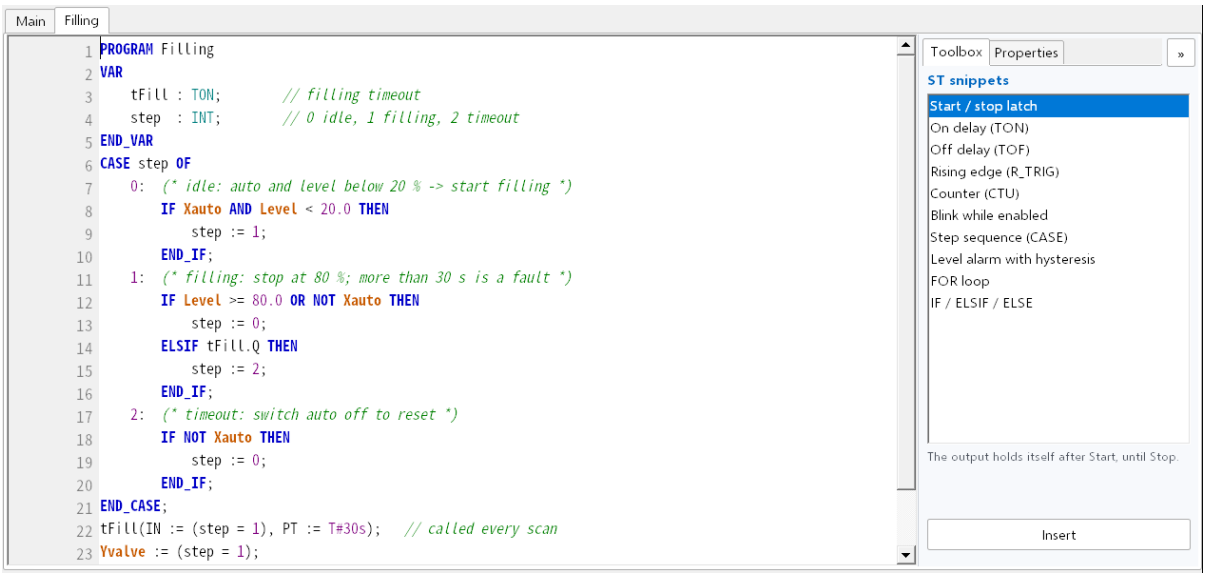


Figure 8-2 The ST editor: the program text on the left, "ST snippets" on the right

8.1.1 Syntax colors

The editor colors the code by syntax, so typing mistakes are easy to spot:

Color	Content	Example
Blue bold	Keywords	`IF`, `THEN`, `END_IF`, `VAR`, `CASE`
Teal	Data types	`BOOL`, `INT`, `REAL`, `TON`
Green italic	Comments	`(* … *)`, `// …`
Dark red	Strings	`'G1 X10 Y10'`
Purple	Numbers, time constants	`100`, `16#FF`, `T#2s`
Black	Variable names and others	`Level`, `tFill`

Errors are underlined; hover the mouse over the underline to see the error description. After you click **Check** (Shift+F7), the errors are also listed in the "Messages" window (section 9.6).

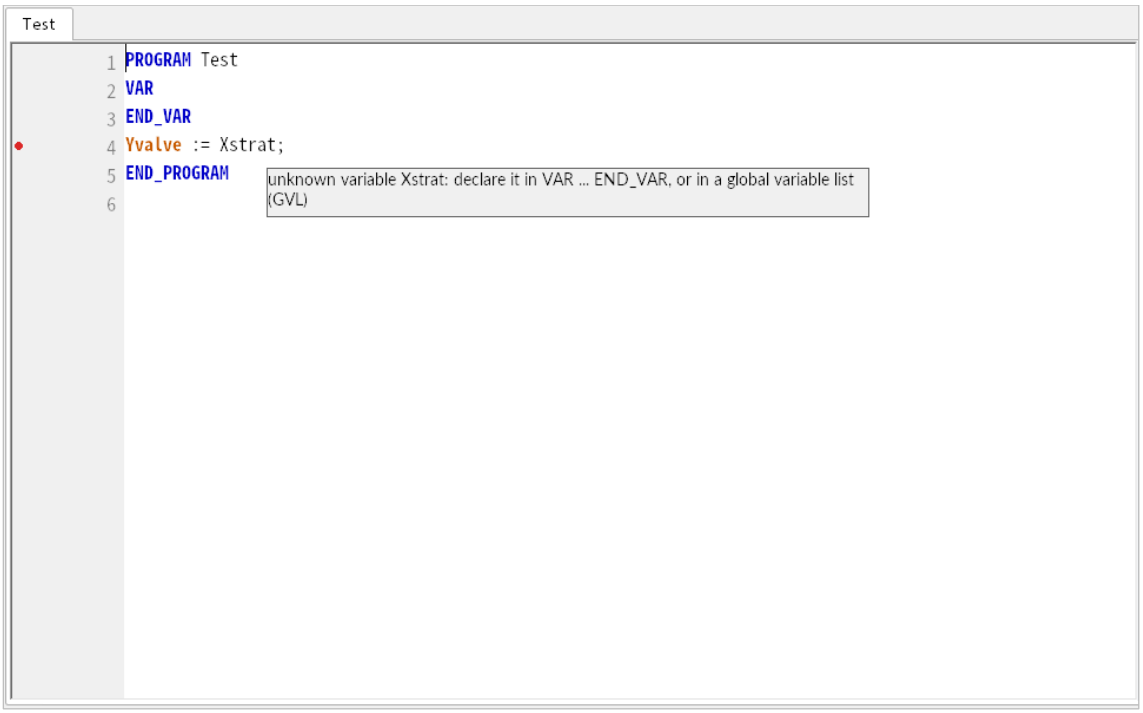


Figure 8-3 Error underline: hover the mouse to see the error description

8.1.2 Completion and parameter hints

As you type a variable or keyword, the completion list opens automatically, listing the matching names with their type, origin (GVL or local variable) and comment:

- It opens automatically when you type the **first two letters** of a name; it also opens when you type `.` (for example `tFill.` lists members such as Q and ET), `#` or `(`.
- Choose with ↑ ↓ (or PageUp / PageDown), insert with **Enter** or **Tab**, close with **Esc** (it then does not open again automatically for the same word).
- When you type the `(` or `,` of a function block or function, its parameter hint is shown (for example `TON(IN, PT)`); it disappears when you type `)` or `;`.
- Completion does not open inside comments and strings.

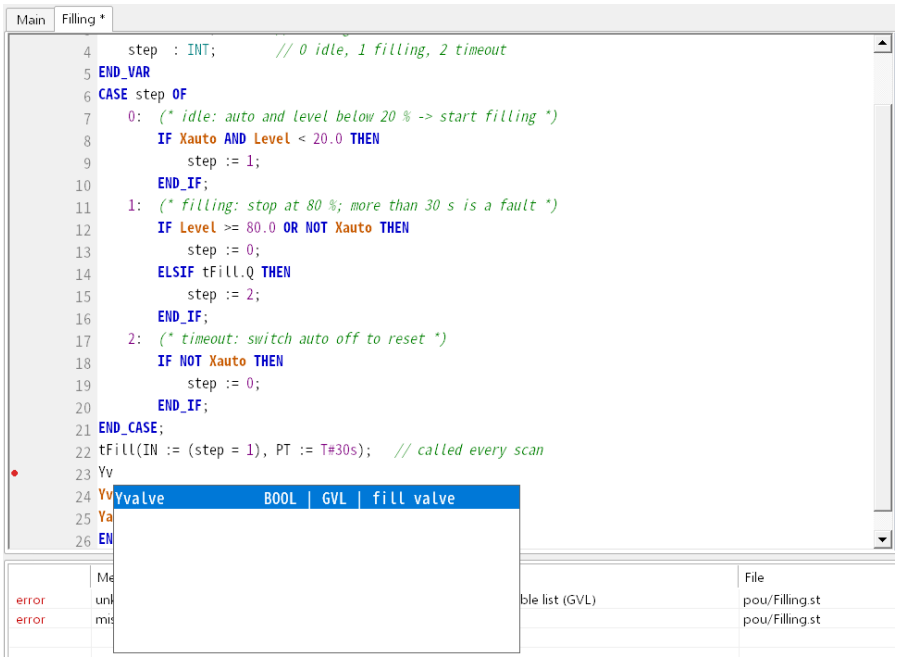


Figure 8-4 Completion: typing Yv lists Yvalve (BOOL, GVL, comment)

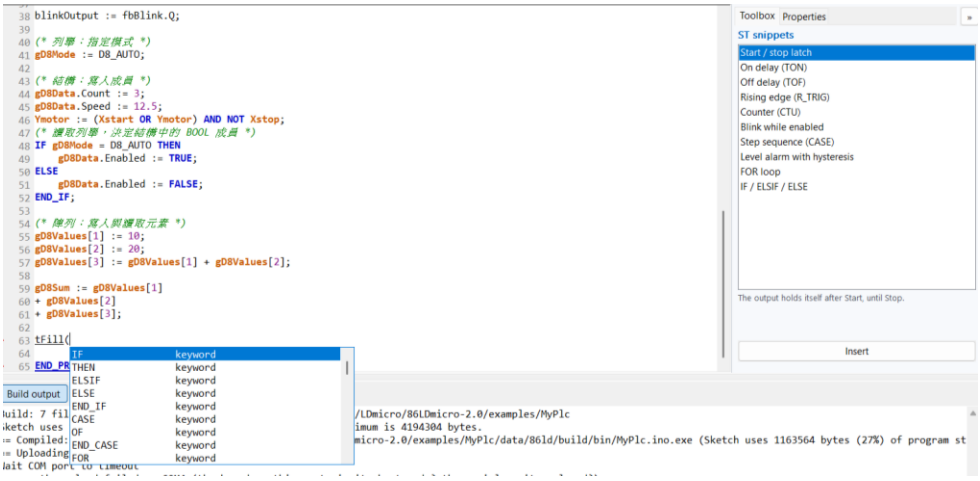


Figure 8-5 Parameter hint: typing tFill(shows the inputs of TON

> ***Note:** In 86LDmicro 2.0.0, **Ctrl+Space does not open the completion list**; just type the first two letters of the name (or `.`) and the list appears automatically.

8.1.3 ST snippets

ST snippets in the side panel offers templates of common code. Select a snippet (its description appears below), click **Insert** (or double-click) to open the form, fill in the variable names and click **OK**: the code is inserted at the cursor. The function block instances the snippet needs (for example TON) are added to the program's VAR automatically, and new global variables are added to the GVL automatically.

Snippet	Use
Start / stop latch	Press start and the output latches; press stop to release it
On delay (TON)	The output turns ON once the input has stayed ON for the set time
Off delay (TOF)	After the input goes OFF, the output stays on for the set time
Rising edge (R_TRIG)	Runs once each time the input goes from OFF to ON
Counter (CTU)	Counts rising edges, turns the output ON at the set value, can be reset
Blink while enabled	While enabled, the output toggles every half period
Step sequence (CASE)	A state machine: moves to the next step when each step is done
Level alarm with hysteresis	ON above the upper limit, OFF only below the lower limit
FOR loop	Repeats with i from 1 to N
IF / ELSIF / ELSE	Conditional statements

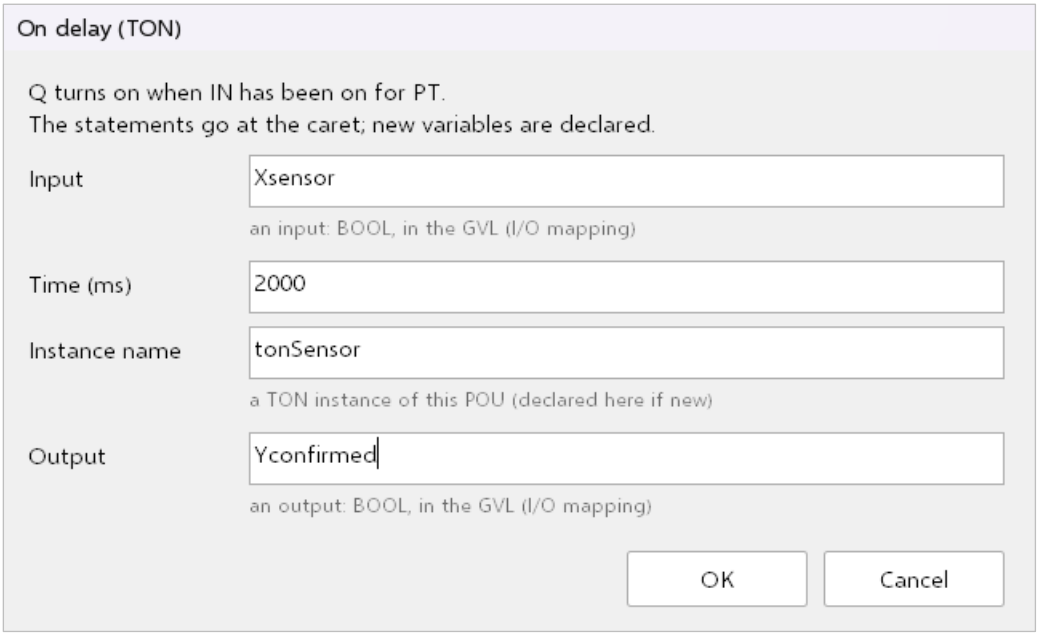


Figure 8-6 Inserting the "On delay (TON)" snippet: fill in the input, time, instance name and output

8.1.4 Auto format, value display and other operations

Action	Key	Description
Auto format	Ctrl+T	Re-indents VAR, IF / ELSIF / ELSE, CASE, FOR / WHILE / REPEAT; the status bar shows "Formatted (Ctrl+T); Ctrl+Z takes it back"
Indent / unindent	Tab / Shift+Tab	Inserts 4 spaces; Shift+Tab removes up to 4 spaces at the start of the line
Auto indent	Enter	A new line keeps the indentation of the previous line; one more level after THEN, DO, OF and the like
Undo / redo	Ctrl+Z / Ctrl+Y	—
Find	Ctrl+F, F3, Shift+F3	Find in this tab (section 3.7)
Variable menu	Right-click a variable	Add to watch, Add to the trace, "Where is <variable> used?", Its declaration, I/O mapping, 86HMI binding; in simulation or online, also Write, Force

Hovering the mouse over a variable shows its type; **in simulation or online** it also shows the current value, so you do not need to add the variable to the watch.

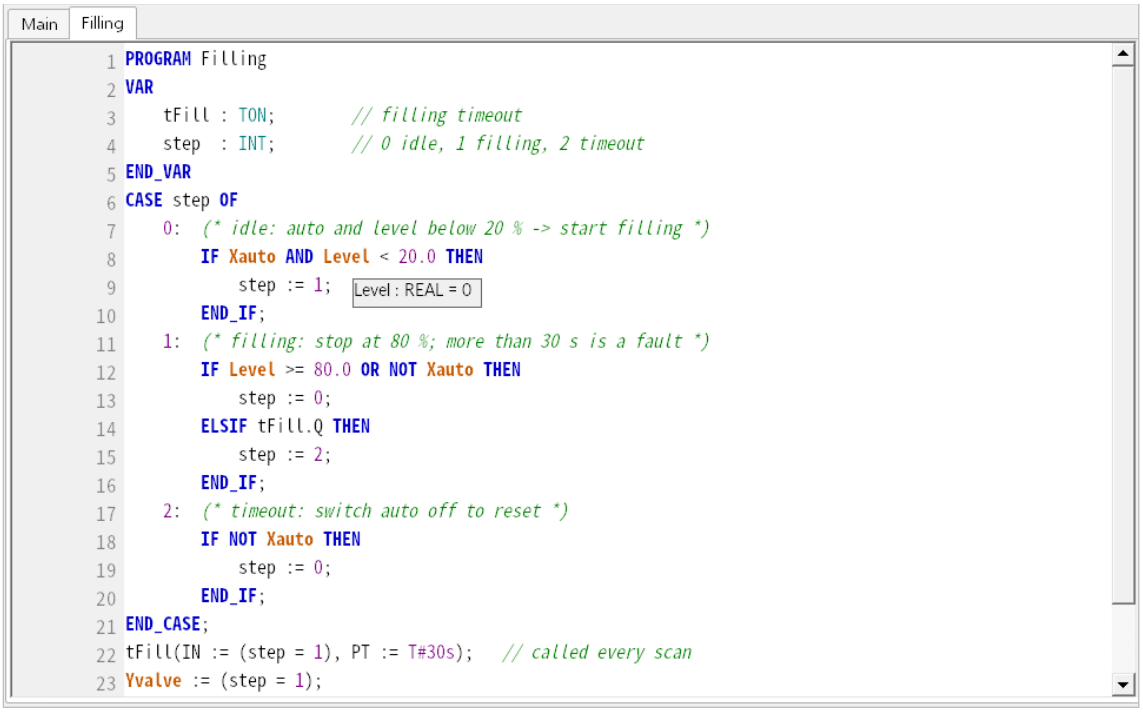


Figure 8-7 In simulation: hovering over a variable shows its type and current value

8.2 ST syntax quick reference

The tables below summarize the common ST syntax of 86LDmicro (IEC 61131-3). All of it has been confirmed in 86LDmicro 2.0.0; the entries are fragments, so to paste one in and check it, put it in a program that declares the variables involved (for example Recipe and Axis1 from section 4.5).

8.2.1 Program structure and declarations

```
PROGRAM Name                    // or FUNCTION_BLOCK Name, FUNCTION Name : Type
VAR                             // Local variables (in a PROGRAM or FUNCTION_BLOCK the values are kept
until the next scan)
    tFill : TON;                // Function block instance
    step  : INT := 0;           // An initial value can be given
    buf   : ARRAY[0..9] OF INT; // Array
    s     : STRING[40];         // String (maximum length)
END_VAR
VAR CONSTANT
    MAXN : INT := 10;           // Constant
END_VAR
(* Program body *)
END_PROGRAM
```

Section	Used in	Description
`VAR … END_VAR`	All	Local variables
`VAR CONSTANT`	All	Constants, cannot be written
`VAR_TEMP`	All	Temporary variables: start from their initial value on every call and are not kept until the next
`VAR_INPUT`	FUNCTION, FUNCTION_BLOCK	Input parameters
`VAR_OUTPUT`	FUNCTION, FUNCTION_BLOCK	Outputs (function block: read as instance.name after the call; function: retrieved in the call with name => variable)
`VAR_IN_OUT`	FUNCTION, FUNCTION_BLOCK	The variable passed in itself, which can be modified directly; it must be a local variable of the calling ST program (not a constant, an expression or a global variable)
`VAR_GLOBAL`	Global variable list (GVL)	Variables shared by the whole project (section 4.5)

A PROGRAM cannot declare VAR_INPUT, VAR_OUTPUT or VAR_IN_OUT (the check gives an error). The table above is what 86LDmicro 2.0.0 actually supports.

8.2.2 Statements

Item	Syntax
Assignment	`x := y + 1;`
Comment	`(* multi-line *)`, `// single line`
Condition	`IF a THEN ... ELSIF b THEN ... ELSE ... END_IF;`
Multiple choice	`CASE step OF 0: ...; 1, 2: ...; 10..20: ...; ELSE ... END_CASE;`
FOR loop	`FOR i := 1 TO 10 BY 2 DO ... END_FOR;`
WHILE loop	`WHILE n > 0 DO ... END_WHILE;`

Item	Syntax
REPEAT loop	`REPEAT ... UNTIL n >= 3 END_REPEAT;`
Leave a loop	`EXIT;`
End this run	`RETURN;` (the PROGRAM does not run any further in this scan)
Call a function block	`t1(IN := start, PT := T#2s);  IF t1.Q THEN ... END_IF;`
Call a function	`y := LIMIT(0.0, x, 100.0);  n := REAL_TO_INT(r);`
Array, structure	`buf[i] := 0;  Recipe[2] := 10;  Axis1.pos := 10.0;`

8.2.3 Operators and constants

Category	Syntax
Arithmetic	`+  -  *  /  MOD` (an integer division by 0 gives 0)
Comparison	`=  <>  <  <=  >  >=`
Logic (BOOL) and bitwise (WORD etc.)	`AND  OR  XOR  NOT`
Integer	`100`, `-5`, `16#FF` (hexadecimal), `2#1010` (binary)
Real	`1.5`, `1.5E3`
Time	`T#2s`, `T#1s500ms`, `T#100ms`
Boolean, string	`TRUE`, `FALSE`, `'text'`

8.2.4 Common functions and function blocks

Category	Names
Math	ABS, SQRT, SIN, COS, EXPT, TRUNC
Selection and limits	MIN, MAX, LIMIT, SEL, MUX
Bits	SHL, SHR, ROL, ROR
Strings	LEN, CONCAT, LEFT, RIGHT, MID, FIND
Type conversion	`<type>_TO_<type>`, for example `INT_TO_REAL`, `REAL_TO_DINT`, `INT_TO_STRING`, `STRING_TO_REAL`
Timers	TON (on delay), TOF (off delay), TP (pulse)
Counters	CTU, CTD, CTUD
Edges	R_TRIG (rising edge), F_TRIG (falling edge)
Flip-flops	SR (set dominant), RS (reset dominant)
Motion control	MC_* (PLCopen, section 12.7), M86_* (Motion86 machines, section 12.8)

*__Tip:__ A function block instance (for example TON) must be **called in every scan** for a timer to accumulate time. If you put the call inside an IF or CASE, it is not updated in scans where the condition is false; the robust way is to call it outside the IF / CASE and use the condition as an input, for example `tFill(IN := (step = 1), PT := T#30s);`.

8.3 Custom FUNCTIONs and FUNCTION_BLOCKs

Calculations you reuse can be written as your own FUNCTION or FUNCTION_BLOCK and called from ST, LD and SFC, just like the built-in functions and function blocks:

Kind	Memory	How to create	How to use
FUNCTION	No instance state: each call computes from scratch; suits calculations and conversions (when it computes only from its parameters, the same parameters give the same result)	"Project > Add a FUNCTION (ST)..."	`y := Scale(raw, 0.0, 100.0);`
FUNCTION_BLOCK	Yes: each instance keeps its own internal variables (for example timers)	"Project > Add a FUNCTION_BLOCK (ST)..." or "Add a FUNCTION_BLOCK (ladder)..."	First declare an instance `bl : Blinker;`, then call `bl(...)`

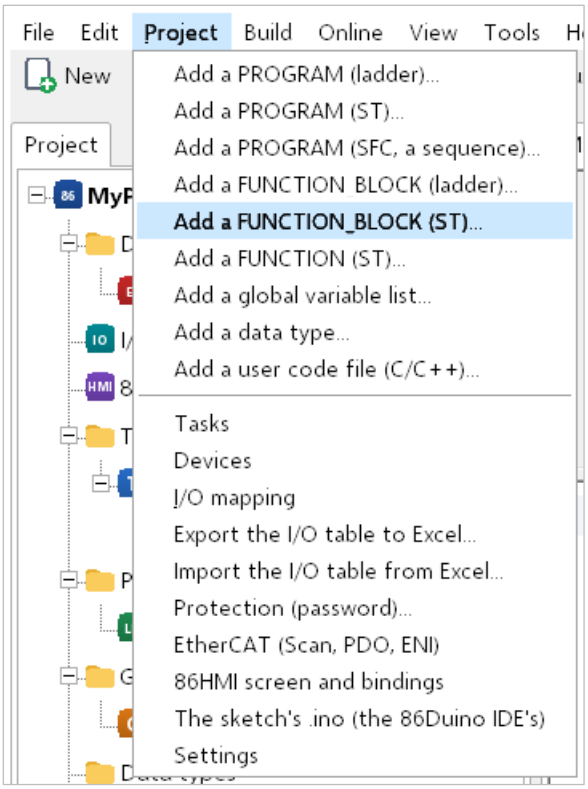


Figure 8-8 The "Project" menu: Add a FUNCTION_BLOCK (ST) and Add a FUNCTION (ST)

8.3.1 Example: analog scaling and blinking

Scenario: The raw value of an analog input, `AnalogIn` (0 to 4095), is to be converted into the level percentage `Level`; while running (`Rrun`), the lamp `Yblink` turns on and off every 0.5 s. The scaling is written as the FUNCTION `Scale` and the blinking as the FUNCTION_BLOCK `Blinker`, both called by the ST program `Panel`.

Variable	Type	I/O mapping	Use
Rrun	BOOL	— (written by another program)	Running
AnalogIn	INT	Analog input (section 4.6)	Raw analog value 0 to 4095
Level	REAL	—	Level (%)
Yblink	BOOL	Digital output	Blinking lamp

FUNCTION Scale (`pou/Scale.st`): the function name is the return value; assign to it inside the function.

```
FUNCTION Scale : REAL                  // Analog scaling: 0–4095 → lo–hi
VAR_INPUT
    raw : INT;
    lo, hi : REAL;
END_VAR
Scale := lo + (hi - lo) * INT_TO_REAL(raw) / 4095.0;
END_FUNCTION
```

FUNCTION_BLOCK Blinker (`pou/Blinker.st`): the internal TON and the output `Out` are kept separately in each instance.

```
FUNCTION_BLOCK Blinker                 // Blink: a function block with memory
VAR_INPUT
    Enable : BOOL;
    Half   : TIME;
END_VAR
VAR_OUTPUT
    Out : BOOL;
END_VAR
VAR
    t : TON;
END_VAR
t(IN := Enable AND NOT t.Q, PT := Half);   // Toggles every half period
IF t.Q THEN
    Out := NOT Out;
END_IF;
IF NOT Enable THEN
    Out := FALSE;
END_IF;
END_FUNCTION_BLOCK
```

PROGRAM Panel (`pou/Panel.st`, run in a task):

```
PROGRAM Panel
VAR
    bl : Blinker;                       // Function block instance
END_VAR
Level := Scale(AnalogIn, 0.0, 100.0);  // 0–4095 → 0–100 %
bl(Enable := Rrun, Half := T#500ms);
Yblink := bl.Out;
END_PROGRAM
```

The same function block can also be used in a ladder diagram with **CALL** (section 7.8). The network below declares another instance, `bl2`, connects the power flow to Enable, and Out drives the coil `Ylamp2`:

```
PROGRAM Main
VAR
    bl2 : Blinker;
END_VAR
NETWORK 'Custom function block: blink'
    NO(Rrun) CALL(bl2, Enable := @, Half := T#250ms, Out => @) OUT(Ylamp2)
END_PROGRAM
```

#	Action
1	"Project > Add a FUNCTION (ST)...", name `Scale`; enter the program above.
2	"Project > Add a FUNCTION_BLOCK (ST)...", name `Blinker`; enter the program above.
3	Declare Rrun, AnalogIn, Level and Yblink (Ylamp2) in the GVL.
4	"Project > Add a PROGRAM (ST)...", name `Panel`; enter the program above (it is added to the first task automatically).
5	Click **Check** (Shift+F7) and make sure "Messages" shows no errors.
6	Click **Simulate** (F5) and add Rrun, AnalogIn, Level and Yblink to the Watch window; write 2048 to AnalogIn and TRUE to Rrun.

Expected result: With AnalogIn = 2048, Level is about 50.01; while Rrun is TRUE, Yblink toggles about every 0.5 s (Ylamp2 in the ladder diagram toggles about every 0.25 s); while Rrun is FALSE, both are off.

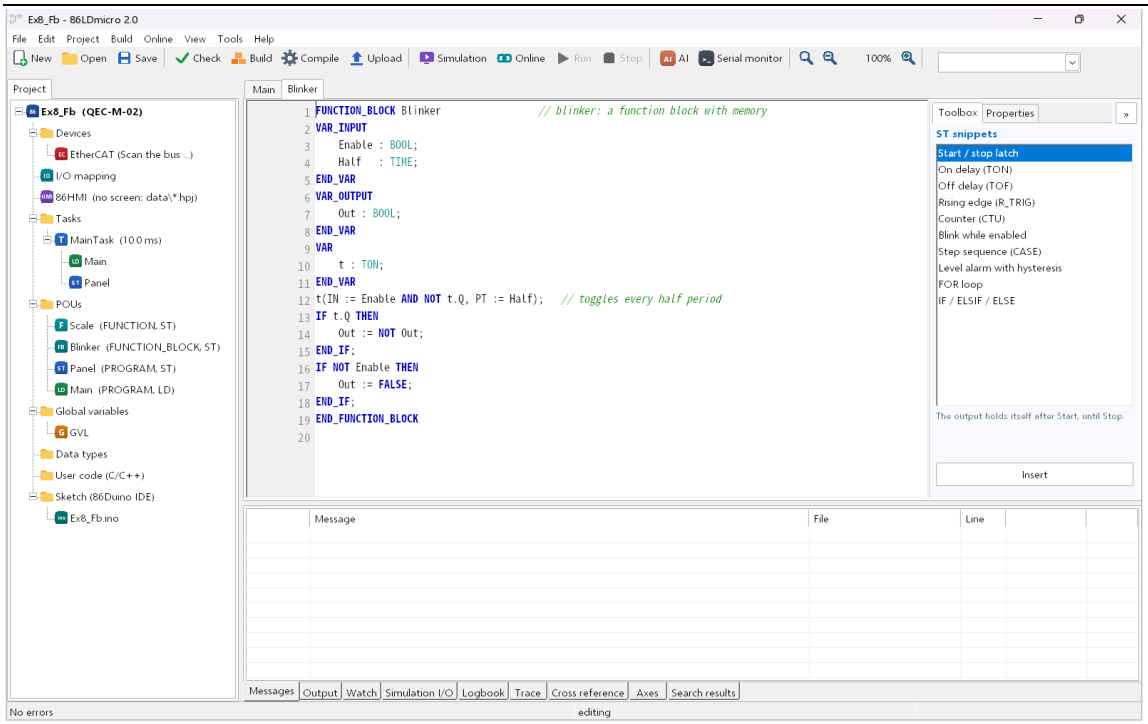


Figure 8-9 The editor of FUNCTION_BLOCK Blinker

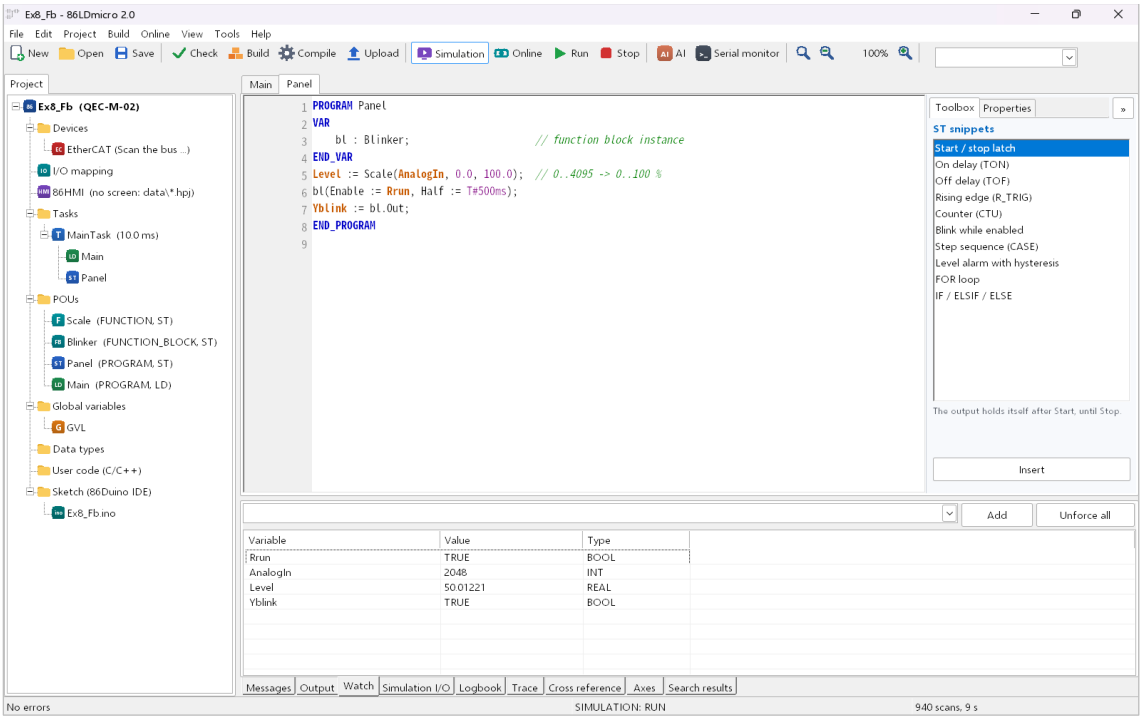


Figure 8-10 In simulation: Level is 50.01 and Yblink blinks

*__Tip:__ The local variables of a FUNCTION start from their initial values on every call (they are not kept); for anything with memory, such as timers, counters or edge detection, write a FUNCTION_BLOCK. FUNCTIONs and FUNCTION_BLOCKs are not placed in tasks; the program that calls them decides when they run.

8.4 The SFC editor

SFC describes a process with **steps** and **transitions**: only the active steps run their actions, and when a transition's condition holds, the chart moves to the next step. It suits automatic machine sequences, recipe procedures and the like.

Create it with "Project > Add a PROGRAM (SFC, a sequence)..." or by right-clicking POU in the project tree; the file is `pou/<name>.sfc`, and like LD and ST it runs when placed in a task. A new SFC already has a simple chart (initial step Init → step Step1 → back to Init after 1 s) that you can modify directly.

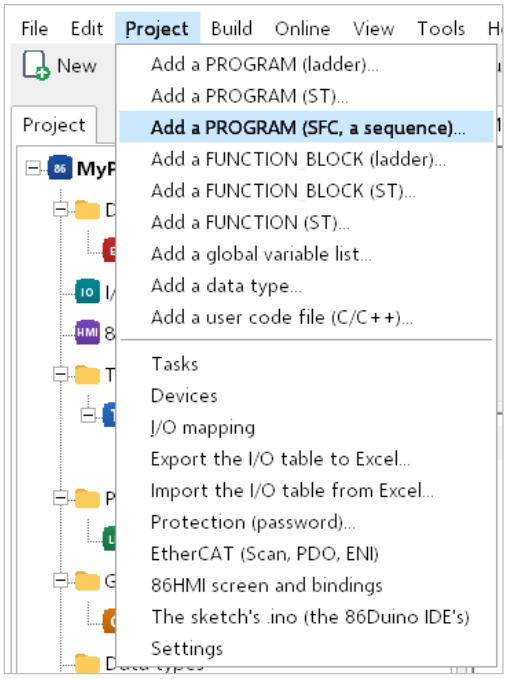


Figure 8-11 "Project > Add a PROGRAM (SFC, a sequence)..."

The SFC editor has three areas: the **declaration section** (VAR … END_VAR) at the top, the **chart** in the middle, and the "**Chart**" tool buttons in the side panel.

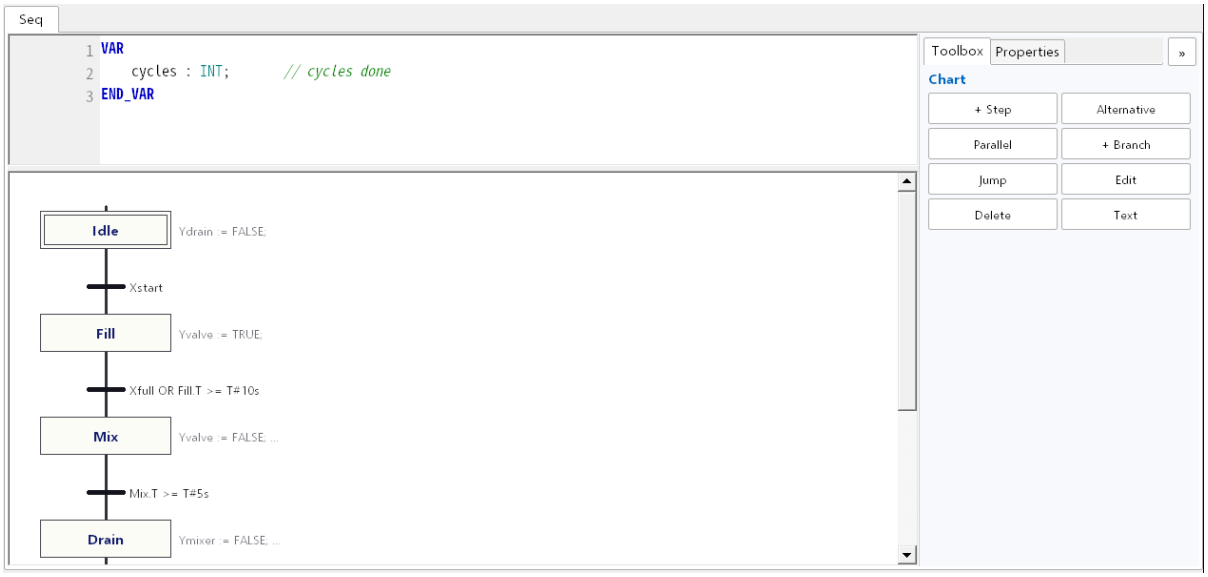


Figure 8-12 The SFC editor: the declaration section at the top, the chart in the middle, the "Chart" tools on the right

8.4.1 The "Chart" tools

Button	Key	Function
+ Step	Ins	Adds a transition and a step after the selection
Alternative	—	Click with a step selected: branches after the step, from left to right; the first transition that holds takes effect
Parallel	—	Click with a transition selected: all branches run at the same time and join at the next transition
+ Branch	—	Adds one more branch to the selected alternative / parallel branch
Jump	—	Goes after the last transition of a branch or of the chart: jumps back (or on) to a step
Edit	Enter, double-click	Edits the step's name and actions, or the transition's condition
Delete	Del	Deletes the selection (a step together with its transition)
Text	Ctrl+T	Views and edits the chart as .sfc text; the syntax is checked when you click "Back to the diagram"

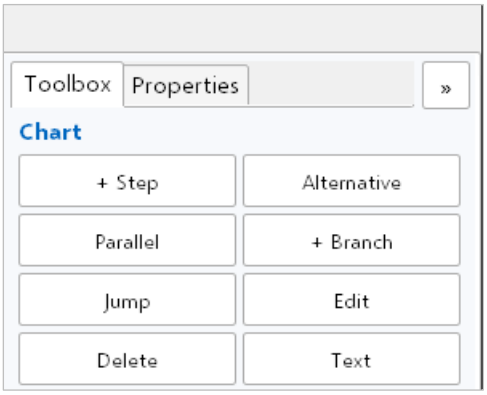


Figure 8-13 The "Chart" tool buttons in the side panel

Right-clicking the chart gives the same commands. Other operations:

- Arrow keys: ↑ ↓ select the previous / next item in the same sequence, ← → select the neighboring branch.
- Ctrl+Z / Ctrl+Y: undo / redo.
- The chart cannot be changed in simulation or online (stop the simulation or disconnect first).
- When the project is password protected, SFC is read-only (section 15.6).

8.4.2 Text mode

Click **Text** (Ctrl+T) to edit the whole chart directly as .sfc text (for the format, see 8.5.4), which suits large changes or copying from another SFC; when you click **Back to the diagram**, the syntax is checked, and if there are errors you are told and stay in text mode.

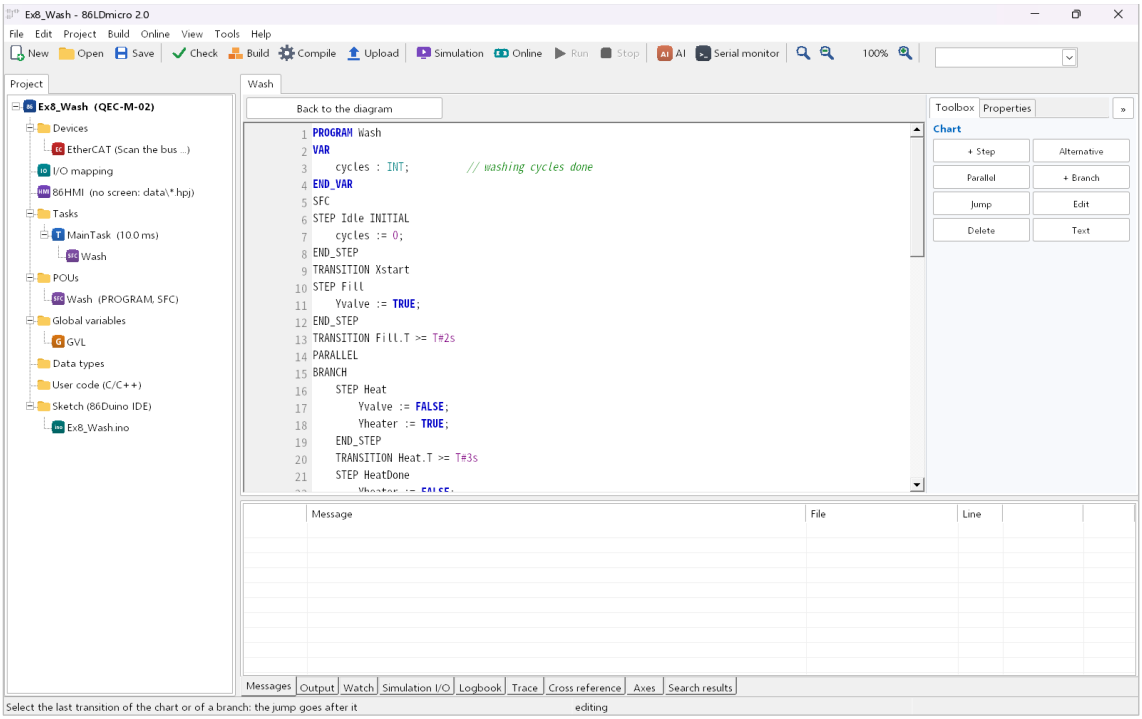


Figure 8-14 SFC text mode: the whole chart as .sfc text

8.5 Steps, transitions and branches

8.5.1 Steps

Select a step and click **Edit** (Enter or double-click) to open the "Step" window:

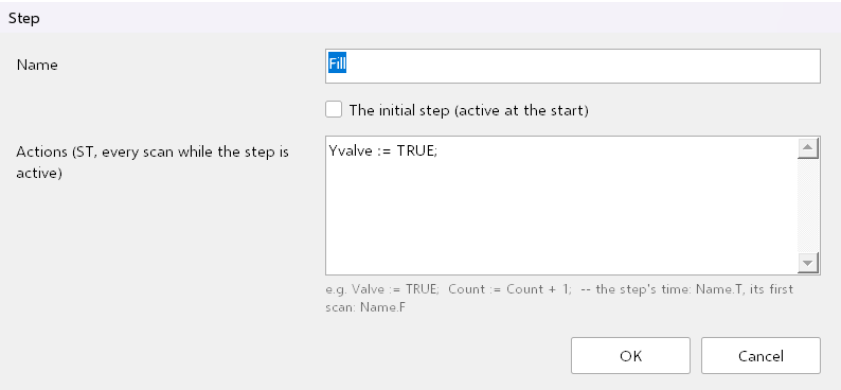


Figure 8-15 Step: name, initial step, actions

Field	Description
Name	Letters, digits and _; when you rename a step, the jumps (JUMP) to it are renamed too
The initial step (active at the start)	The step active when the PLC starts running, shown with a double frame; a chart can have only one
Actions (ST, every scan while the step is active)	ST statements, for example `Valve := TRUE;  Count := Count + 1;`; the first line of the actions is shown to the right of the step in the chart

8.5.2 Transitions

Select a transition and click **Edit** to open the "Transition" window. The condition is an ST expression: `TRUE`, a BOOL variable or any expression whose result is a BOOL:

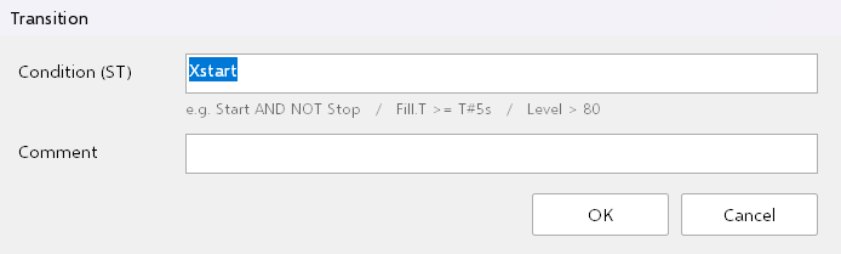


Figure 8-16 Transition: condition (ST) and comment

Condition	Meaning
`Start AND NOT Stop`	Start pressed and stop not pressed
`Fill.T >= T#5s`	Step Fill has been active for 5 s
`Level > 80`	Level above 80
`mv.Done`	A function block (for example MC_MoveAbsolute) is done
`TRUE`	Transition at once (in the next scan)

8.5.3 Branches and jumps

Structure	How to create	How it runs
Alternative branch (ALTERNATIVE)	Select a step → **Alternative**; each branch starts with a transition; **+ Branch** adds one more	Checked from left to right: the branch whose transition is the **first to hold** is taken (the left has priority)
Parallel branch (PARALLEL)	Select a transition → **Parallel**; each branch starts with a step; **+ Branch** adds one more	All branches start **at the same time**; they join when the last step of every branch is active and the next transition holds
Jump (JUMP)	Select the last transition of a branch or of the chart → **Jump...**, and choose the step to jump to	When the transition holds, jumps to the given step (it can go back to form a loop)

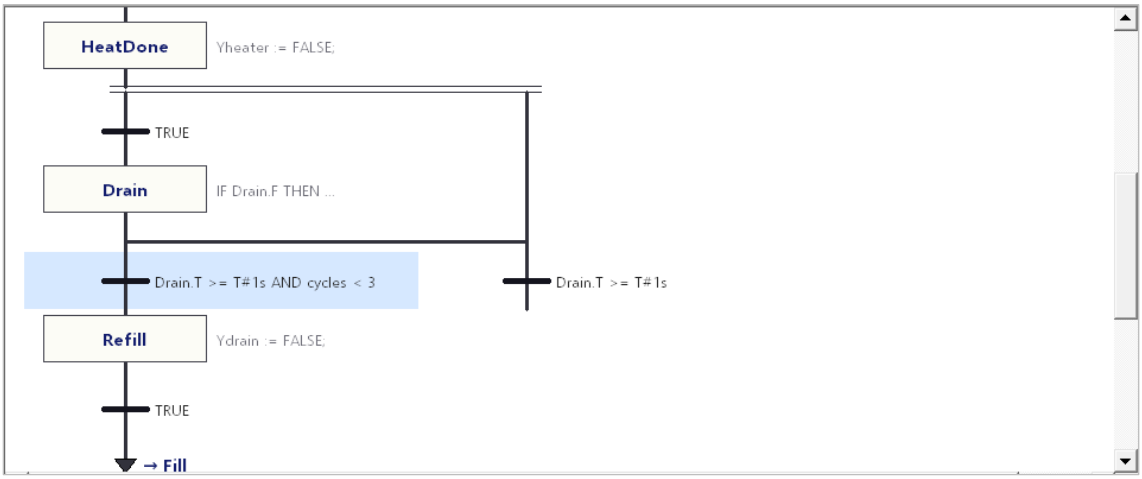


Figure 8-17 Alternative branch: two paths after Drain, the left back to Fill, the right on to Done

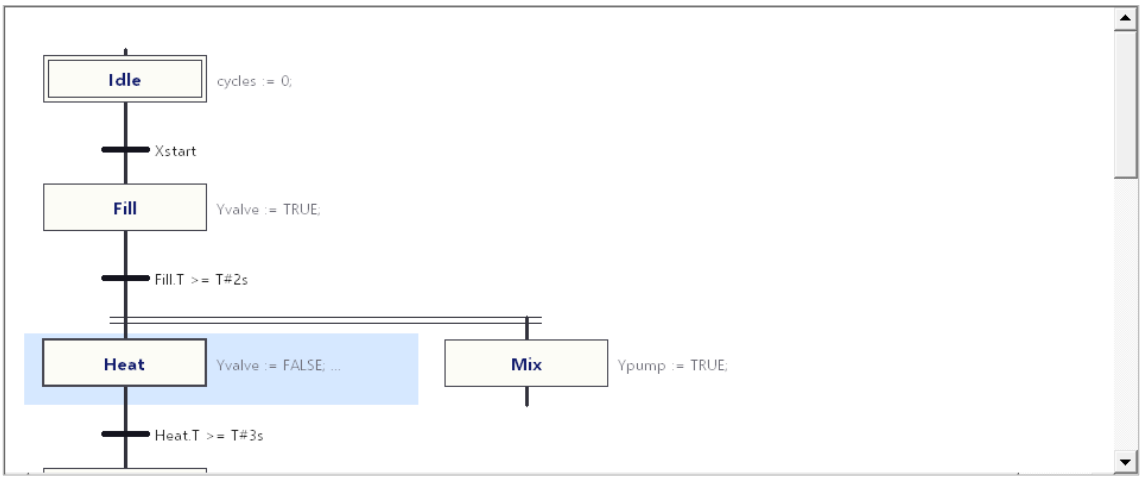


Figure 8-18 Parallel branch: Heat and Mix run at the same time and join at a TRUE transition

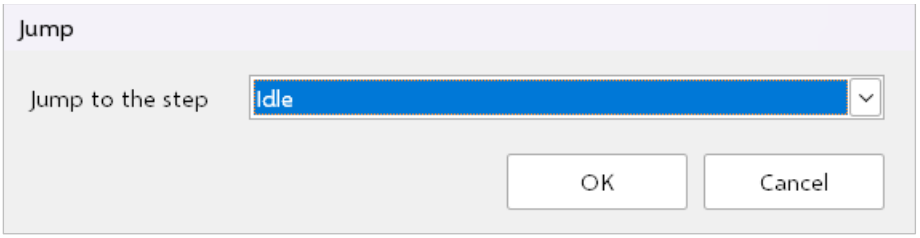


Figure 8-19 Jump: choose the step to jump to

When the structure is wrong (for example two steps connected directly, or a parallel branch starting with a transition), the errors are listed in the "Messages" window below (the error, file and line number); double-click one to go to that line.

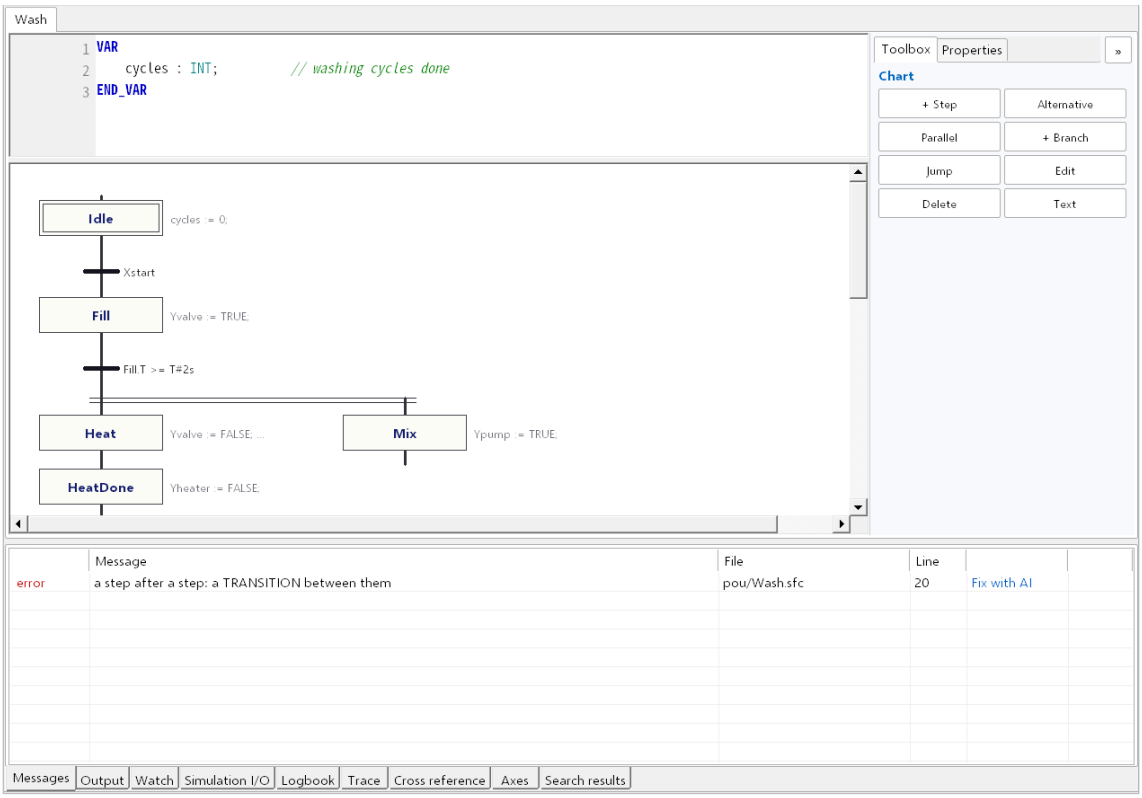


Figure 8-20 A structure error: "Messages" below lists the error, file and line number

8.5.4 Example: a washing process (with alternative and parallel branches)

Scenario: After the switch `Xstart` is turned on: fill with water for 2 s → heat for 3 s while stirring at the same time → drain for 1 s; after 3 repetitions the done lamp lights, and when the switch is turned off the process returns to standby.

Variable	Type	I/O mapping	Use
Xstart	BOOL	Digital input	Start switch
Yvalve	BOOL	Digital output	Water inlet valve
Yheater	BOOL	Digital output	Heater
Ypump	BOOL	Digital output	Stirring pump
Ydrain	BOOL	Digital output	Drain valve
Ydone	BOOL	Digital output	Done lamp
cycles	INT	— (a local variable of Wash)	Number of washes completed

In the SFC editor, click **Text** (Ctrl+T), paste the following and click **Back to the diagram**:

```
PROGRAM Wash
VAR
    cycles : INT;          // Number of washes completed
END_VAR
SFC
STEP Idle INITIAL
    cycles := 0;
END_STEP
TRANSITION Xstart
STEP Fill
    Yvalve := TRUE;
END_STEP
TRANSITION Fill.T >= T#2s
PARALLEL
BRANCH
    STEP Heat
        Yvalve := FALSE;
        Yheater := TRUE;
    END_STEP
    TRANSITION Heat.T >= T#3s
    STEP HeatDone
        Yheater := FALSE;
    END_STEP
BRANCH
    STEP Mix
        Ypump := TRUE;
    END_STEP
END_PARALLEL
TRANSITION TRUE
STEP Drain
    IF Drain.F THEN
        cycles := cycles + 1;
    END_IF;
    Yvalve := FALSE;
    Ypump := FALSE;
    Ydrain := TRUE;
END_STEP
ALTERNATIVE
BRANCH
    TRANSITION Drain.T >= T#1s AND cycles < 3
    STEP Refill
        Ydrain := FALSE;
    END_STEP
    TRANSITION TRUE
    JUMP Fill
```

```
BRANCH
    TRANSITION Drain.T >= T#1s
END_ALTERNATIVE
STEP Done
    Ydrain := FALSE;
    Ydone := TRUE;
END_STEP
TRANSITION NOT Xstart
STEP Reset
    Ydone := FALSE;
END_STEP
TRANSITION TRUE
JUMP Idle
END_SFC
END_PROGRAM
```

Text	Graphic equivalent
`STEP name [INITIAL] … END_STEP`	A step and its actions; INITIAL marks the initial step
`TRANSITION condition`	Transition
`ALTERNATIVE BRANCH … BRANCH … END_ALTERNATIVE`	Alternative branch (each branch starts with TRANSITION)
`PARALLEL BRANCH … BRANCH … END_PARALLEL`	Parallel branch (each branch starts with STEP)
`JUMP step`	Jump
`SFC … END_SFC`	Start and end of the chart (before END_PROGRAM)

#	Action
1	"Project > Add a PROGRAM (SFC, a sequence)...", name `Wash`.
2	Declare Xstart, Yvalve, Yheater, Ypump, Ydrain and Ydone (BOOL) in the GVL.
3	Click **Text** (Ctrl+T), replace the whole content with the text above, and click **Back to the diagram**.
4	Click **Check** (Shift+F7) and make sure there are no errors.
5	Click **Simulate** (F5), write TRUE to Xstart in the Watch window, and watch the chart and the outputs.

Expected result: Fill becomes active and Yvalve turns ON; after 2 s, Heat and Mix become active at the same time (Heat turns Yvalve off; Yheater and Ypump turn ON); after 3 s, Heat ends and HeatDone becomes active (Yheater OFF), and the parallel branch joins at Drain (Ypump OFF, Ydrain ON, cycles plus 1); after 1 s, while cycles is less than 3 the left branch goes back to Fill, and the 3rd time the right branch goes to Done (Ydone ON). After you write FALSE to Xstart, it goes back to Idle. The whole process takes about 18 s.

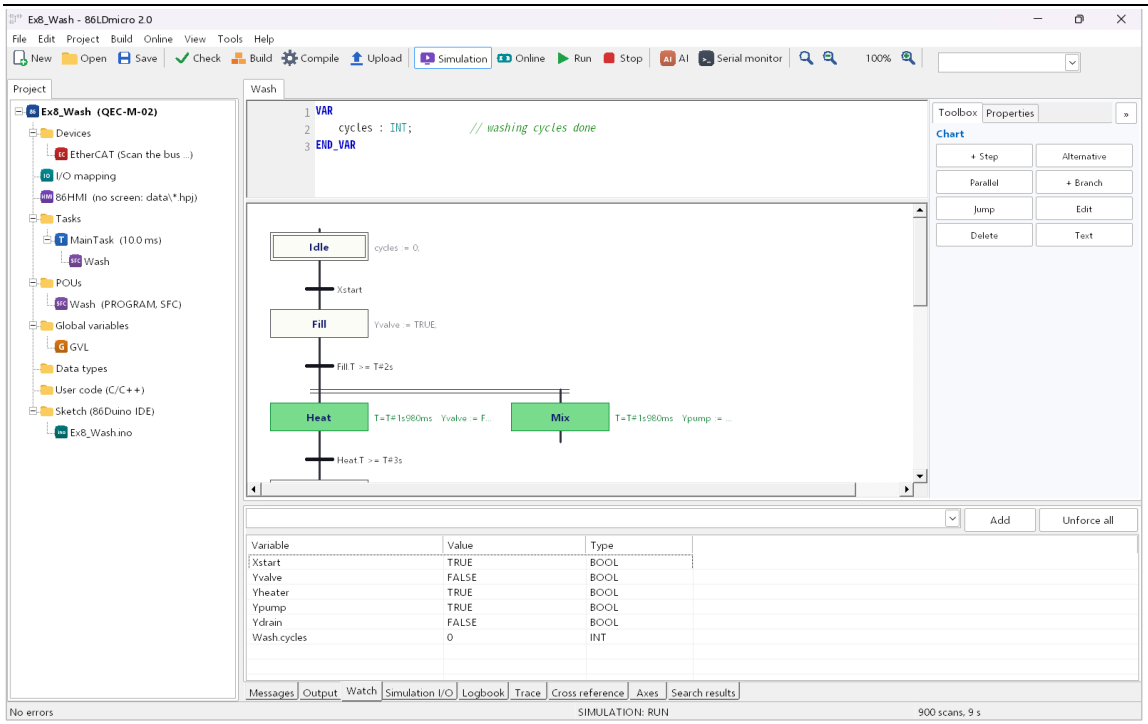


Figure 8-21 Wash in simulation: active steps are green, with the time they have been active shown on the right

8.6 SFC execution semantics

Knowing what SFC does in each scan lets you write predictable sequences.

8.6.1 The order within each scan

1. Using the step states **at the start of the scan**, evaluate the transitions after all active steps.
2. Fire the transitions that hold: leave the previous step and activate the next one (for an alternative branch after the same step, **the left has priority**).
3. Run the actions of all active steps.

So a chart advances at most one step per scan; a `TRUE` transition passes only in the next scan.

8.6.2 Step flags

Syntax	Type	Meaning	Used in
`step.X`	BOOL	The step is active	Conditions and actions of the same SFC, for example `Fill.X`
`step.T`	TIME	How long the step has been active (reset to zero when it is left)	Timeouts, delayed transitions, for example `Fill.T >= T#5s`
`step.F`	BOOL	TRUE in the **first scan** after the step becomes active	Actions done only once, for example counting `IF Drain.F THEN … END_IF;`
`program.step_X`	BOOL	Reads whether a step is active from **another POU**	For example `Wash.Done_X`; the Watch window uses this form too

8.6.3 Outputs stay as set

A variable set in an action keeps its value until other code changes it (similar to ordinary actions in CODESYS). When a step is left, 86LDmicro does not set it back to FALSE automatically. To turn it off, write := FALSE in a later step (in the example above, the Heat step turns Yvalve off at its start and the Drain step turns Ypump off; without that, the inlet valve would stay open through heating and stirring).

*__Tip:__ When a function block (for example MC_MoveAbsolute) is in a step's actions, it is no longer called after the step is left; set its Execute back to FALSE in the next step so that there is a new rising edge the next time the chart returns to that step. Function blocks that must be called all the time (for example MC_Power) are best called permanently in a separate ST or LD program (section 12.7).

8.6.4 Display in simulation and online

During simulation (F5) or online (Ctrl+F5), active steps are shown in **green**, with the time since activation displayed on the right; you can add `Program.Step_X` to the Watch window.



Figure 8-22 In simulation: active steps are green, with the time since activation shown on the right

Chapter 9 Simulation and debugging

86LDmicro has a built-in simulator that runs the whole project on your PC without a controller, letting you operate the inputs and observe the outputs. This chapter covers simulation, Simulation I/O, watching and forcing, Trace, cross reference and search, and common error messages. Watching, forcing and Trace work exactly the same way when you are **online** (connected to the controller, section 10.4).

9.1 Simulation

Click **Simulate** (F5) on the toolbar, or choose "Online > Simulate", to start the simulation; press F5 again to end it. The simulation runs the whole project: all tasks, the I/O process image, timers, serial ports, EEPROM, axes and Motion86 machines, the same way they run on the controller.

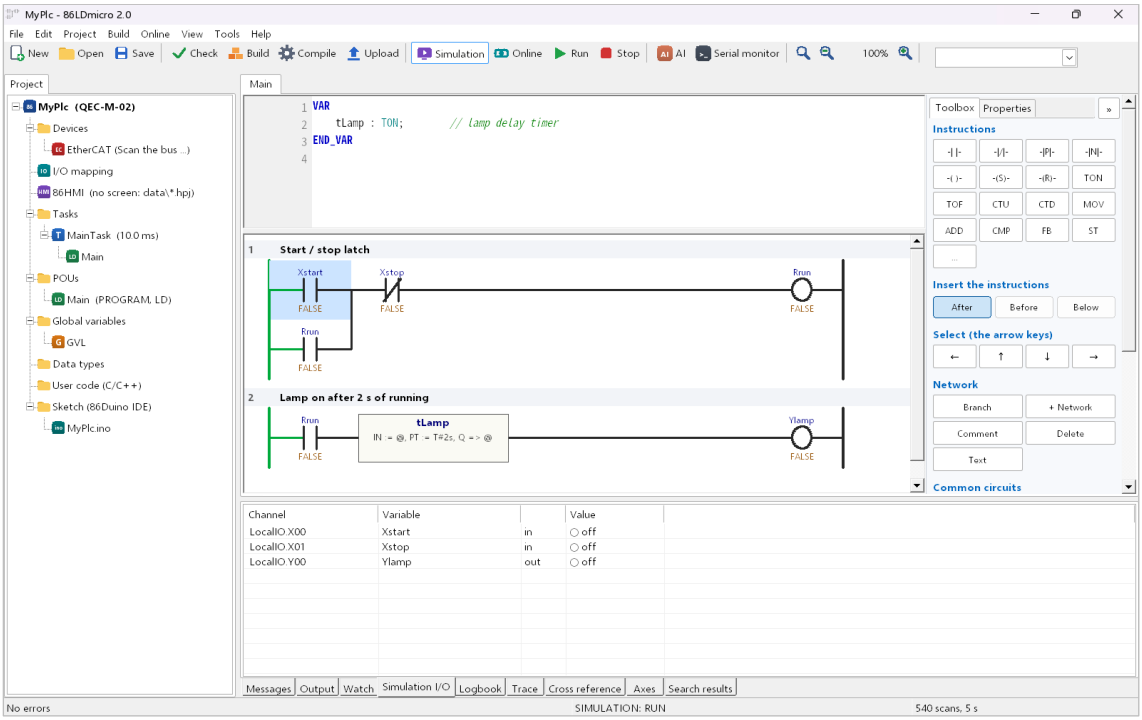


Figure 9-1 The "Simulate" button on the toolbar (pressed) and "SIMULATION: RUN" in the status bar

9.1.1 What you see during simulation

Editor	Shown during simulation	Operation
Ladder diagram (LD)	Green segments show power flow; variable values are shown below the elements, and timer blocks show the elapsed time	Double-click a contact or coil to toggle its variable between TRUE and FALSE
ST	Hover over a variable to see its type and current value	Right-click a variable: Write, Force, Add to watch
SFC	Active steps are green, with the time since activation shown on the right	—
Machine page, axis page, Axes (commissioning)	Axis position, velocity and status	Power on, jog, move, homing (sections 12.5 and 12.6)
86HMI simulator window	Opens automatically when the sketch has 86HMI screens	Click the widgets on the screen directly on your PC (section 13.5)

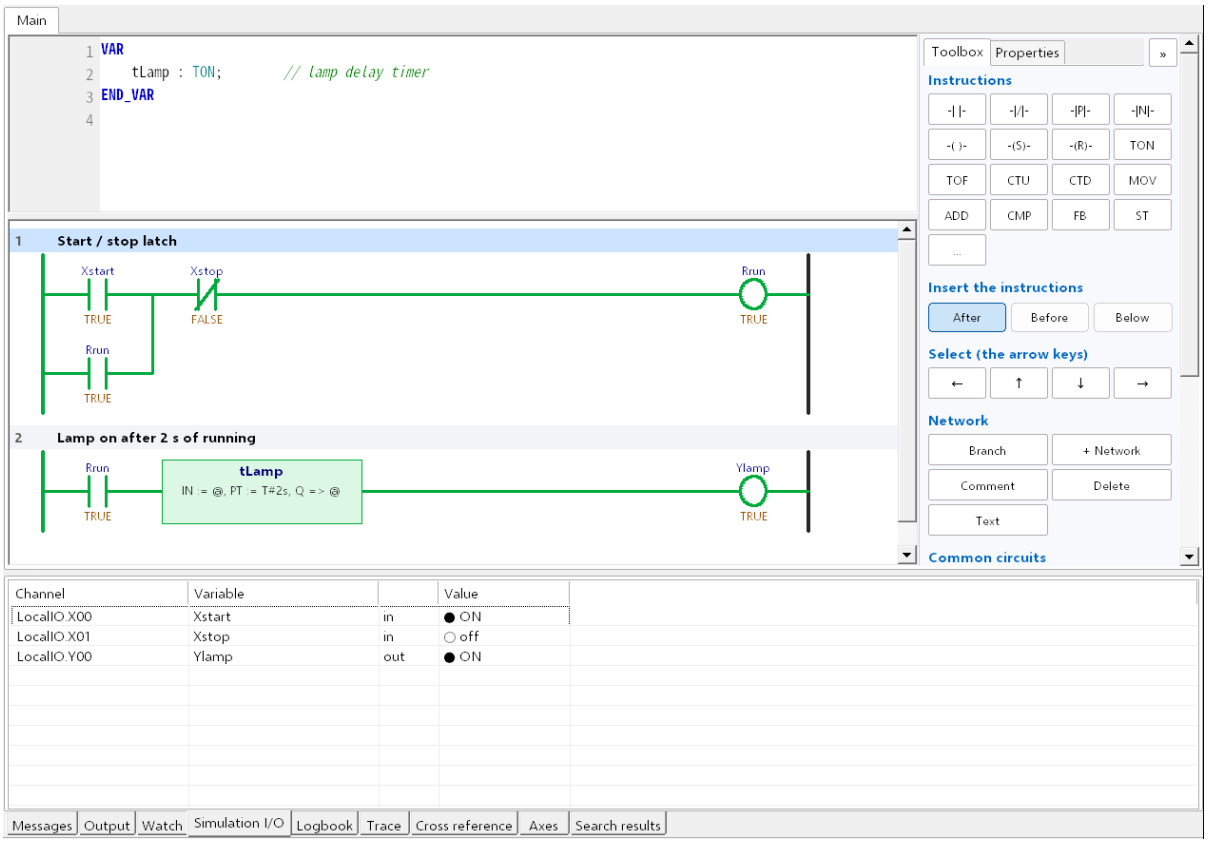


Figure 9-2 A ladder diagram in simulation: green is power flow, and timer blocks show the elapsed time

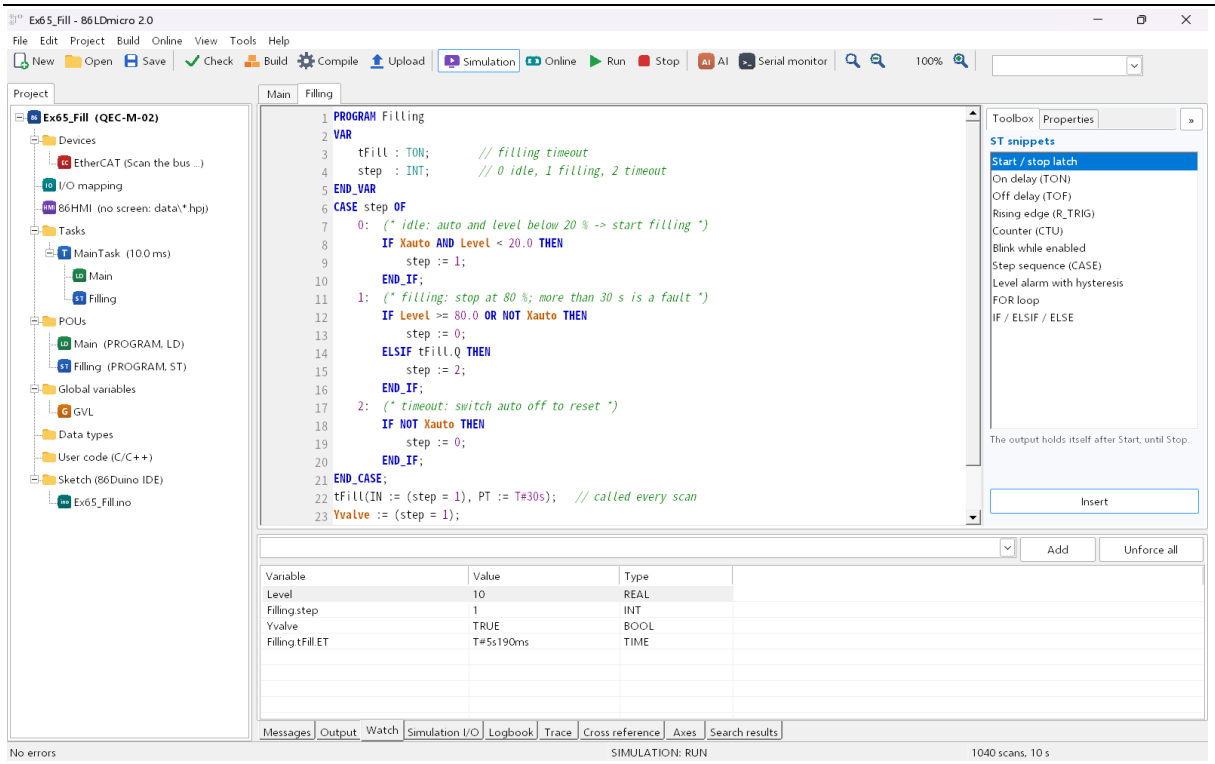


Figure 9-3 An ST program and the Watch window in simulation

*__Note:__ You cannot modify the program during simulation. To make changes, press F5 to end the simulation first, then press F5 again when you are done (the simulation starts again from the initial values).

9.1.2 Simulation speed and RUN/STOP

Commands in the "Online" menu related to simulation:

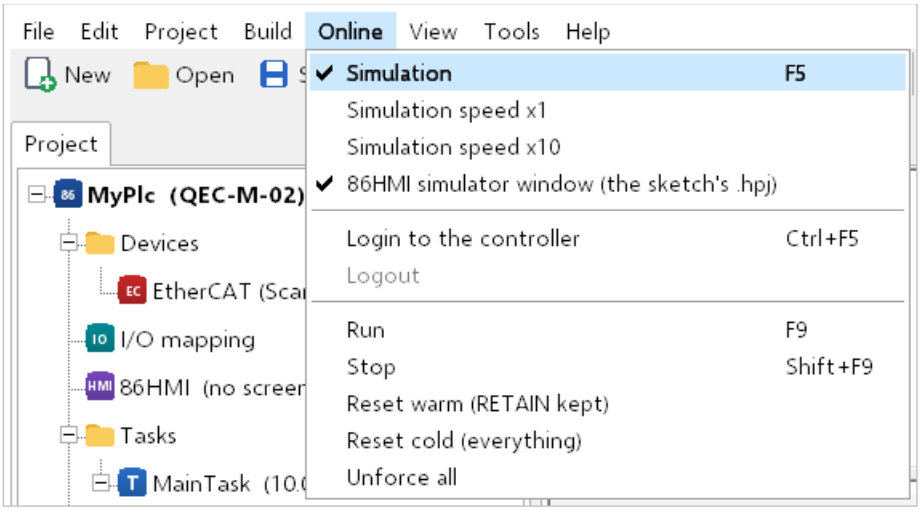


Figure 9-4 The "Online" menu: Simulate, simulation speed, Run/Stop, warm/cold restart

Item	Shortcut	Purpose
Simulate	F5	Start/end the simulation
Simulation speed x1	—	Run in real time (default)
Simulation speed x10	—	Time runs 10 times faster: timers and axis moves are all faster, which is useful for testing long processes
86HMI simulator window (the sketch's .hpj)	—	Open/close the 86HMI simulator window (section 13.5)
Run	F9	The program continues running
Stop	Shift+F9	The program stops running (the status bar shows "SIMULATION: STOP")
Reset warm (RETAIN kept)	—	Variables return to their initial values; RETAIN variables are kept
Reset cold (everything)	—	All variables (including RETAIN) return to their initial values
Unforce all	—	Release all forces (section 9.3)

*__Tip:__ A runtime error in the program during simulation (for example, an array index out of range) stops the simulation (STOP), and the cause is recorded in the "Logbook", for example `STOP: Main line 12: ….` Fix the program and simulate again.

9.2 Simulation I/O

The "**Simulation I/O**" tab in the lower tool window lists the physical point mapped to each I/O (section 4.6), including local I/O and EtherCAT PDO entries: channel, variable, direction and current value.

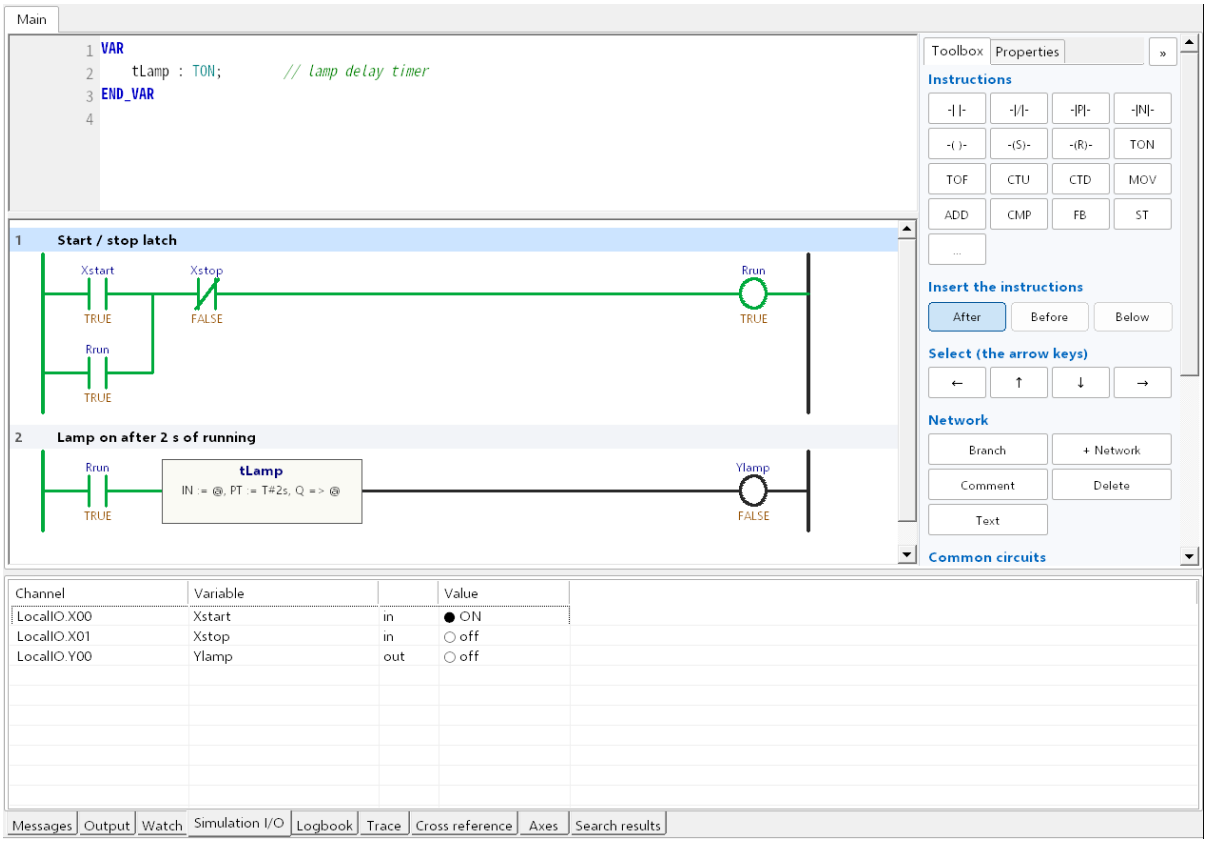


Figure 9-5 Simulation I/O: physical points such as LocalIO.X00 (Xstart)

Physical point	Operation
Digital input	Double-click to toggle ON/OFF, just like pressing or releasing a physical button
Analog input	Double-click and enter a value
Outputs (digital, analog)	Display only; set by the program

Inversion, filtering and scaling in the I/O mapping are handled in simulation the same way as on the controller.

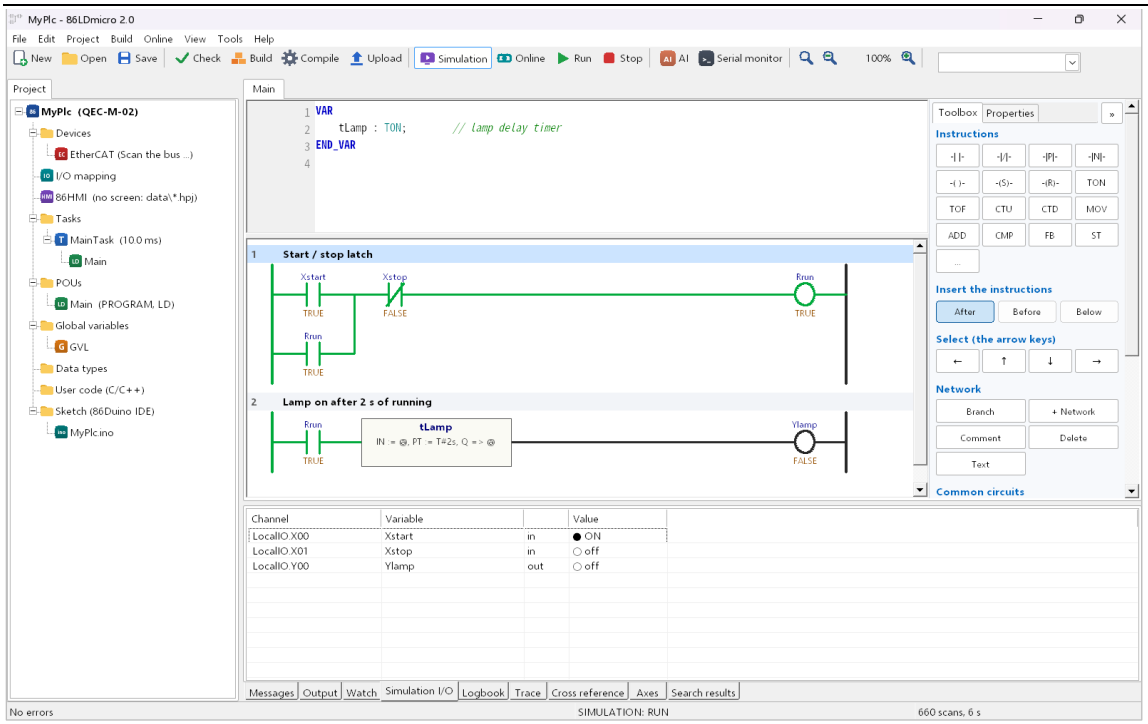


Figure 9-6 After double-clicking LocalIO.X00: Xstart is ON and Rrun latches

> ***Note:** A variable mapped to a physical input (for example `Xstart`, connected to `LocalIO.X00`) is set by "Simulation I/O" during simulation; if you write it directly in the Watch window, the input value overwrites it on the next scan. Toggle it in "Simulation I/O", or use force (section 9.3).

9.3 Watch, write and force

The "**Watch**" window lists the variables you choose with their live values, and lets you write or force them. Values are shown, and can be written, only during simulation or online.

9.3.1 Adding variables

- Type a variable name in the input box at the top of the Watch window (the drop-down lists all variables) and click **Add**. You can enter `Program.Variable` (for example `Wash.cycles`), `Instance.Member` (for example `Panel.bl.Out`), an array element, or an SFC `Program.Step_X`.
- Or right-click a variable in the program > "Add <variable> to the watch window".
- Select a row and press Del, or right-click > "Remove from the watch", to remove that variable.

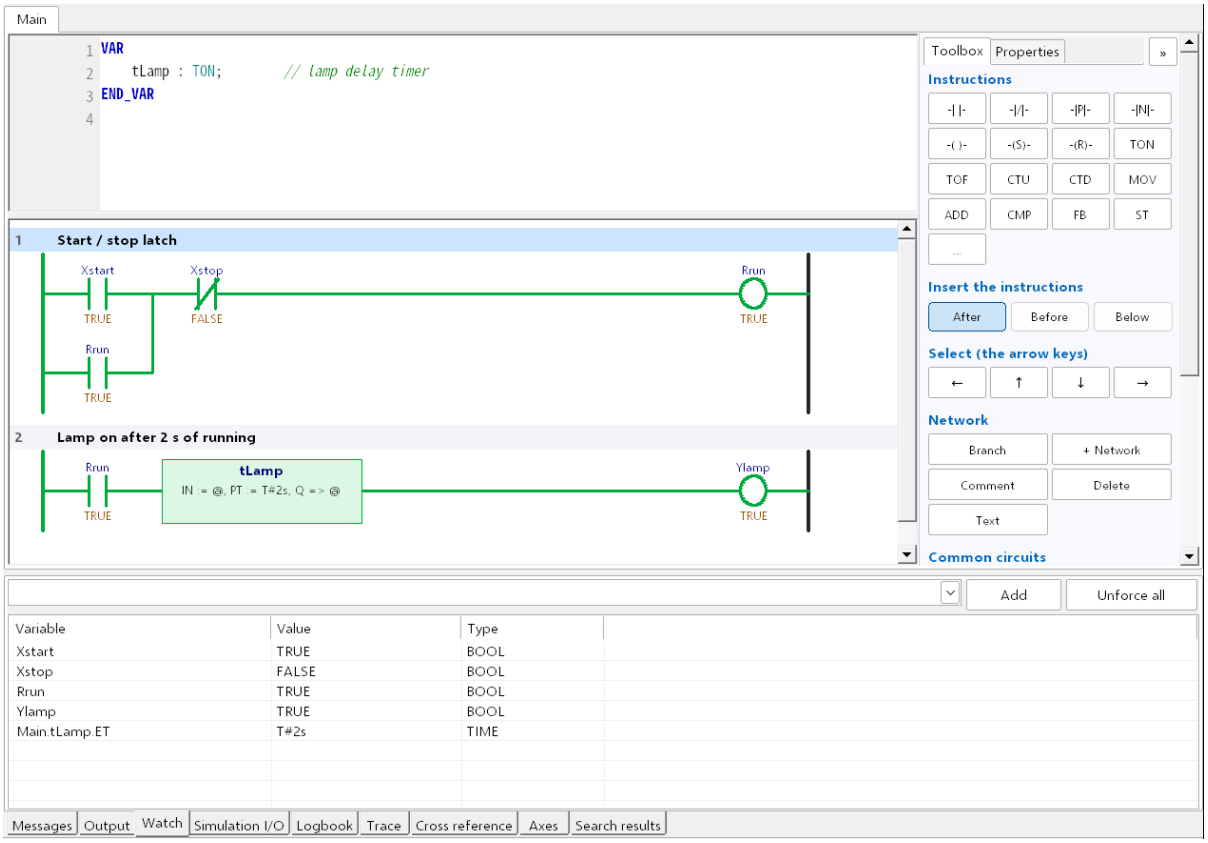


Figure 9-7 The Watch window: values are shown live after variables are added

9.3.2 Write and force

Function	How	Effect
Write	Double-click a row in the Watch window: a BOOL toggles between TRUE and FALSE directly, other types prompt for a value; or right-click > "Write <variable>..."	Writes **once** only; the program can still change it afterwards
Force	Right-click > "Force <variable>..." and enter a value	The same value is written every scan, and `[F]` is shown after the value
Unforce	Right-click > "Unforce"	Releases the force on this variable
Unforce all	The **Unforce all** button at the top right of the Watch window, or "Online > Unforce all"	Releases the force on all variables

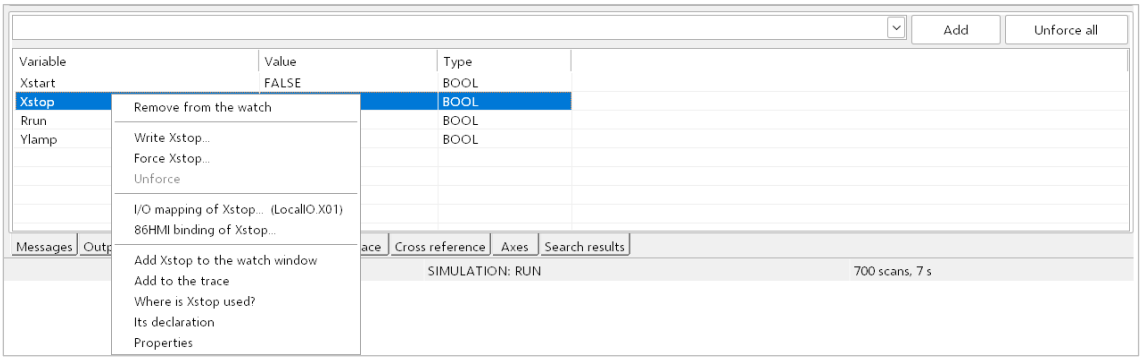


Figure 9-8 The Watch window's context menu: Write, Force, Unforce, Add to the trace, etc.



Figure 9-9 Forcing Xstop: enter the value to hold

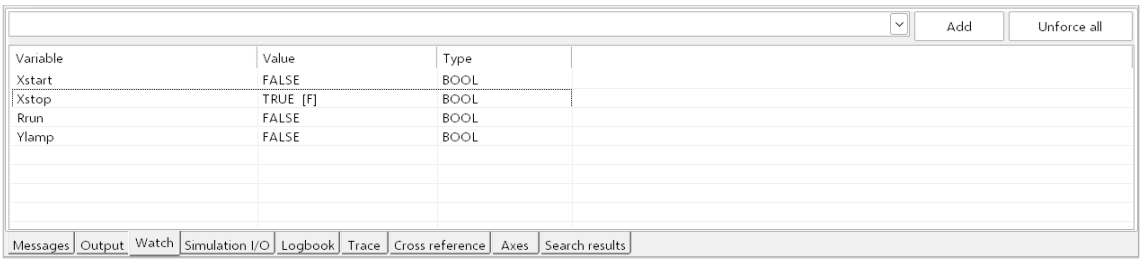


Figure 9-10 A forced variable shows [F] after its value

9.3.3 When forced values are written

A forced value is written **twice** every scan, as in CODESYS:

1. After the inputs are read and before the programs run: the programs read the forced value.
2. After the programs run and before the outputs are written: the forced value goes to the physical output.

If a program writes the variable itself during the scan, the programs that run **after it** in the same scan read the value the program wrote, not the forced value; the forced value only overwrites it again at the end of the scan. Therefore:

- Forcing **is** most effective on **input variables written by the I/O mapping** (physical inputs) and **output variables.**
- For an intermediate variable that a program rewrites every scan, the Watch window shows the forced value after forcing, but other programs may not read it.
- When a force "has no effect" or "reverts right away", use the cross reference (section 9.5) to find which programs write that variable.

*__Tip:__ Example: in the demo mode of `tools\example\ex30_dispense` (`gDemo = TRUE`), the first program, PlantSim, rewrites `sim*` and IoMap then rewrites `x*`, so forcing these variables gets overwritten (it reverts after one cycle). To test the process with forcing, turn the demo mode off first (`gDemo = FALSE`; you can also force it), then force the physical inputs `diEntry`, `diDoorClosed`, etc.

*__Safety:__ Forcing takes the outputs out of the program's control; on a real machine, make sure the site is safe before you do it. When the connection to the controller is lost for 3 seconds, the controller releases all forces automatically.

9.4 Trace

The "**Trace**" window plots the changes of variables as curves, which is useful for observing timing, analog values and axis positions.

1. Right-click a variable in the Watch window (or in the program) > "**Add to the trace**". Up to 8 variables.
2. The "Trace" tab comes to the front automatically: it samples every 100 ms and shows the last minute.
3. Right-click the trace: **Pause**, **Clear the samples**, **Remove all variables**, **Copy as CSV** (paste into Excel for analysis).

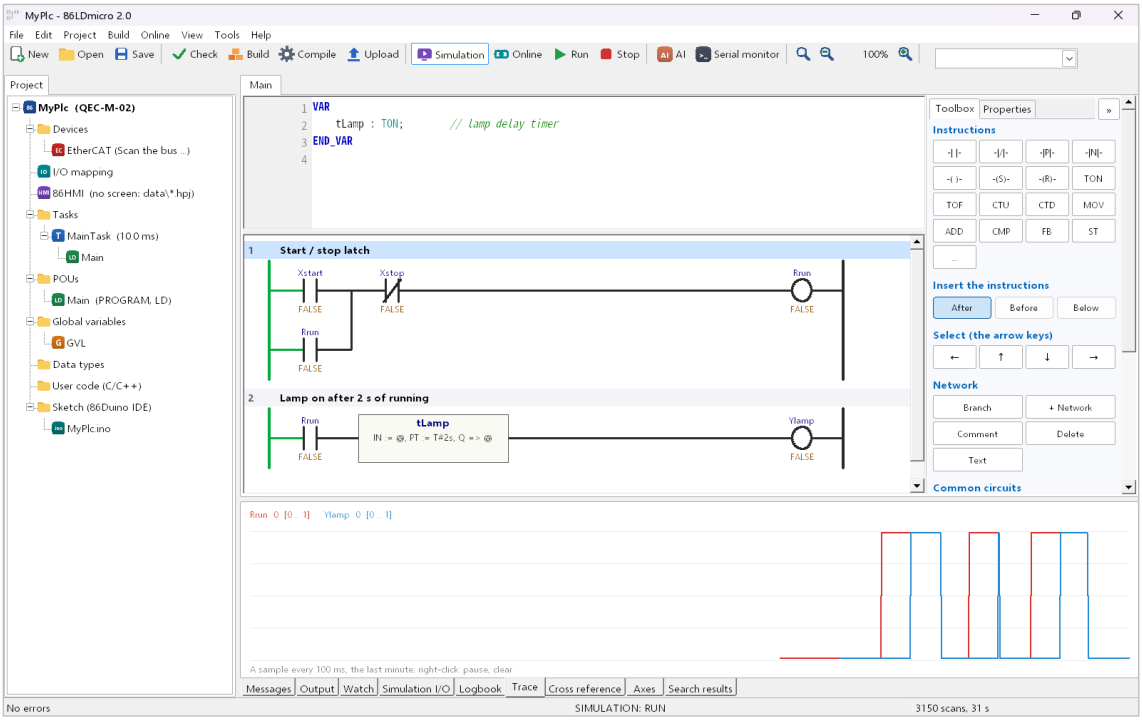


Figure 9-11 Trace: changes of Rrun and Ylamp

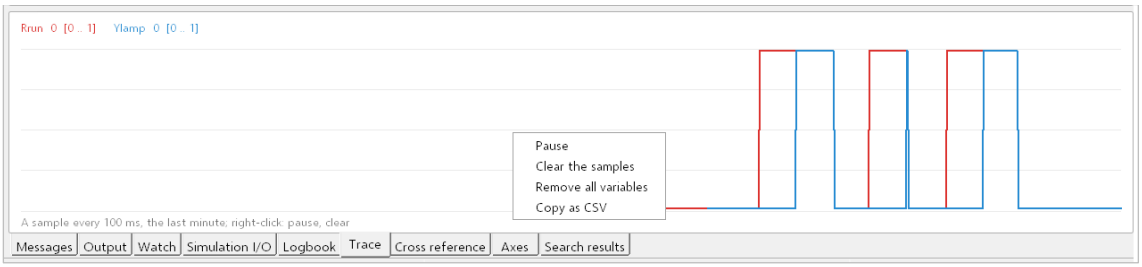


Figure 9-12 The Trace context menu

9.5 Cross reference and search

As a project grows, you often need to find "where this variable is used" or "which program contains this text". 86LDmicro offers three ways to search:

Function	Shortcut	Searches for	Best for
Find in this tab	Ctrl+F (F3/Shift+F3 next/previous)	Text or ladder elements in the current tab	Searching one program item by item
Find a variable in the project (Cross reference)	Ctrl+Shift+F	Reads, writes and declarations of a variable everywhere	Finding out what writes a variable
Find in the project	Ctrl+Shift+H	Any text in the whole project (including comments and .ino)	Finding comments, strings, functions, or things whose name you are not sure of

9.5.1 Cross reference (Ctrl+Shift+F)

1. Press **Ctrl+Shift+F** ("Edit > Find a variable in the project..."), enter the name in the "Find a variable" window, and click **OK**. You can also right-click a variable > "Where is <variable> used?".
2. The "**Cross reference**" tab at the bottom lists every occurrence: file, line, use (**Declarations**, **Read**, **Write**) and the content of that line; the status bar shows the total count.
3. Double-click a row to open that file and jump to that line.

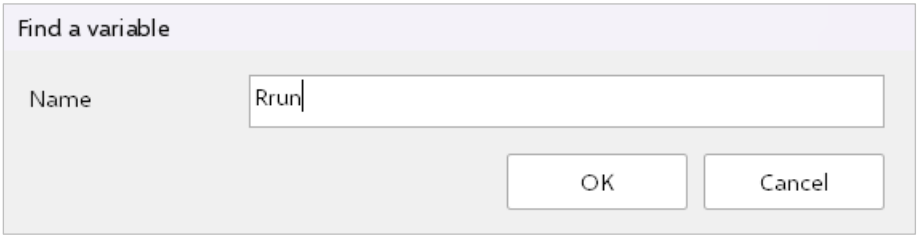


Figure 9-13 The "Find a variable" window

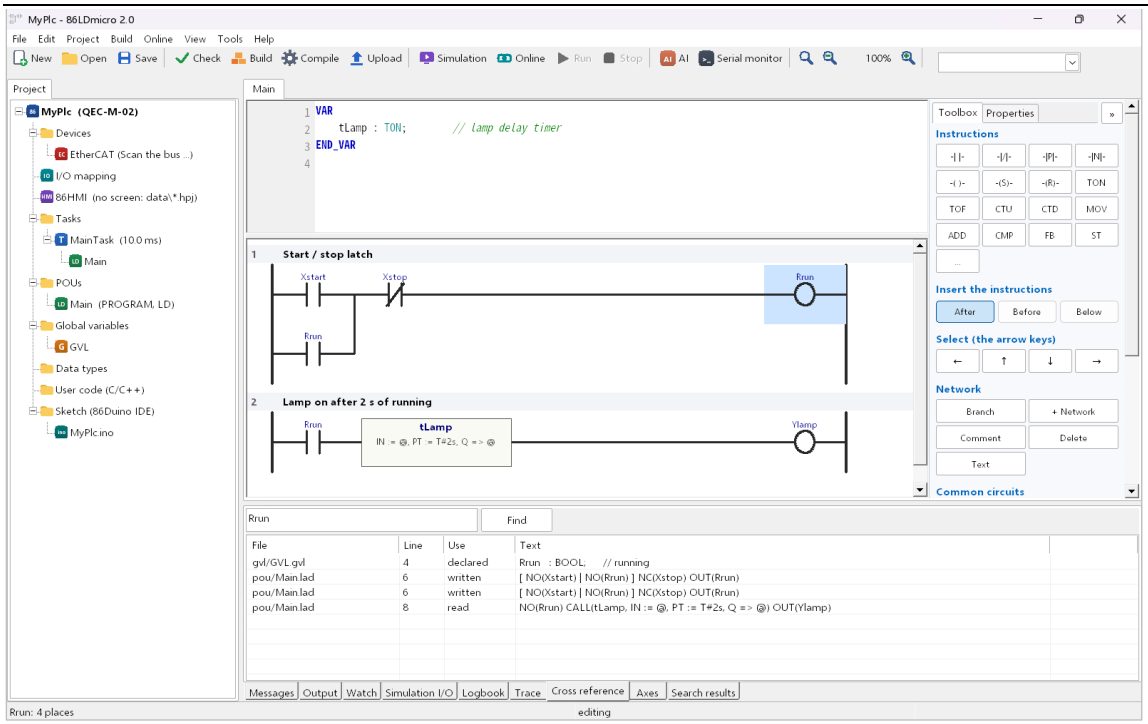


Figure 9-14 Cross reference: where Rrun is declared, read and written

***Tip:** A location whose use is "Write" means the left side of :=, or OUT, SET, RST, MOV, etc. in a ladder diagram. When two or more programs write a variable, the program that runs last determines its value at the end of the scan.

9.5.2 Find in the project (Ctrl+Shift+H)

Searches all POUs (network titles, comments and elements of ladder diagrams, ST, SFC), global variable lists, data types, user C/C++ code, and the sketch's .ino.

1. Press **Ctrl+Shift+H** ("Edit > Find in the project..."). The lower tool window switches to "**Search results**". If you select some text in the program before pressing it, that text is searched directly.
2. Enter the text to find and press **Enter** or click **Find**. If needed, check "**Match case**" or "**Whole word**" (for example, so that searching for `Pump` does not list `PumpSpeed`).
3. Each result row is one occurrence: location (POU name and language), line number and the content of that line; the top right shows the total count.
4. **Double-click** a row (or select it and press Enter) to open that file and jump to that line: in ST and C/C++ the line is selected; in a ladder diagram the view scrolls to and selects that network.

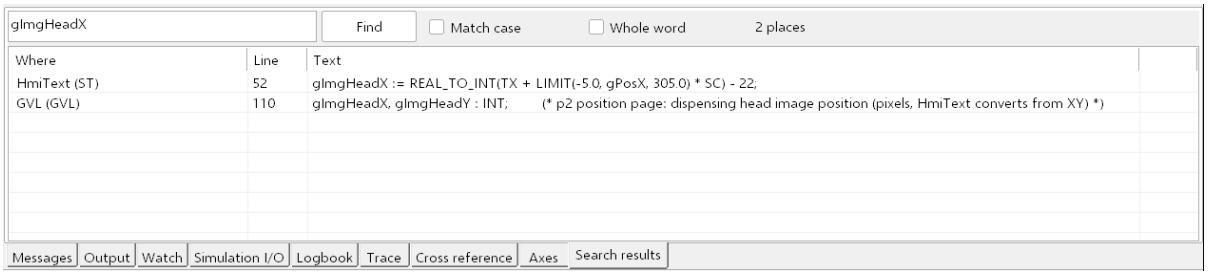


Figure 9-15 "Search results": searching for gImgHeadX finds 2 occurrences (an ST program and a global variable list)

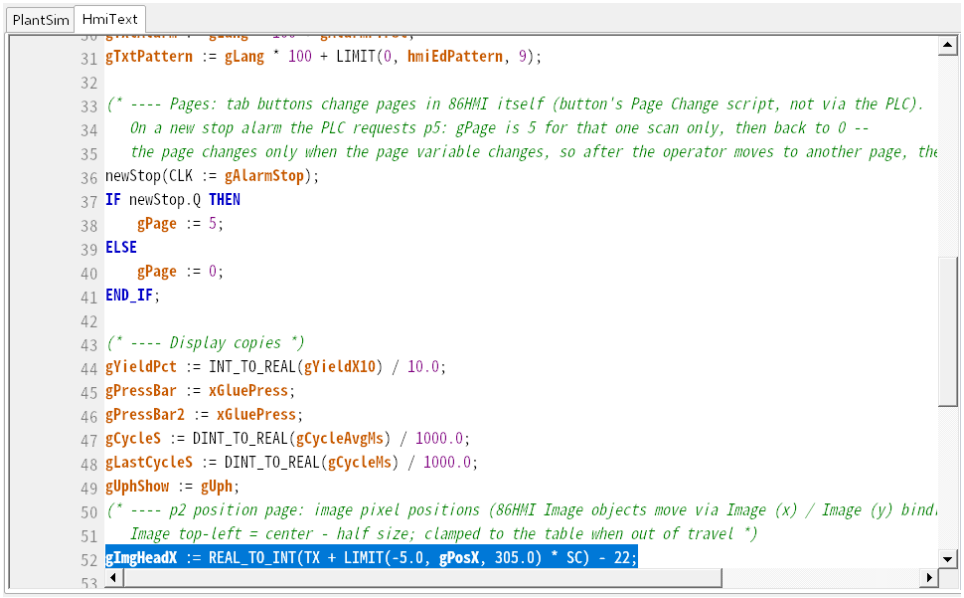


Figure 9-16 Double-clicking the first row: HmiText opens with line 51 selected

> ***Tip:** Search only finds; it does not replace. To rename a variable, open and fix each result one by one, then click **Check** (Shift+F7) to make sure nothing was missed. You can also search from the command line: `86ldmicro-cli search <project> <text> [--case] [--word]` (section 15.5).

9.6 Common error messages

When you click **Check** (Shift+F7) or build, the "**Messages**" window lists errors (which must be fixed) and warnings (which you should review). Elements with errors are marked with a **red background** in ladder diagrams and underlined in ST.

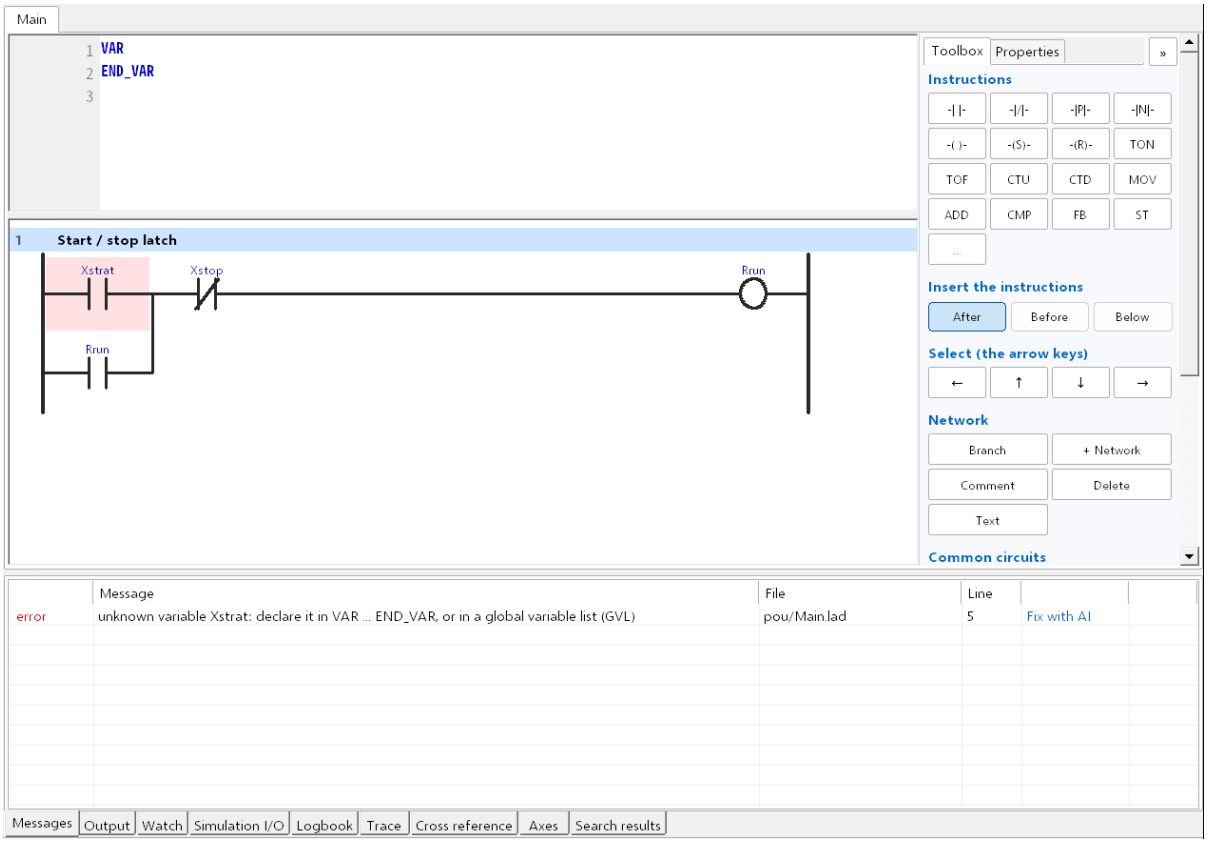


Figure 9-17 A mistyped variable name: the element has a red background, and the build output shows "Build: the project has errors (Messages)"

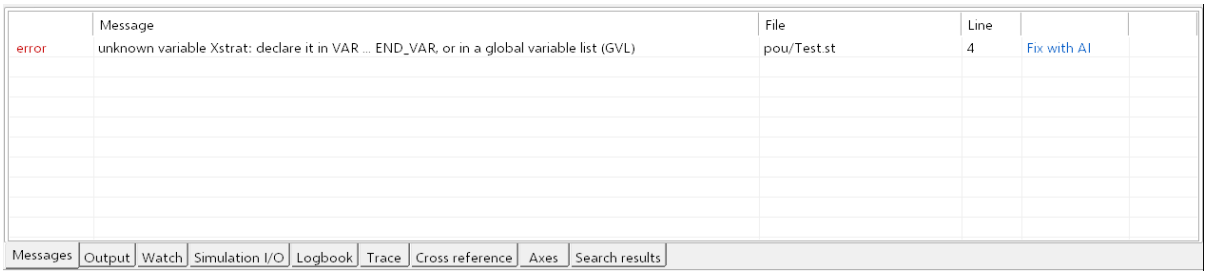


Figure 9-18 The "Messages" window: one error per row; double-click to jump to the element

1. Switch to the "Messages" tab at the bottom.
2. Double-click an error row; the view jumps to the element or program line with the error and selects it.
3. After fixing it, click **Check** (Shift+F7) again until there are no errors.
4. If needed, click **Fix with AI** on that row to send the error and the program to the AI assistant (section 15.3).

9.6.1 Compiler message reference

The table below lists common messages (the names in quotes depend on your program):

Message (excerpt)	Cause	Fix
unknown variable Xstrat: declare it in VAR ... END_VAR, or in a global variable list (GVL)	Name misspelled or not declared	Fix the spelling, or declare it in the GVL/VAR
REAL is not BOOL	A number used as a contact or condition	Write a comparison: `x <> 0`, `CMP(x > 10)`
expected a statement, but found '1' (END_IF missing)	An ST IF/FOR/CASE has no end	Add END_IF/END_FOR/END_CASE
missing ';'	Semicolon missing at the end of an ST statement	Add `;` after the indicated position
expected ')', but found ';'	Unbalanced parentheses	Check the parentheses
unknown function block instance tLamp	The instance used by CALL is not declared	Declare `tLamp : TON;`
TON has no input PTT (inputs: IN, PT)	Wrong function block parameter name	Use a parameter name listed in the message
Count is declared twice in the GVL	Variable declared twice	Delete one of them
the value 100000 does not fit INT	Constant out of the type's range	Use DINT instead, or fix the value
unknown variable AxisQ	Wrong axis name (MC function blocks)	Use the axis name from the machine page (for example AxisX)
task MainTask runs Ghost, which is not a PROGRAM of the project	A POU in the task was renamed or deleted	Remove it or choose another one in the task settings (section 4.3)
a step follows a step (SFC)	Transition missing between two steps	Insert a transition between them
Warning: Xauto is only read here: it has no I/O mapping and no program writes it -- it is always FALSE	An input variable was not mapped	Add a mapping on the I/O mapping page (section 4.6), or confirm that this is intended
Warning: `the program Test is in no task: it does not run` (English message)	A newly added PROGRAM was not put in a task	Add it in the task settings (section 4.3)

9.6.2 Messages during upload, connection and runtime

Message/symptom	Fix
Could not open COMx (in use by another program, such as the 86Duino IDE's Serial Monitor?)	Close the program using the port, then upload again (section 10.1)
The controller runs another program (or another version of this one)	Upload again (F8), then connect (section 10.4)
Online COMx: no answer	Make sure "The online view" is enabled in the project settings; do not let the program use the USB Serial; upload again
Serial monitor: `EtherCAT: begin failed`	The EtherCAT cable is not connected to the MainDevice's EtherCAT port, or the SubDevices are not powered (section 11.7)
Serial monitor: `EtherCAT: the ENI … cannot be used`	The ENI was not uploaded to the MainDevice or does not match the actual network; Scan again, then export and upload the ENI (section 11.7)
Serial monitor: `EtherCAT: no slave`	EtherCAT cable not connected, SubDevices not powered, or the cable plugged into OUT instead of IN
Logbook: `MainTask watchdog: STOP, …`	The scan time exceeded the watchdog setting, and the PLC went to STOP (section 10.5)
Logbook: a drive left Operation Enabled	Check the drive's alarm code; after clearing the cause, run MC_Reset and then MC_Power (section 12.9)

For more problems and solutions, see Chapter 16.

Chapter 10 Building, uploading and the online view

Once the program has been verified in simulation, you can build it, compile it and upload it to the QEC controller, then use the online view to observe the live state on the controller. This chapter explains these commands in order, the files a build generates, the serial monitor, the online view, the Logbook and the watchdog, and how to upload only the 86HMI resources.

10.1 Check, build, compile and upload

The four commands are in the "Build" menu and on the toolbar; from left to right, each does one step more than the previous one: upload builds and compiles first, and compile builds first.

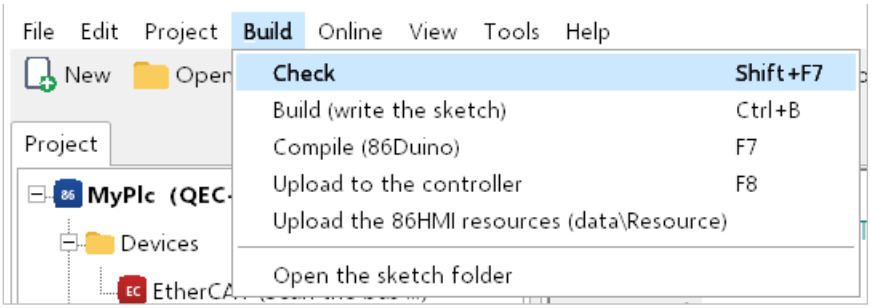


Figure 10-1 The "Build" menu

Item	Shortcut	Purpose	Requires
Check	Shift+F7	Compiles only the PLC program and lists errors and warnings in "Messages"; writes no files	—
Build (write the sketch)	Ctrl+B	Writes the project as the sketch's `plc86.cpp` and other files (section 10.2); when started from the IDE, tells the IDE to reload	—
Compile (86Duino)	F7	After building, compiles with the 86Duino IDE's arduino-builder; C/C++ errors point back to the line numbers in .cpp86	86Duino IDE (section 2.6)
Upload to the controller	F8	After compiling, uploads over USB with v86dude; if the sketch has `data\Resource`, it is uploaded too	IDE, USB cable, serial port selected on the toolbar
Upload the 86HMI resources (data\Resource)	—	Uploads only the 86HMI images, fonts and screens; no compile, no change to the program (section 10.6)	IDE, USB cable, serial port
Open the sketch folder	—	Opens the sketch folder in File Explorer	—



Figure 10-2 The toolbar: Check, Build, Compile, Upload and the serial port selection

The progress is shown in "**Build output**" in the "Output" window at the bottom, and errors go to "**Messages**" (section 9.6).

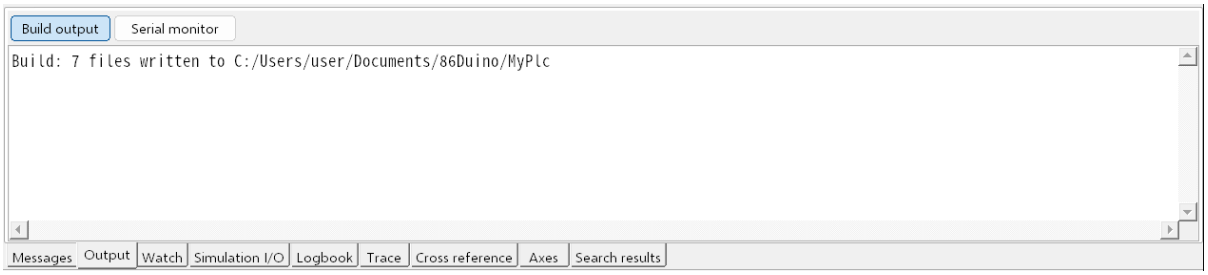


Figure 10-3 "Build output": the progress of building, compiling and uploading is all shown here

10.1.1 Uploading step by step

1. Connect the PC and the QEC controller with a USB cable, and power on the controller.
2. In the serial port list on the right of the toolbar, select the controller's COM port (for example COM4; if it is not listed, choose "Tools > Refresh the serial ports", section 2.4).
3. Click **Upload** (F8). "Build output" shows "Build: … files written …", "== Compiled…", "== Uploading to COM4 ..." and the progress in turn, and finally "== Uploaded".
4. The controller restarts and begins running the program. If "Online again after an upload" is checked in "Tools > Options" and you were online before, it reconnects automatically (section 2.6).

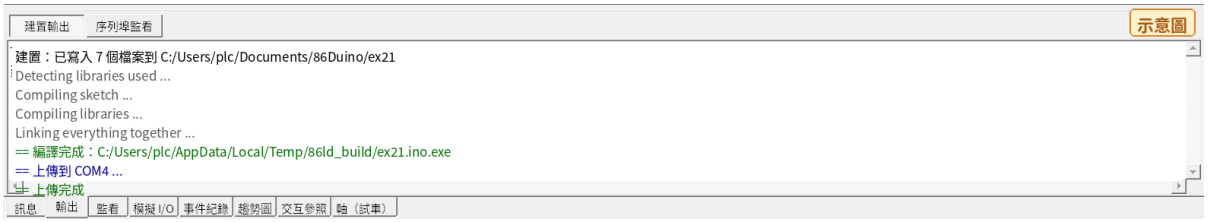


Figure 10-4 A successful upload: Build -> Compiled -> Uploading to COM4 -> Uploaded (schematic)

10.1.2 When the upload fails

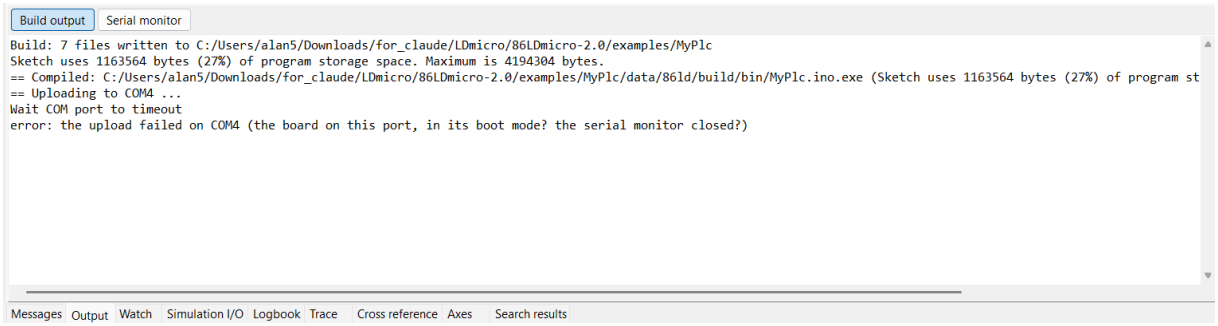


Figure 10-5 Upload failed: COM4 could not be opened (the serial port is in use by another program)

Message in the output	Cause	Fix
Could not open COMx (in use by another program…)	The 86Duino IDE's Serial Monitor or a terminal program has the same port open	Close that program, then press F8 again
COMx not found	USB cable unplugged, wrong port selected, or the controller not powered	Check the COM port (section 2.4) and select it again
The 86Duino IDE is not found	86LDmicro was started on its own and the IDE folder is not set	Set the 86Duino IDE folder in "Tools > Options" (section 2.6)
Compile error (.cpp file name and line number)	An error in the C/C++ user code or a library	Double-click the message to jump to the line in .cpp86 and fix it (Chapter 14)
Build: the project has errors	The PLC program has errors (elements with a red background)	Look at the "Messages" tab (section 9.6)
Timeout partway through the upload	Unstable USB cable or hub, or the controller is rebooting	Connect directly to a USB port on the PC and upload again

10.1.3 One USB cable, four uses

On the PC, the QEC's USB cable is one serial port (the COM port selected on the toolbar). It is used for the following four things, but **only one can use it at a time**; 86LDmicro releases and reconnects it automatically:

Use	Used by	Description
Upload	v86dude (F8)	Writes the program to the controller; during upload 86LDmicro releases the serial port automatically (going offline first if online) and reconnects when done
Online view	86LDmicro (Ctrl+F5)	When "The online view" is enabled in the project settings and the serial port is `Serial`, watching, forcing and power flow all go over this cable
Serial.print	Your program	Text printed by the program with `Serial.print()`; when online it is interleaved with the watch packets and shown automatically in the "Serial monitor"
Configurator (USB)	The EtherCAT page	The EtherCAT Configurator in USB mode also uses this cable (3 Mbps); it pauses automatically during upload or when online (Chapter 11)

***Note:** In C/C++ user code, use `Serial1` to `Serial3` (the controller's COM1 to COM3) to communicate with external devices; the USB `Serial` is reserved for uploading and the online view. Do not print in `plc86_user_ethercat()`.
If the program itself prints to USB with `Serial.print` while the PC is not reading that port (86LDmicro not connected, serial monitor not open), `Serial.print` waits indefinitely once the buffer is full, and the whole PLC scan stops. Programs in production should not print to USB, or should check `Serial.availableForWrite()` first.

10.2 Generated files

A build converts the whole project (LD, ST, SFC, GVL, I/O mapping, devices, 86HMI bindings) into C++, writes it into the sketch folder, and then the 86Duino IDE's compiler compiles it. These files are **regenerated on every build; do not edit them by hand**.

File	Generated by	Content
`plc86.cpp`, `plc86.h`	Build (Ctrl+B)	The PLC program itself: all POUs, global variables, I/O processing, task scheduling; `plc86_setup()`/`plc86_loop()`
`plc86_rt.cpp`, `plc86_rt.h`	Build	Runtime: online view protocol, forcing, RUN/STOP, restart, RETAIN, Logbook, trace sampling, watchdog
`plc86_ecat.cpp`, `plc86_ecat.h`	Build	EtherCAT MainDevice (ENI, SubDevices, PDO mapping, Motion86 machines); an empty file when there are no EtherCAT devices
`<sketch>.ino`	Created when the project is created	Calls `plc86_setup()`, `plc86_loop()`; you can add your own code, and a build only adds the parts the PLC needs (for example `#include "plc86.h"`, `Hmi.begin();`)
`data\86ld\build\plc86.symbols.json`	Build	Symbol table: used by the online view to match the program on the controller and find variables
`data\86ld\build\bin\`	Compile (F7)	Intermediate files and executable of the compile (`<sketch>.ino.exe`)
`myhmi.cpp`, `myhmi.h`, `data\Resource\`	"Code Generation" in the 86HMI Editor	86HMI screens and resources (Chapter 13); not generated by a build

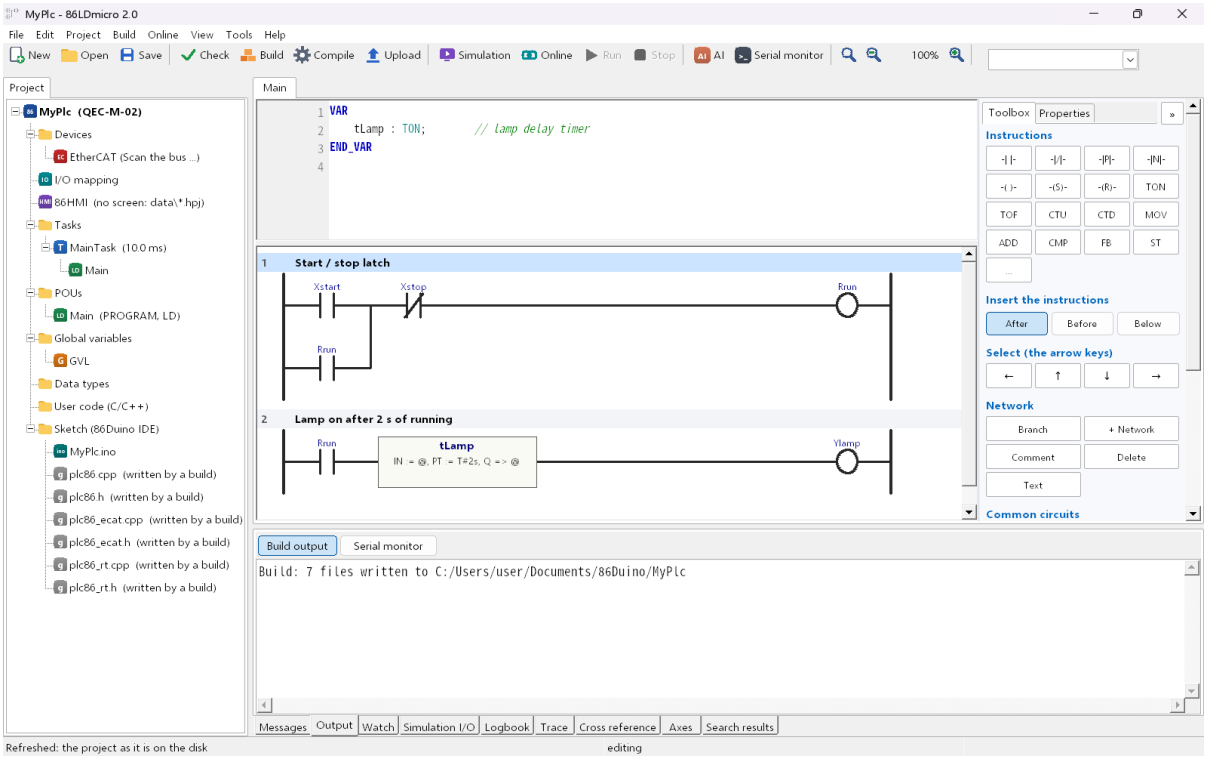


Figure 10-6 "Sketch (86Duino IDE)" in the project tree: files generated by a build are marked "(written by a build)", and the build output shows "7 files written"

Under "Sketch (86Duino IDE)" in the project tree, you can open the .ino and these files directly to view them; they also appear as tabs in the 86Duino IDE.

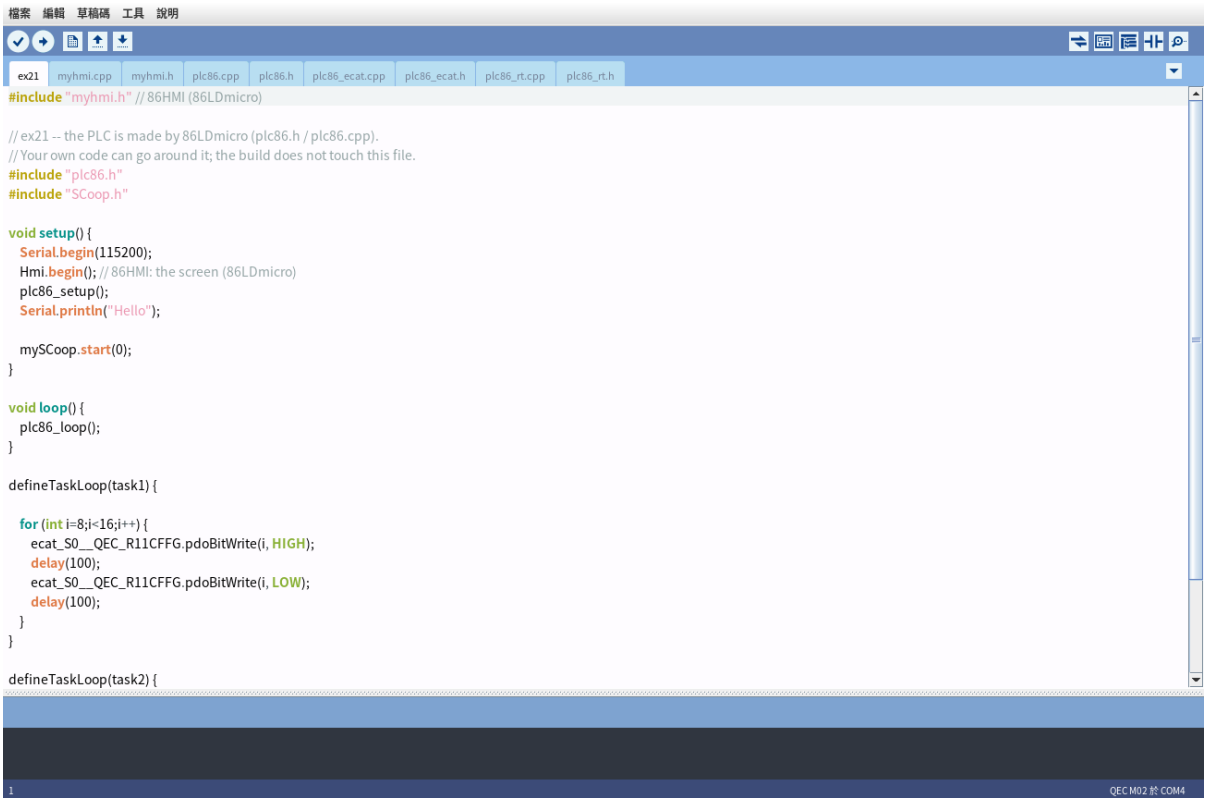


Figure 10-7 The 86Duino IDE after a build: tabs for plc86.cpp, plc86.h, plc86_ecat.cpp, etc.

> ***Tip:** The `plc86*` files and `data\86ld\build` can be left out of version control (git, etc.); just build again when you need them.

10.3 Serial monitor

The "Serial monitor" shows the text the controller prints on a serial port, such as the program's `Serial.print()` output and the status of the EtherCAT MainDevice. Click **Serial monitor** on the toolbar, or switch to "Serial monitor" at the top of the "Output" window at the bottom.

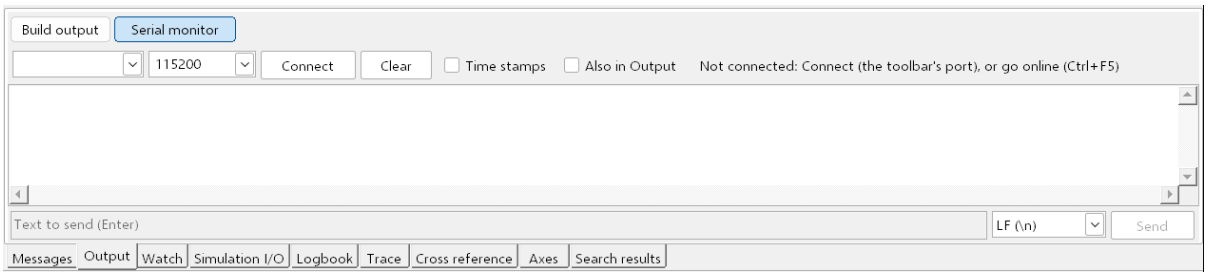


Figure 10-8 The serial monitor: port selection, baud rate, connect, time stamps, send

Situation	How to view
When online (Ctrl+F5)	The program's `Serial.print()` output is shown automatically over the online connection, with no separate connection needed; you cannot send while online
When not online	Select the serial port (blank = the toolbar's port; you can also select a USB-to-RS-232/485 adapter to view Serial1 to Serial3) and the baud rate, then click **Connect**; click **Disconnect** to end
During upload	Releases the serial port automatically, and reconnects after the upload is done and the controller has restarted

- **Time stamps**: adds the time received at the start of each line.
- **Also in Output**: each line also appears in "Build output".
- Type text in the input box at the bottom and press Enter or click **Send** (a line ending can be selected); **Clear** clears the display.

When there are EtherCAT devices, the MainDevice status is also printed here, so you can check it without going online, for example:

```
EtherCAT: running, 2 slaves, 1000 us, sync, ENI C:/eni/QEC-ENI-1.xml
```

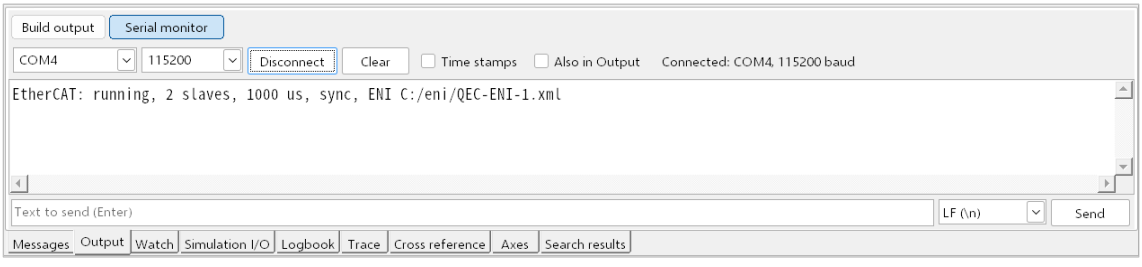


Figure 10-9 EtherCAT MainDevice status in the serial monitor

10.4 Online view

The online view connects 86LDmicro to the controller so that you can see the program running on the controller from your PC: power flow, variable values, SFC steps; you can also write, force and RUN/STOP.

10.4.1 Preparation

"**The online view (monitor, force, power flow)**" in "Project > Settings" is checked by default; the serial port used is `Serial` (USB, the same cable as for uploading), and the baud rate defaults to 115200. If you change these settings, build and upload again (section 4.2).

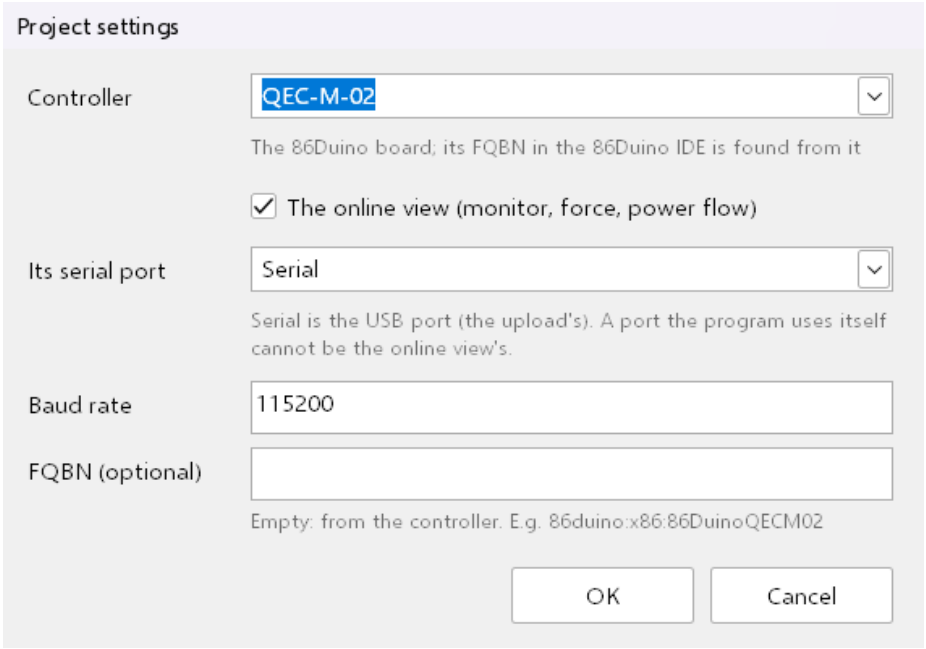


Figure 10-10 Project settings: online view, serial port used, baud rate

10.4.2 Connecting

1. Upload the program (F8).
2. Click **Online** on the toolbar, or choose "Online > Login to the controller" (Ctrl+F5).
3. Once connected, the status bar shows "ONLINE COM4: RUN" together with the scan time, its maximum, the number of scans and the number of overruns; ladder diagrams and SFCs show live power flow and values.
4. To end, choose "Online > Logout", or click **Online** on the toolbar again.

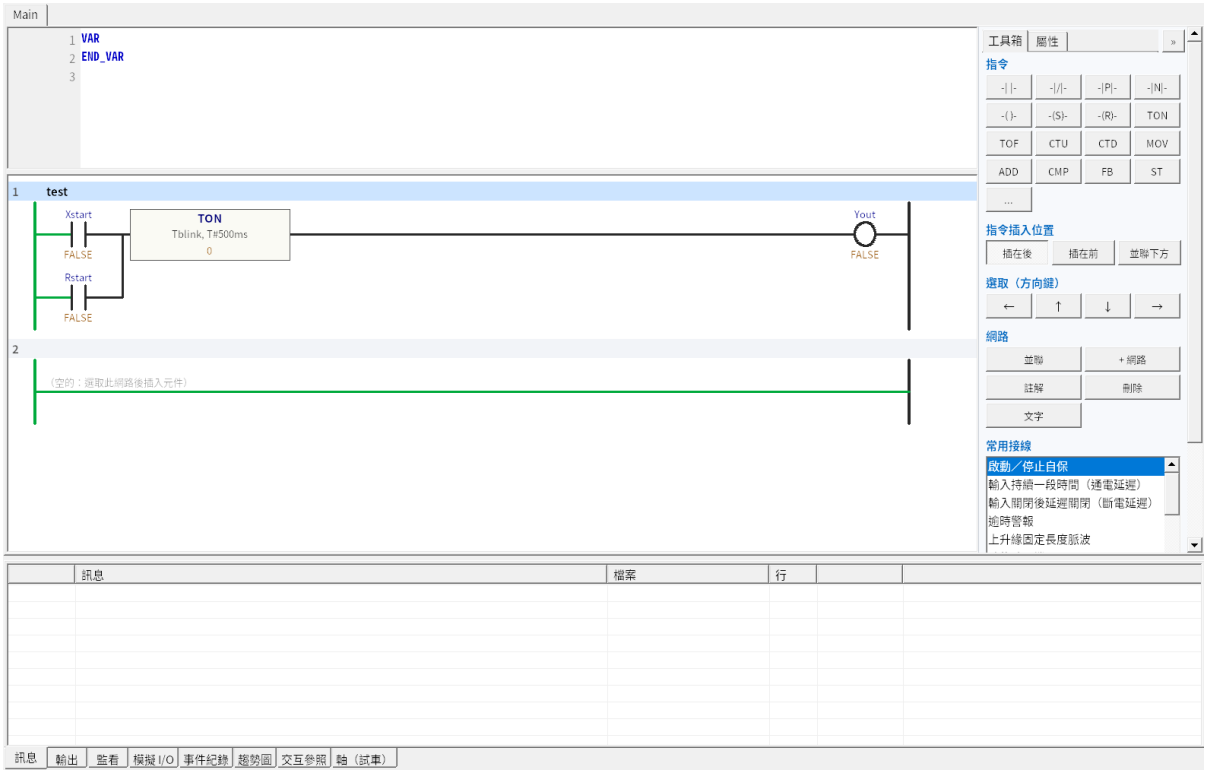


Figure 10-11 Online view: the ladder diagram shows the live power flow on the controller



Figure 10-12 The status bar when online: ONLINE COM4: RUN and the scan time

If the controller is not running the program from this build, 86LDmicro refuses to connect and shows "The controller runs another program (or another version of this one)… Generate the code, upload it, then go online". Upload again (F8).



Figure 10-13 The message when the program on the controller differs from this build

10.4.3 Operations when online

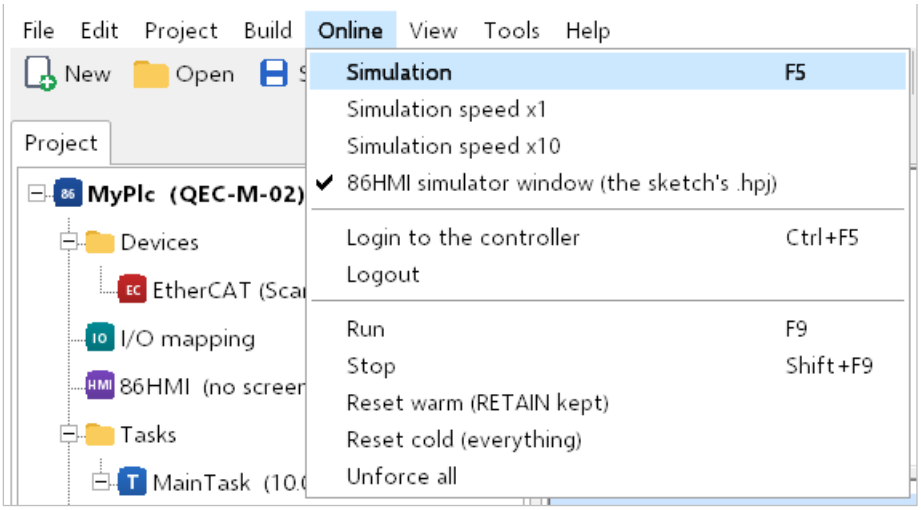


Figure 10-14 The "Online" menu when online

Item	Shortcut	Purpose
Login to the controller	Ctrl+F5	Starts the online view
Logout	—	Ends the online view (the controller keeps running)
Run	F9	The program continues running
Stop	Shift+F9	The program stops running; I/O and EtherCAT keep updating, outputs hold their last values; all Motion86 machines stop and jogging ends
Reset warm (RETAIN kept)	—	Variables return to their initial values, RETAIN variables are kept, and all forces are released; the RUN/STOP state does not change (in RUN, the program restarts from the initial values)
Reset cold (everything)	—	All variables (including RETAIN) return to their initial values
Unforce all	—	Releases all forces

*__Safety:__ STOP only stops running the program; **it is not a safe stop**: outputs hold their last values. Implement the machine's emergency stop with a hardware safety circuit.

Watching, writing, forcing and Trace work the same way as in simulation (sections 9.3 and 9.4):

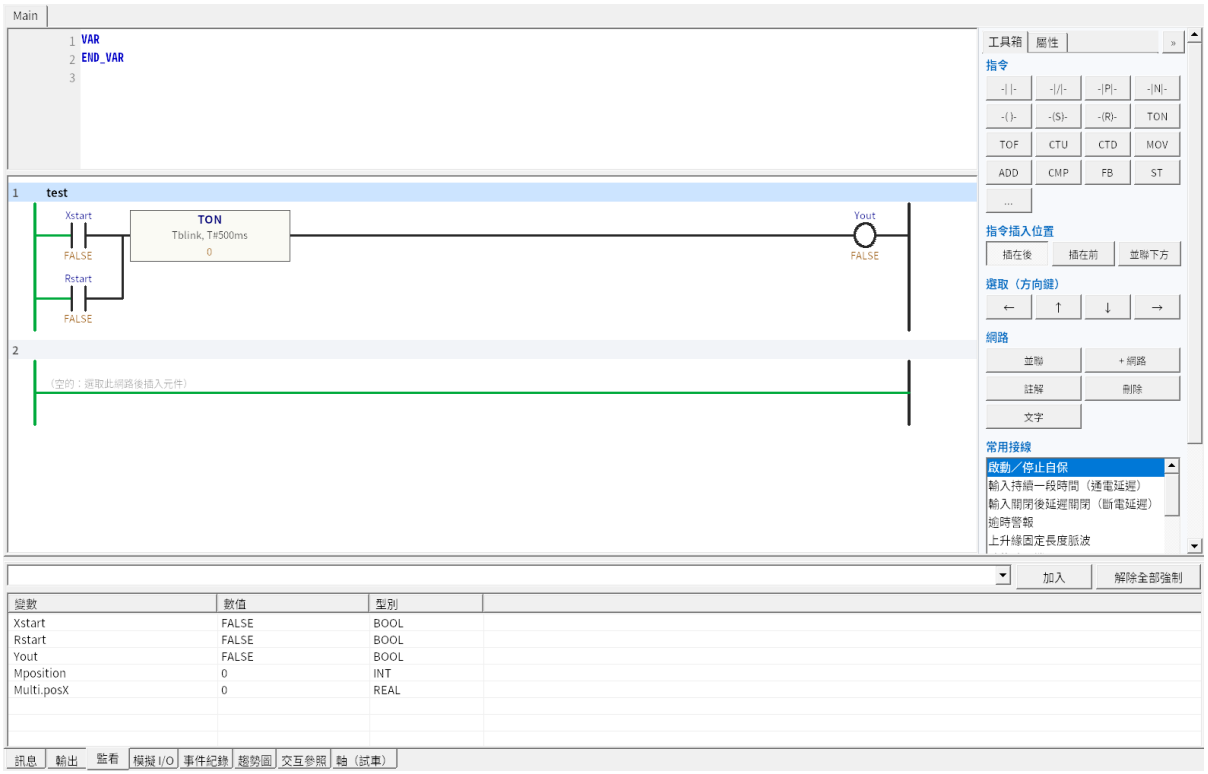


Figure 10-15 Online: the ladder diagram and the Watch window show the values on the controller

- **Power flow and values**: green segments in ladder diagrams and the values below the elements; green steps in SFCs; in ST, the value when you hover over a variable.
- **Watch window**: double-click to write once; right-click to force (shows `[F]`), unforce, or add to the trace.
- **EtherCAT**: the SubDevice page shows the live value of each PDO entry (section 11.6).
- **Axes**: the machine page, axis page and "Axes" window operate the axes on the controller (Chapter 12).
- **Logbook**: the controller's logbook (section 10.5).

***Note:** When the connection is lost for 3 seconds (USB cable unplugged, PC goes to sleep, etc.), the controller releases all forces automatically.

10.5 Logbook and watchdog

10.5.1 Logbook

The "**Logbook**" window (a tab of the lower tool window) shows the controller's logbook: the last 32 events, each with a time (seconds since the controller started) and a message. During simulation it shows the simulator's log, and the time column shows "Simulation".

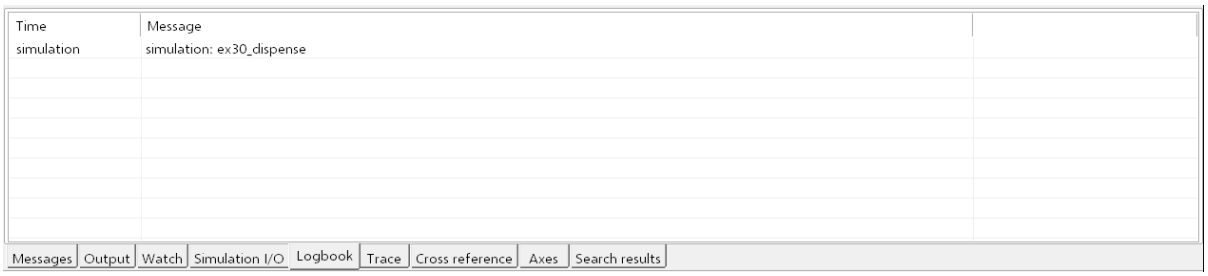


Figure 10-16 Logbook: EtherCAT MainDevice startup messages and the program starting

Event	Example
Power-up, program start	`start: MyPlc (build 1A2B3C4D)` (the ID of this build in parentheses)
RETAIN	`RETAIN: the values kept are loaded`, `RETAIN: nothing stored (first start, or RETAIN changed): initial values`
RUN/STOP (from the PC)	`RUN (from the PC)`, `STOP (from the PC)`
Warm restart/cold restart	`warm restart: the variables to their initial values, RETAIN kept`, `cold restart: every variable to its initial value`
Connection lost	`the PC is gone: the forces are released`
Motion86	`Motion86: an axis' drive left operation enabled: the machine stopped (MC_Reset, MC_Power again)`, etc. (section 12.9)
Scan overrun	`task MainTask: a scan took 12500 us (the interval is 10000 us)` (logged the first time and every 1000 times)
Watchdog	`MainTask watchdog: STOP, 152 ms (3 other), Motion 140 ms`
EtherCAT	`EtherCAT: running, 2 slaves, 1000 us, sync, …`, `EtherCAT: begin failed`, `no slave N`
Runtime error in simulation	`STOP: Main line 12: …` (simulation only)

10.5.2 Watchdog

The watchdog is a scan overrun check: each cyclic task can have a "Watchdog (us)" setting; when the program time in one scan exceeds this value, the PLC stops (STOP). 0 means no watchdog. Double-click the task under "Tasks" in the project tree to set it (section 4.3).

> ***Note: The watchdog checks the time only at the end of each scan, after the programs return. If a program enters an infinite loop during the scan and never returns, the watchdog cannot interrupt it; it is not an independent hardware watchdog and cannot replace the machine's safety circuit.**
> Do not test infinite loops on a running machine; a long loop of finite length can be used to confirm that the watchdog stops the PLC.

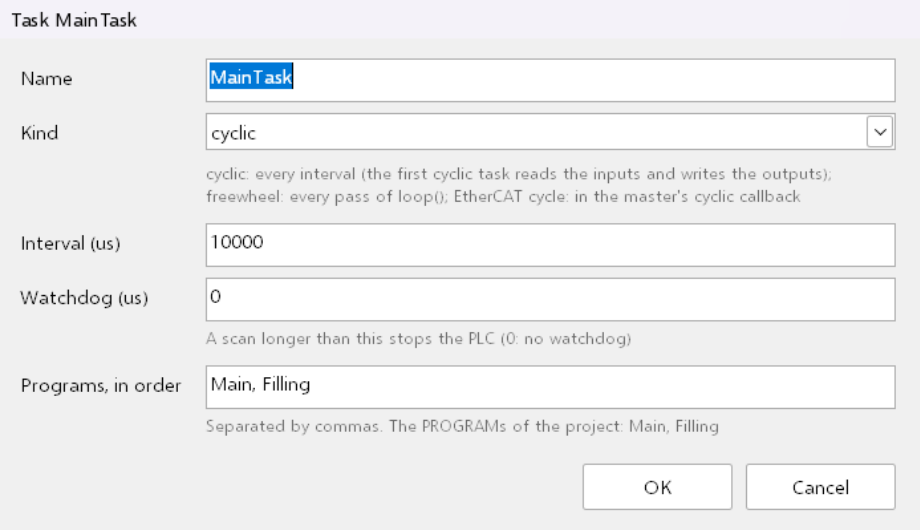


Figure 10-17 Task settings: interval and Watchdog (us)

How the controller applies the watchdog:

- It measures **the time of the programs themselves in one scan** of that task: the whole scan time minus the time that does not belong to the programs -- EEPROM writes (PERSIST; the 86Duino's EEPROM occasionally has to erase a flash sector, which can pause for several hundred ms) and EtherCAT cycle interrupts (Motion86, EtherCAT tasks).
- RETAIN writes happen between scans, and 86HMI screen updates run after the first task's scan; neither is **counted** in the scan time.
- The **first 3 scans** after power-up are not checked (loading RETAIN, first initialization of variables).
- When the setting is exceeded: the PLC goes to STOP (the program no longer runs and outputs hold their last values), all Motion86 machines stop, and a watchdog entry is written to the Logbook.
- The simulator does not check the watchdog; verify it on the controller.

Format of the watchdog entry in the Logbook:

```
MainTask watchdog: STOP, 152 ms (3 other), Motion 140 ms
```

Field	Meaning
`MainTask`	The task that overran
`152 ms`	Total time of this scan
`(3 other)`	The non-program time in it (ms), such as EEPROM writes and EtherCAT interrupts, which is not counted
`Motion 140 ms`	The **slowest program** in this scan and its time: look for the cause in this program first

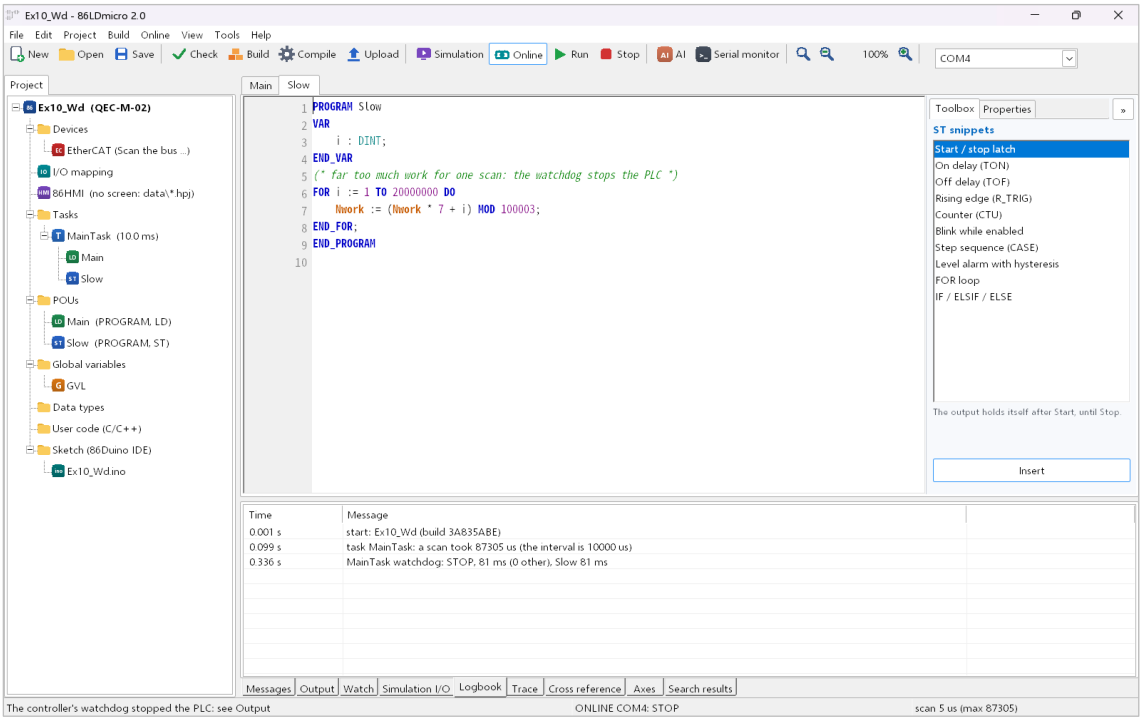


Figure 10-18 A watchdog entry in the Logbook

Remedy	Description
Reduce the computation in each scan	Spread long loops over several scans (for example, 100 items per scan); compute only when needed
Move to a slower task	Put programs that do not need to run every 10 ms (statistics, recipes, text) in a 100 ms task; this only lowers how often they run and does not shorten a single scan, so a scan that is too long must still be spread over several scans
Increase the watchdog time	After confirming that the scan time is reasonable, set the watchdog to several times the normal maximum scan time
Restart	After removing the cause, press RUN (F9), or upload again

***Tip:** When online, the status bar shows the time of each scan and its maximum ("scan … ms (max …)") and the number of overruns, which you can use to decide the watchdog and task interval.

10.6 Uploading the 86HMI resources

When the 86HMI Editor does "Code Generation", besides `myhmi.cpp/myhmi.h` it also writes the images, fonts and screen data used by the screens into the sketch's `data\Resource`. These **resources** are stored on the controller, separately from the program (Chapter 13).

Situation	What to do
First upload, or both the program and the screens changed	Click **Upload** (F8): the program and `data\Resource` are uploaded together
Only images, fonts or screen appearance changed; the PLC program did not	Click **Upload the 86HMI resources**: only the resources are sent, nothing is compiled, and the program on the controller is unchanged (faster than F8)

1. Check the COM port on the toolbar (for example COM4).
2. Choose "Build > Upload the 86HMI resources (data\Resource)", or click **Upload resources** on the 86HMI page.
3. "Build output" shows "== Uploading the resources to COM4: …\data\Resource ..." and the v86dude progress, and finally "== Resources uploaded".
4. If you were online, it goes offline automatically before the upload and reconnects automatically when done.

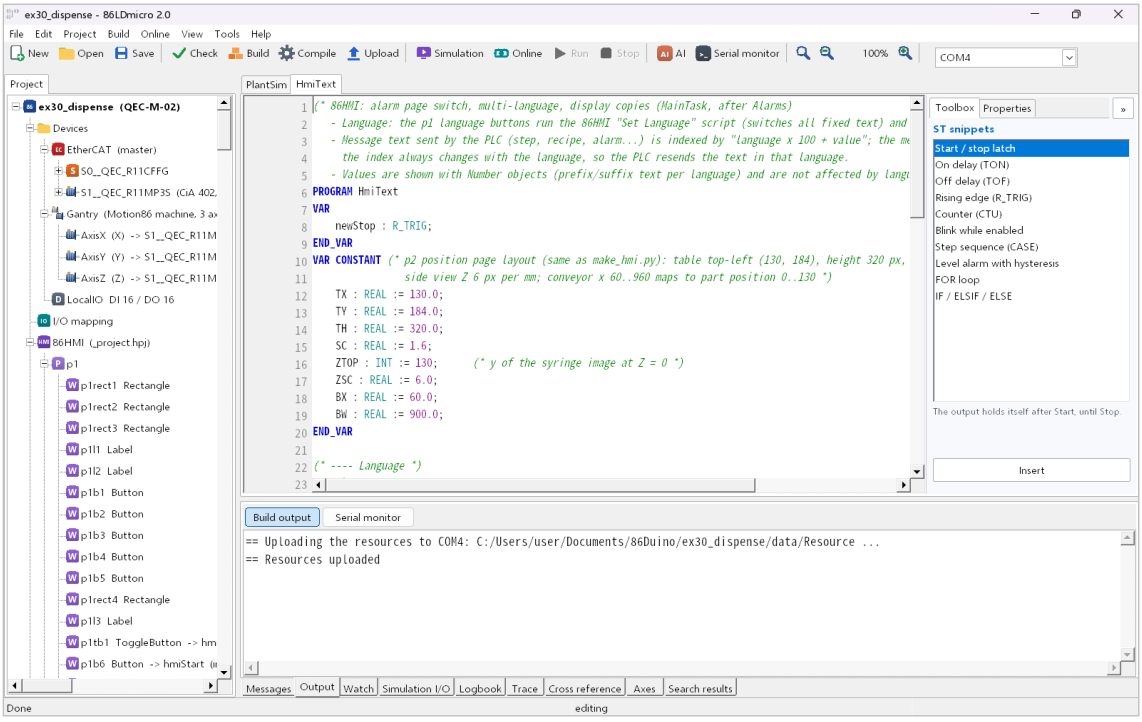


Figure 10-19 Uploading the 86HMI resources: the build output shows the upload progress and "Resources uploaded"

***Tip:** This function is the same as "Tools > Upload resource files to QEC M02" in the 86Duino IDE. If "No resources to upload" appears, the sketch has no `data\Resource` or it is empty: first open the .hpj in the 86HMI Editor and click "Code Generation". Command line: `86ldmicro-cli upload-res <project> <COM>` (section 15.5).

Chapter 11 EtherCAT

The QEC-M-02 is an EtherCAT MainDevice. An 86LDmicro 2.0 project can have its own EtherCAT MainDevice: scan the network, configure the SubDevices and export the ENI with the built-in EtherCAT Configurator, read the ENI back into the project, and map each PDO entry to a variable; the program can then read and write remote I/O and servo drives directly. This chapter uses `tools\example\ex30_dispense` (QEC-R11CFFG I/O module + QEC-R11MP3S three-axis drive) as the example.

11.1 Concepts

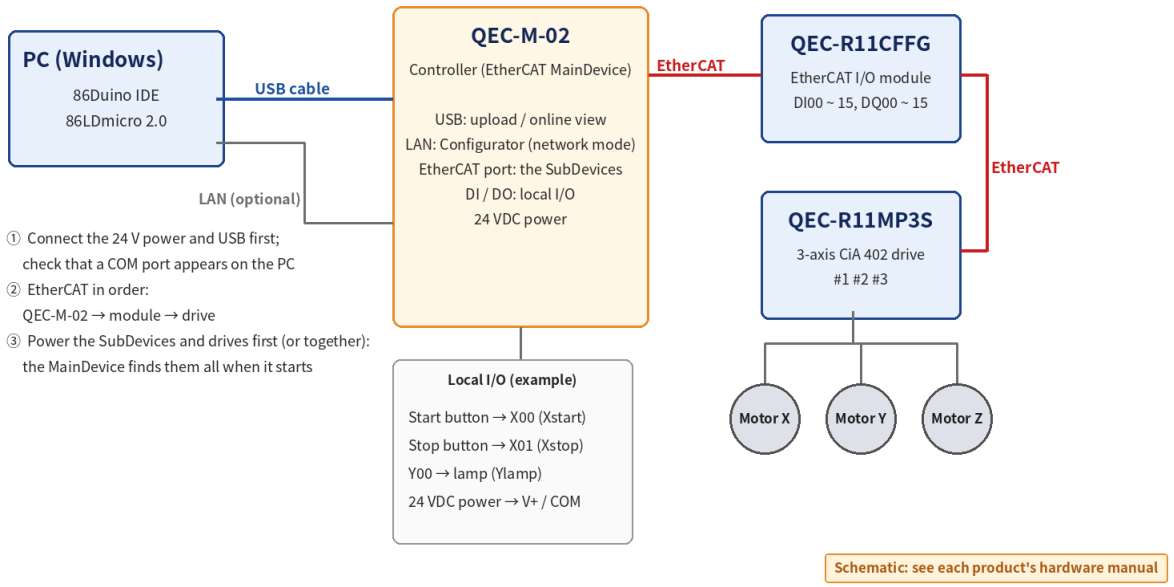


Figure 11-1 A typical system: the PC connects to the QEC-M-02 over USB, and EtherCAT chains the I/O module and the drive (schematic)

Term	Description
MainDevice	The QEC-M-02 controller, which exchanges data with all SubDevices every EtherCAT cycle
SubDevice	I/O modules, servo drives, etc. on the EtherCAT network, for example QEC-R11CFFG, QEC-R11MP3S
PDO (process data)	Data exchanged every cycle: RxPDO goes from the MainDevice to the SubDevice (outputs), TxPDO goes from the SubDevice back to the MainDevice (inputs)
SDO/CoE	Acyclic parameter read/write (object dictionary, Index:Sub), for example drive parameters
ESI	Description file supplied by the SubDevice vendor (.xml): object dictionary, PDOs, icons, module definitions
ENI	Network configuration file (.xml): which SubDevices exist, which PDOs are exchanged, DC synchronization, startup parameters; stored on the controller in `C:/eni/`
Configurator	The QEC's EtherCAT configuration tool (web interface): Scan, Configure, PDO, DC, CoE, Export ENI
DC (distributed clocks)	Makes all SubDevices act at the same instant; recommended for servo drives (CSP)

86LDmicro supports two ways of using EtherCAT:

Method	Description	When to use
EtherCAT network (ENI)	The project's own MainDevice, with the network described by an ENI; supports any EtherCAT SubDevice, CiA 402 drives, Motion86 CSP	Recommended in 2.0 (this chapter)
86EVA	The 1.x way: the 86EVA tool generates `myeva.h/cpp`, which manage the MainDevice and virtual I/O	Continuing 1.x projects

***Note:** The two methods cannot be used at the same time (both drive the same MainDevice). When switching to ENI, remove the 1.x `#include "myeva.h"` and `EVA.begin()` from the .ino, and move `myeva.cpp`/`myeva.h` out of the sketch folder (the build will warn about them).

11.2 The complete workflow

From an empty network to a program that reads and writes PDOs, complete the following steps in order. The first five steps are done in the Configurator (section 11.3); after that you return to 86LDmicro.

Step	Where	What to do	Section
1	86LDmicro EtherCAT page	Select the connection type and click **Upload Configurator** (only once)	11.3.1
2	Configurator	(Recommended) Upload and load the SubDevices' ESI	11.3.5
3	Configurator	**Scan** for SubDevices -> **Configure**	11.3.3
4	Configurator	Set PDO, DC, Startup and Slots as needed; test with **Start**	11.3.6
5	Configurator	**Export ENI** -> **Upload** to save it on the controller in `C:/eni/`	11.3.5
6	86LDmicro	**Take it into the project**: the SubDevices and PDOs are loaded into the project	11.4
7	86LDmicro	Map the PDOs to variables; create Motion86 machines for CiA 402 drives	11.4, Chapter 12
8	86LDmicro	Build and upload the PLC program (replacing the Configurator); check the MainDevice status	11.4.4
9	86LDmicro SubDevice page	View PDO values online, write/force, SDO, CoE object dictionary	11.5, 11.6

11.2.1 Step 1: Open the EtherCAT page and upload the Configurator

Choose "Project > EtherCAT (Scan, PDO, ENI)", or double-click "Devices > EtherCAT" in the project tree. Select the connection type and click **Upload Configurator**.

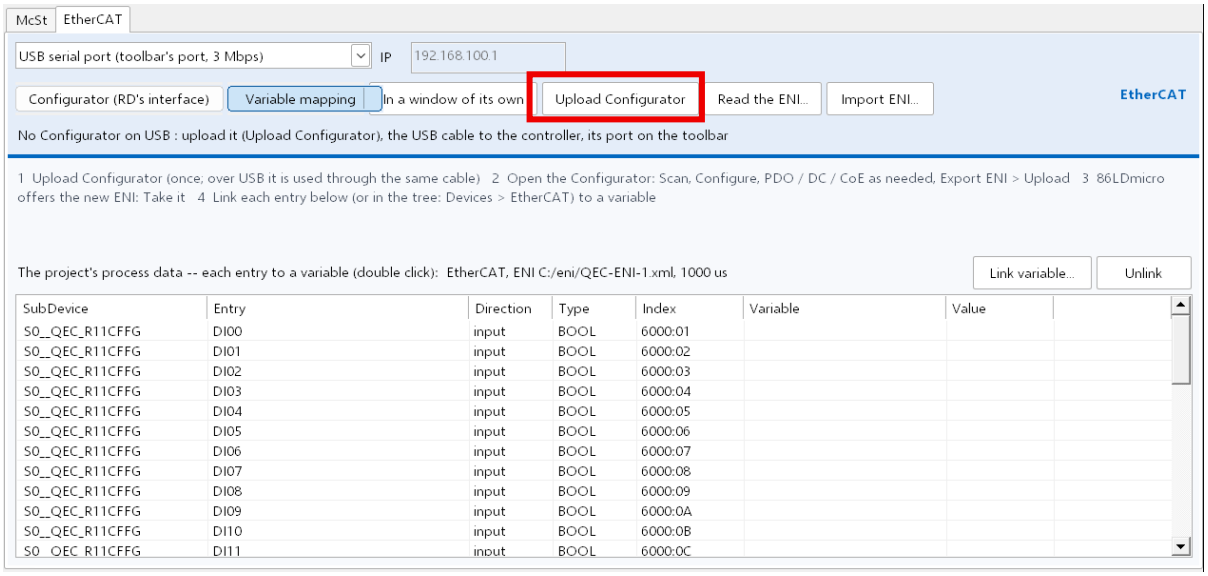


Figure 11-2 The EtherCAT page: connection type, Configurator (RD's interface), Variable mapping, Upload Configurator, Read the ENI, Import ENI

11.2.2 Step 2: Open the Configurator in the tab

Click **Configurator (RD's interface)**; the Configurator web page is shown directly in the EtherCAT page.

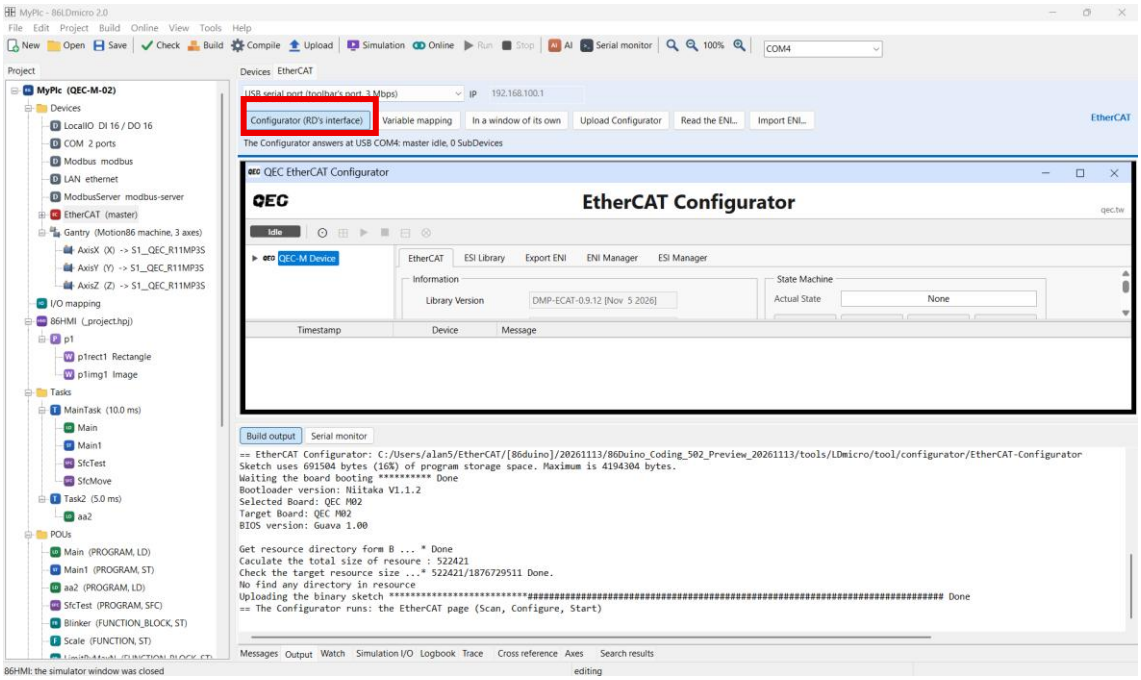


Figure 11-3 The Configurator web page embedded in the EtherCAT page (USB mode)

You can click **In a window of its own** to show the Configurator in a separate page.

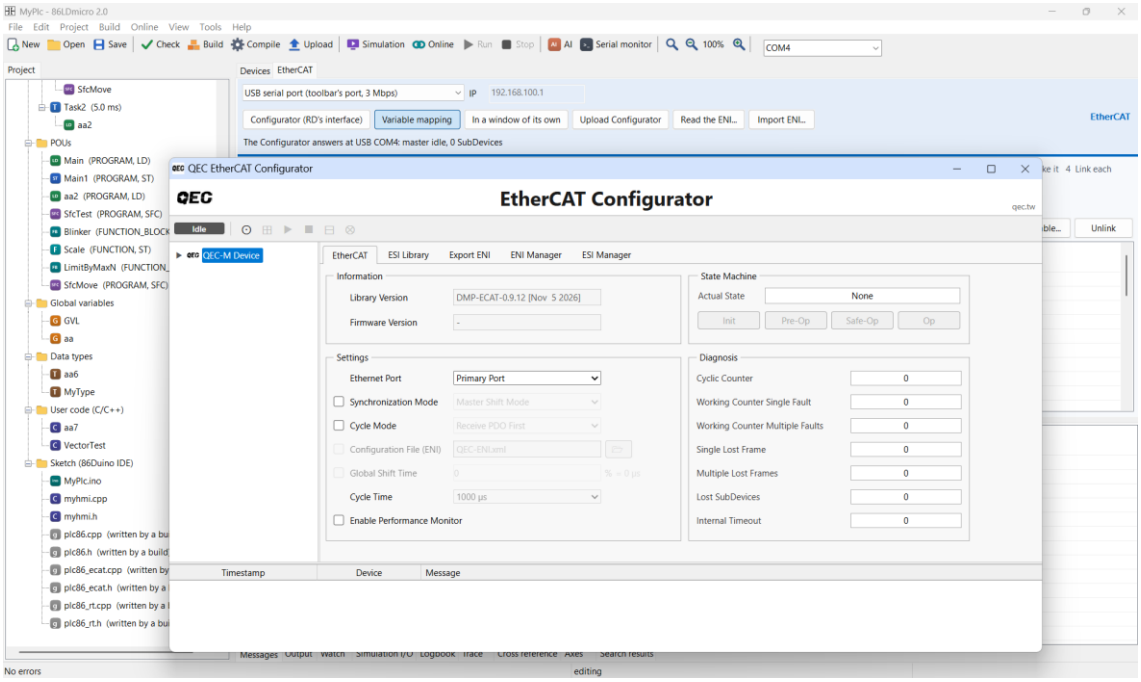


Figure 11-3-1 The Configurator web page in its own window (USB mode)

11.2.3 Steps 3 and 4: Scan, Configure, test with Start

In the Configurator, click **Scan**; the device tree on the left lists all SubDevices.



Figure 11-4-1 Status label Idle (not scanned yet)

Click **Configure** to read the PDOs.



Figure 11-4-2 Status label Init

After you click **Start**, all SubDevices enter the OP state.



Figure 11-4-3 Status label Config (Configure done)

All SubDevices enter the OP state.

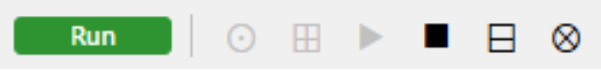


Figure 11-4-4 Status label Run (cyclic exchange running)

The device tree on the left lists all SubDevices.

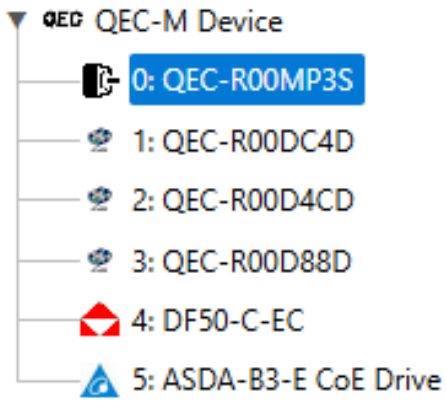


Figure 11-4-5 After Scan: the device tree lists each SubDevice

View the live values on the SubDevice's **Online** tab.

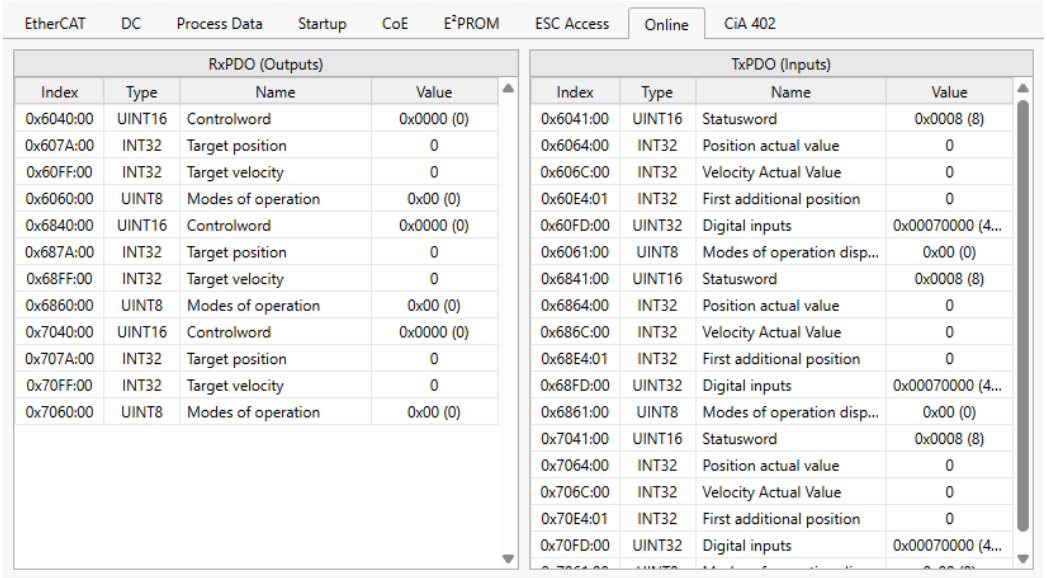


Figure 11-4-6 After Start: the SubDevice's Online tab shows live PDO values

11.2.4 Step 5: Export the ENI and upload it to the controller

Click the **QEC-M Device** > **Export ENI** tab.

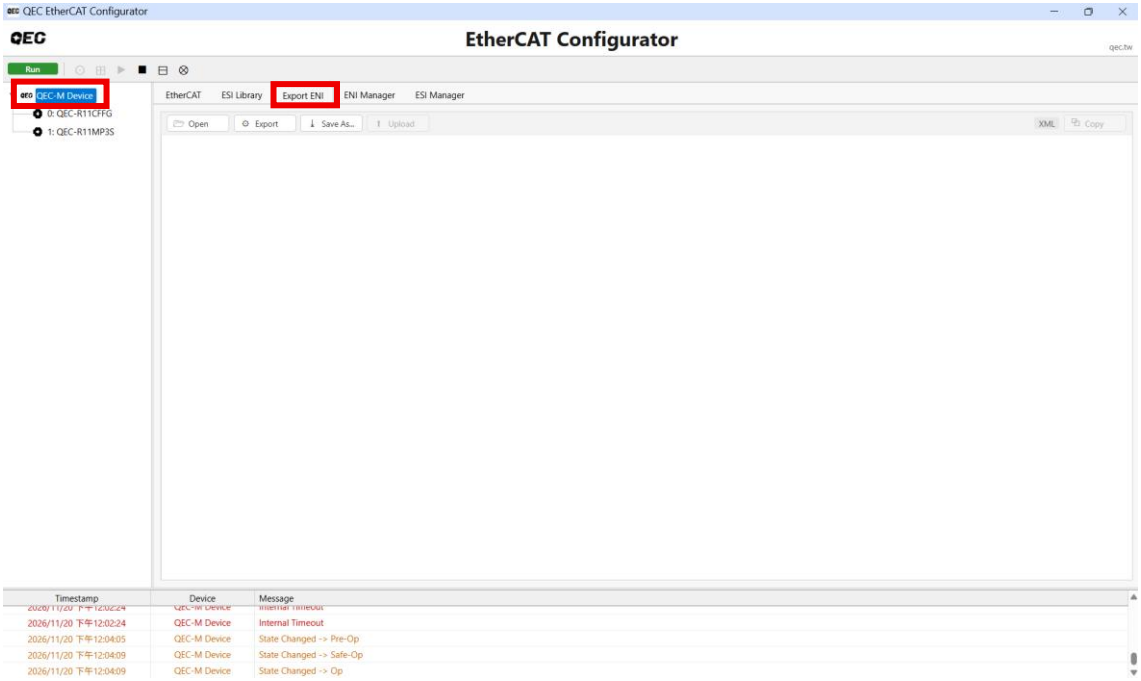


Figure 11-5-1 The Export ENI tab

Click **Export** to generate the ENI.

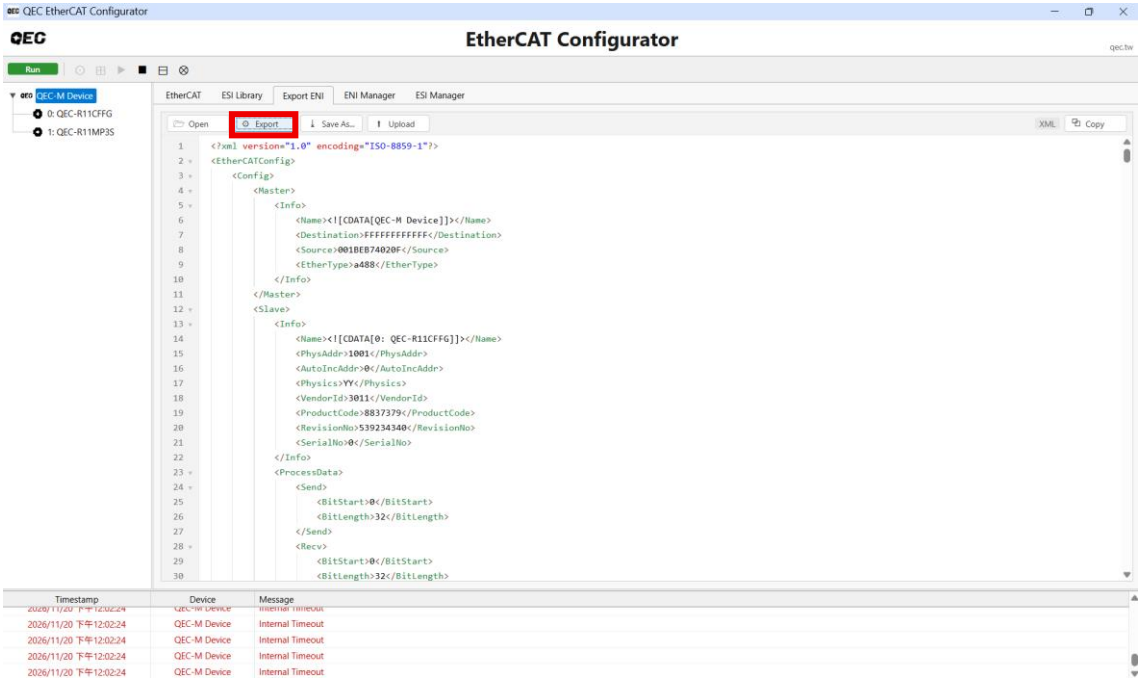


Figure 11-5-2 Click Export to generate the ENI

Then click **Upload**, enter a file name and upload it.

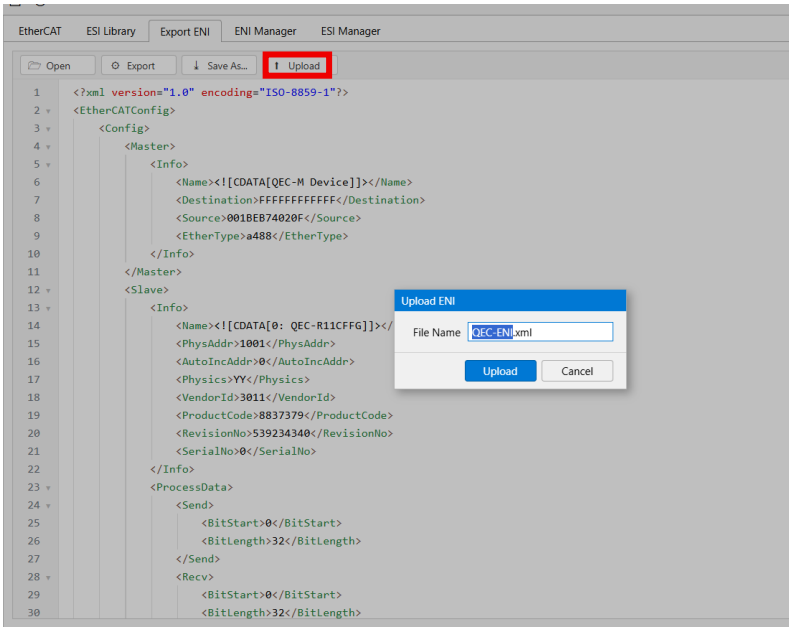


Figure 11-6 Upload ENI: enter a file name; it is saved on the controller in C:/eni/

11.2.5 Steps 6 and 7: Take the ENI into the project and map variables

Back in 86LDmicro, a prompt appears when a new ENI is detected on the controller; click **Take it into the project.** The SubDevice names from the ENI then appear under the EtherCAT category of the project tree on the left.

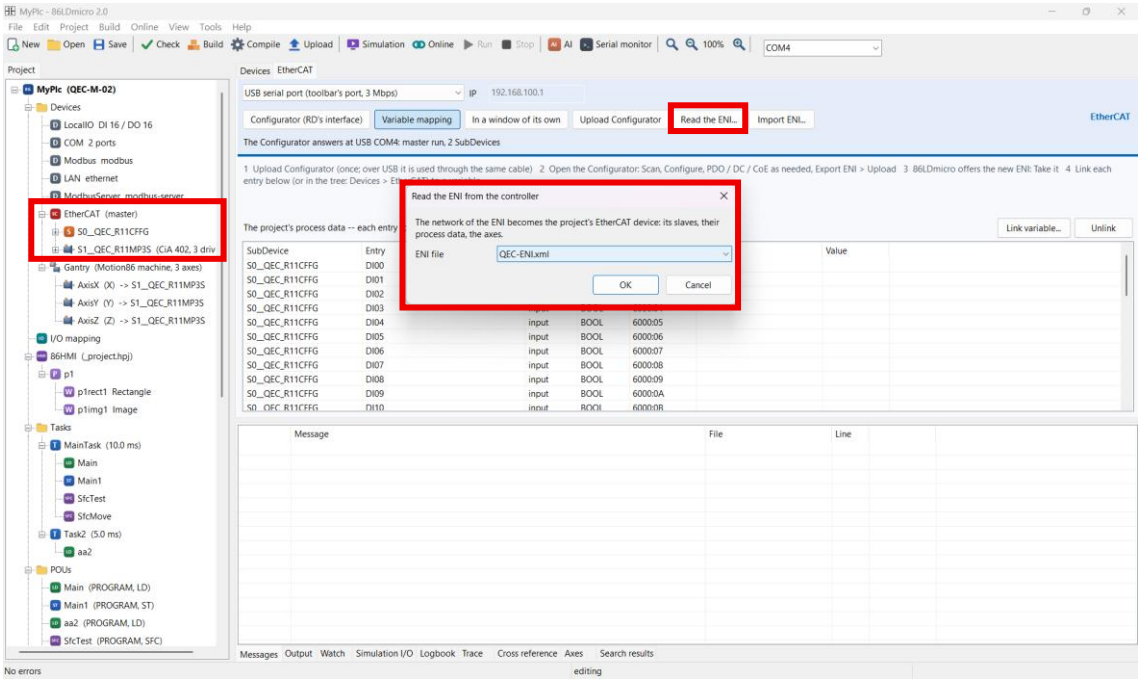


Figure 11-7-1 A new ENI detected on the controller: Take it into the project

Then go to "Variable mapping" and map each PDO entry to a variable.

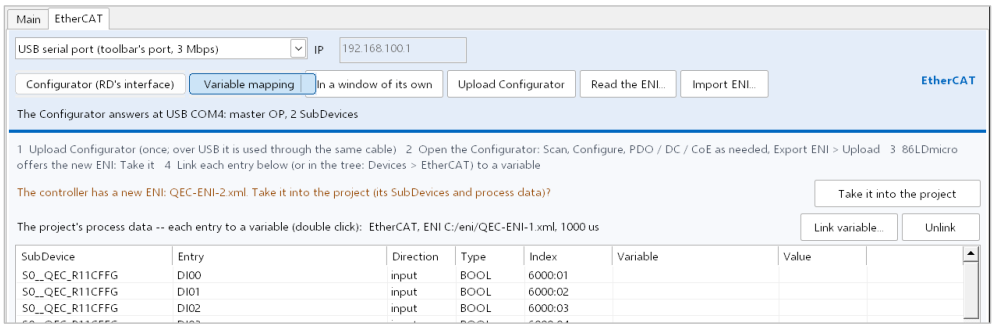


Figure 11-7-2 A new ENI detected on the controller: Take it into the project

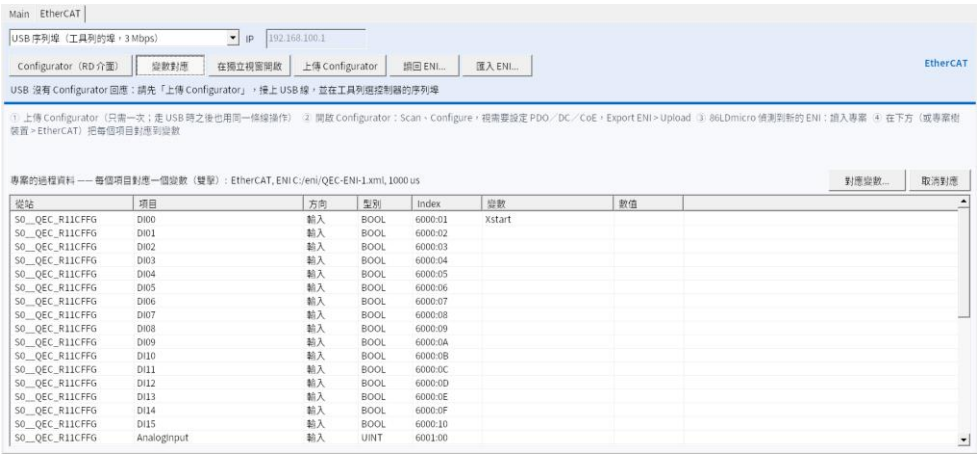


Figure 11-8 "Variable mapping": one row per PDO entry; mapped entries show their variable

11.2.6 Steps 8 and 9: Upload the PLC program and check

Click **Upload** (F8).

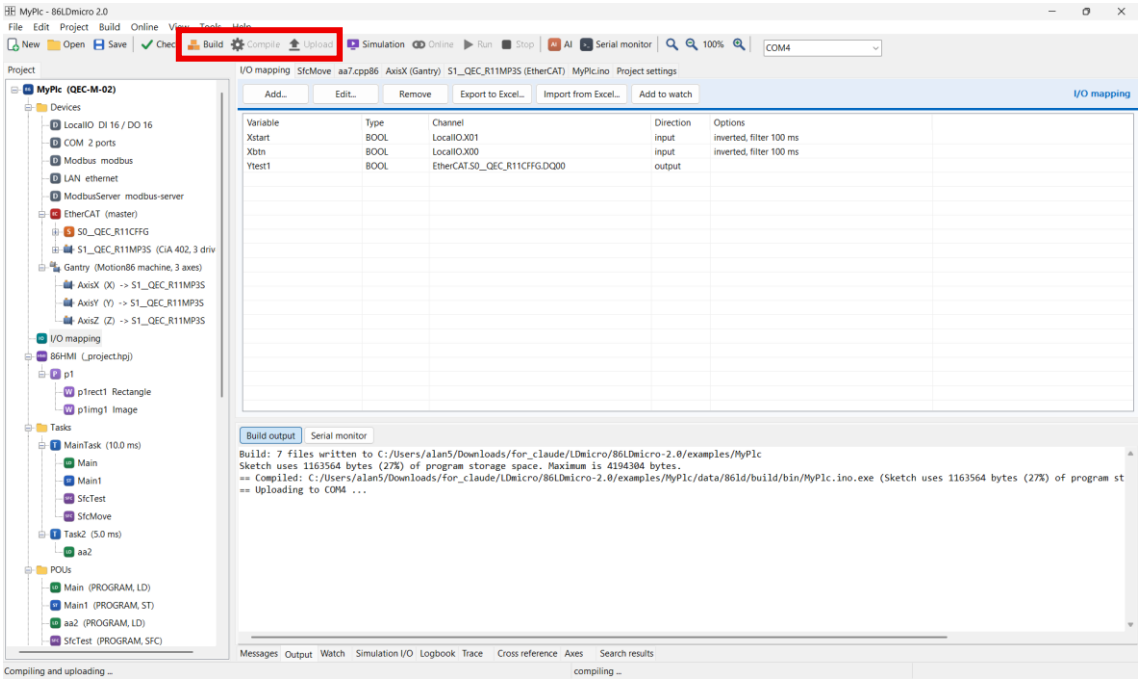


Figure 11-9-1 Upload

The PLC program replaces the Configurator, and at power-up the MainDevice starts with the ENI. Open the "Serial monitor" to check the MainDevice status, and after going online, view the live PDO values on the SubDevice page.

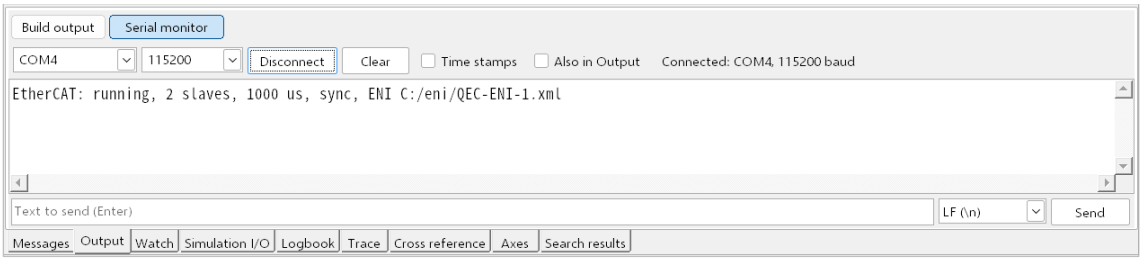


Figure 11-9 Serial monitor: EtherCAT: running, 2 slaves, 1000 us, sync

11.3 EtherCAT Configurator

The EtherCAT Configurator is a configuration program that runs on the QEC controller and is operated through a web page. 86LDmicro uploads it to the controller and embeds the web page directly in the EtherCAT page; once the network is configured, you export the ENI and hand it back to 86LDmicro.

11.3.1 Uploading and opening

Connection type	Description
USB serial port (toolbar's port, 3 Mbps)	Default. Uses the same USB cable as uploading and the online view; 86LDmicro provides the web page on the PC, and only the API goes over the serial port to the controller
Ethernet (the controller's IP)	Connect the PC to the controller's LAN port and enter the controller's IP (default `192.168.100.1`)

1. Select the connection type on the EtherCAT page (for USB, select the controller's COM port on the toolbar).
2. Click **Upload Configurator** (only once). 86LDmicro compiles and uploads the Configurator program, which **temporarily replaces** the PLC program on the controller. You can also use "Tools > EtherCAT > Upload the Configurator to the controller...".
3. After the upload, the page automatically detects whether the Configurator answers and the MainDevice status, for example "The Configurator answers at COM4: master Idle, 0 SubDevices".
4. Click **Configurator (RD's interface)**: the Configurator web page is shown in this tab (embedded with Microsoft Edge). Click **In a window of its own** to detach it into a separate window (for example, to drag it to a second monitor); click **Variable mapping** to return to the PDO list.

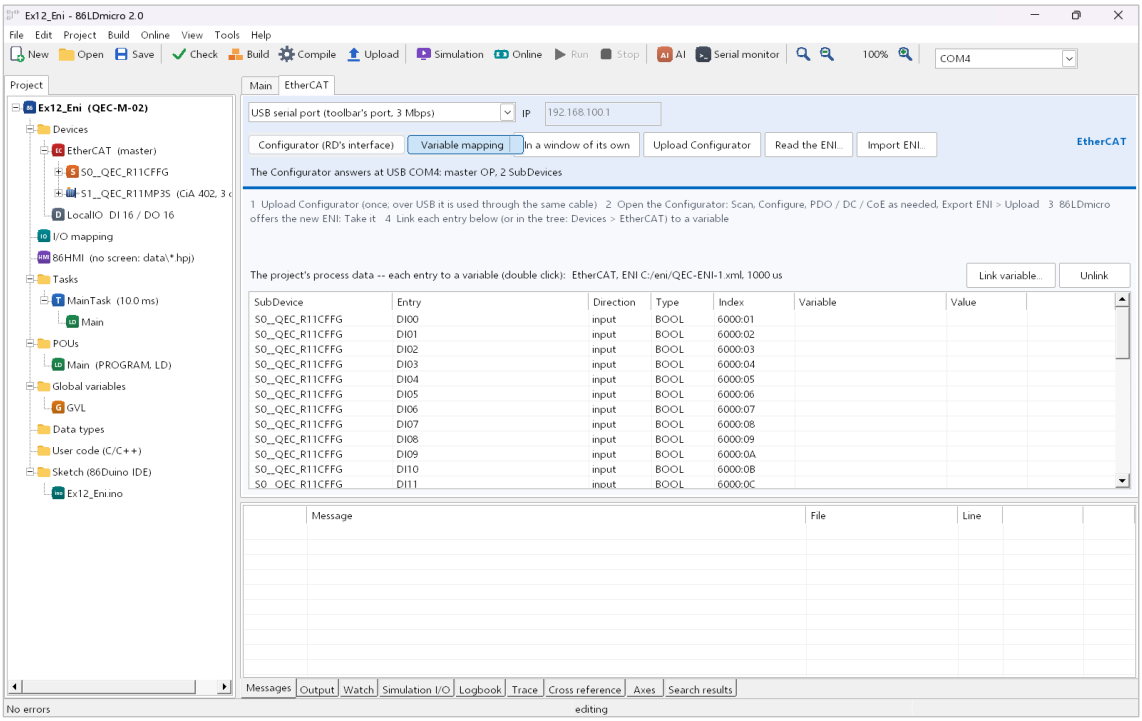


Figure 11-10 Upload Configurator: the progress in the build output and the response message on the page when done

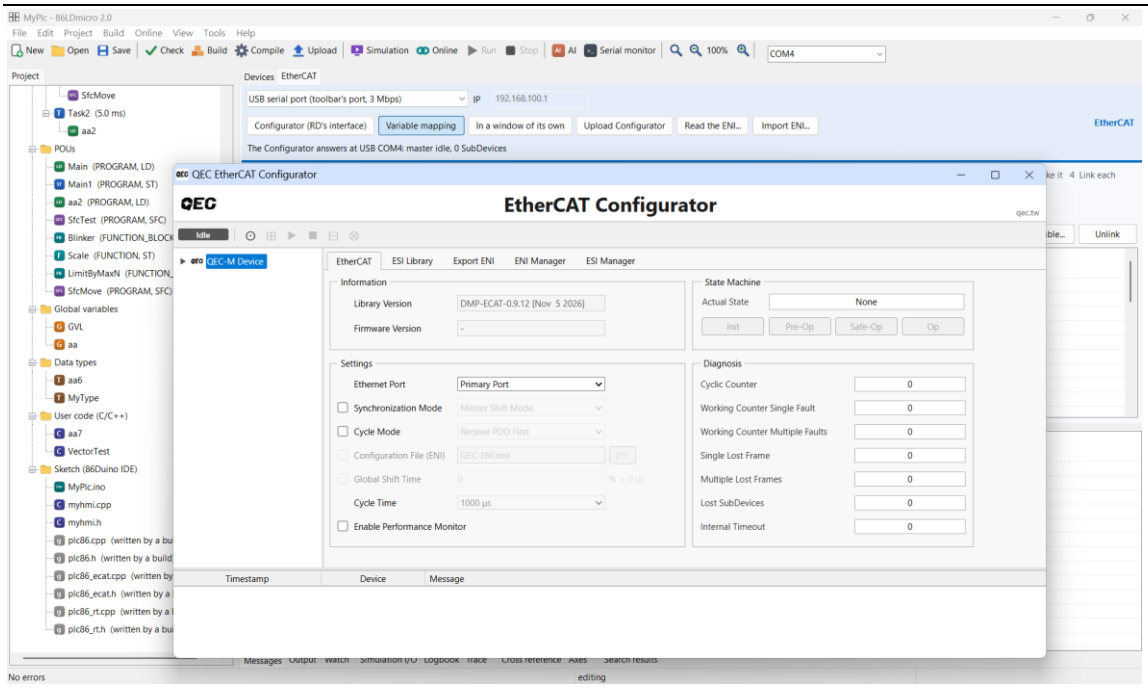


Figure 11-11 "In a window of its own": the Configurator shown in a separate window

*__Tip:__ In USB mode, uploading or the online view automatically pauses the Configurator connection; when the EtherCAT page is not in the foreground, it does not hold the serial port either (released after 2 seconds idle). In network mode you can also open it in a browser: "Tools > EtherCAT > The Configurator's web page (browser)...".

11.3.2 The screen

The Configurator screen is divided into five areas from top to bottom:

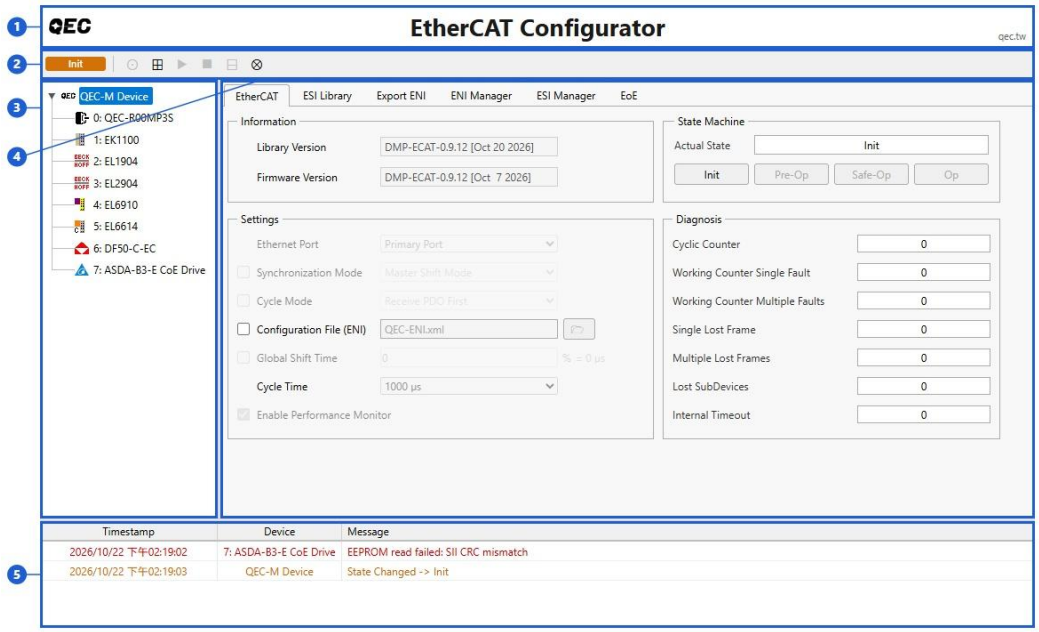


Figure 11-12 The Configurator screen: (1) title (2) main toolbar (3) device tree (4) tab content (5) log

Area	Content
(1) Title	QEC logo and title
(2) Main toolbar	MainDevice status and six action buttons: Scan, Configure, Start, Stop, Deconfigure, Disconnect
(3) Device tree	The MainDevice (QEC-M Device) and the scanned SubDevices
(4) Tab content	Settings tabs of the node selected in the device tree (the tabs depend on the kind of node); when there are too many tabs, scroll left and right with the mouse wheel, or click ▾ on the right to list all tabs
(5) Log	Operation log and system events (11.3.7)

11.3.3 Main toolbar and states

The status label on the left of the toolbar shows the current state of the MainDevice: **Idle**, **Init**, **Config**, **Run**, **Bootstrap**. Each button can be clicked only in its corresponding state; otherwise it is grayed out.

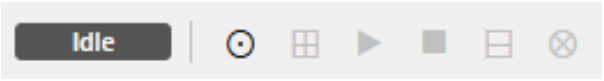


Figure 11-13 Status label Idle (not scanned yet)

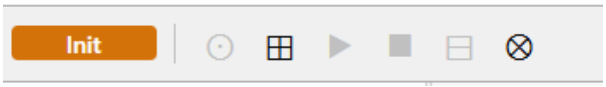


Figure 11-13-1 Status label Init



Figure 11-14 Status label Config (Configure done)



Figure 11-15 Status label Run (cyclic exchange running)

Button	Icon	Description
Scan	⊙	Scans all SubDevices on the bus and creates nodes in the device tree
Configure	⊞	Configures each SubDevice according to the ENI file (or the online configuration) and reads the PDO data
Start	▶	Starts the cyclic process data exchange; the SubDevices enter Op
Stop	■	Stops the cyclic exchange and returns to Safe-Op
Deconfigure	⊟	Undoes the configuration, releases the CiA 402 axis objects, and returns to Init
Disconnect	⊗	Disconnects from the bus and clears the device tree

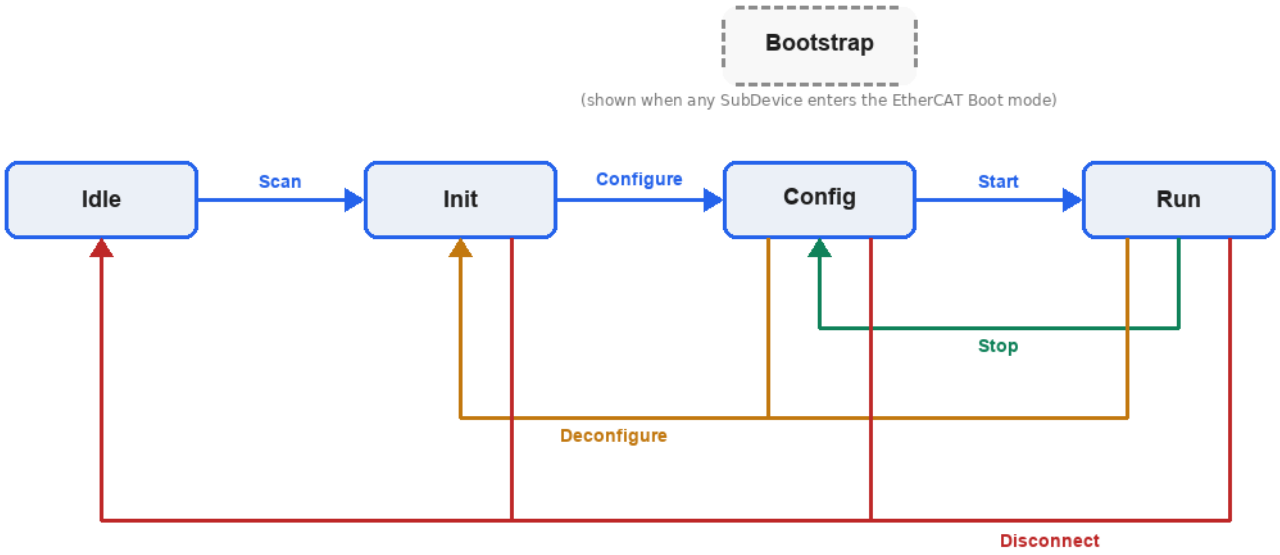


Figure 11-16 MainDevice state transitions: boxes are states, arrows are the buttons pressed

Right-clicking **QEC-M Device** in the device tree offers the same six actions. After you start an action, a progress window appears (for example Scan Network), showing `Step: current step / total steps`, a progress bar and the device being processed; it closes automatically when done (or failed).

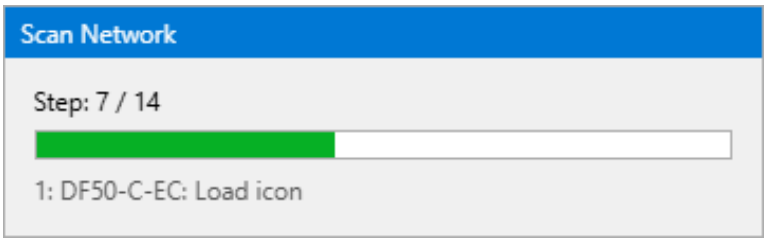


Figure 11-17 The progress window

11.3.4 Device tree

- Before Scan there is only one node, **QEC-M Device** (the MainDevice).
- After a successful Scan, there is one node per SubDevice, named `number: SubDevice name`, for example `0: QEC-R11CFFG`, `1: QEC-R11MP3S`.
- Click a node to show its settings tabs in the middle; right-click a SubDevice > Rename to rename it.

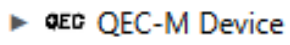


Figure 11-18 The device tree before Scan: only QEC-M Device

11.3.5 Tabs of QEC-M Device (the MainDevice)

When **QEC-M Device** is selected, the tabs are EtherCAT, ESI Library, Export ENI, ENI Manager and ESI Manager; there is an extra EoE tab when a SubDevice supports EoE, and an extra Performance tab in Run when performance monitoring is enabled.

EtherCAT tab: the left half has Information (library and firmware versions) and Settings, the right half has State Machine and Diagnosis (diagnostic counters).

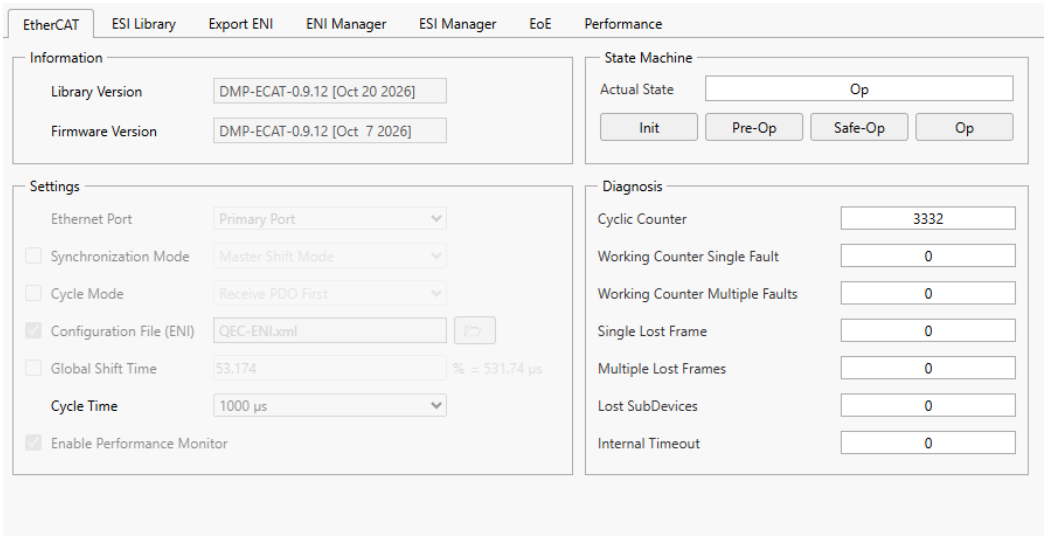


Figure 11-19 The EtherCAT tab of QEC-M Device: settings, state machine and diagnostic counters

Setting	Description	When it can be changed
Ethernet Port	Primary Port/Secondary Port/Cable Redundancy	Idle
Synchronization Mode	Master Shift Mode/Bus Shift Mode (used at Scan)	Idle

Setting	Description	When it can be changed
Cycle Mode	Receive PDO First/Send PDO First	Idle
Configuration File (ENI)	When checked, enter a path or click 🗁 to select an ENI on the controller; used at Configure	Init
Global Shift Time	Entered as a percentage, with the equivalent µs shown next to it (used at Start)	Config
Cycle Time	125 to 10000 µs	Config
Enable Performance Monitor	When checked, a Performance tab appears in Run	Idle

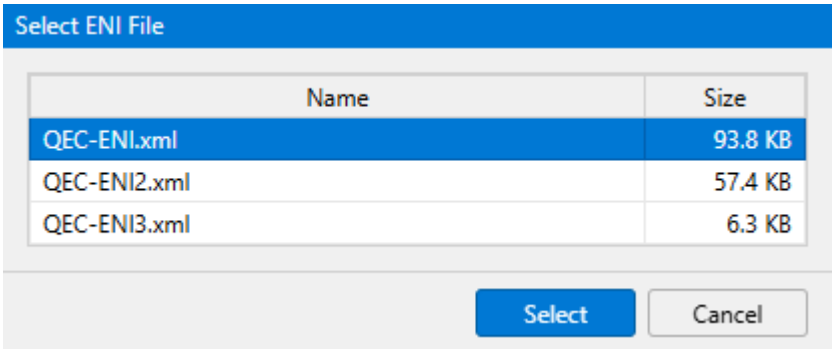


Figure 11-20 Select ENI File: select an ENI file on the controller

The `Init`/`Pre-Op`/`Safe-Op`/`Op` buttons of State Machine force all SubDevices into that state; Diagnosis shows counters such as Cyclic Counter, Working Counter errors, lost frames and lost SubDevices.

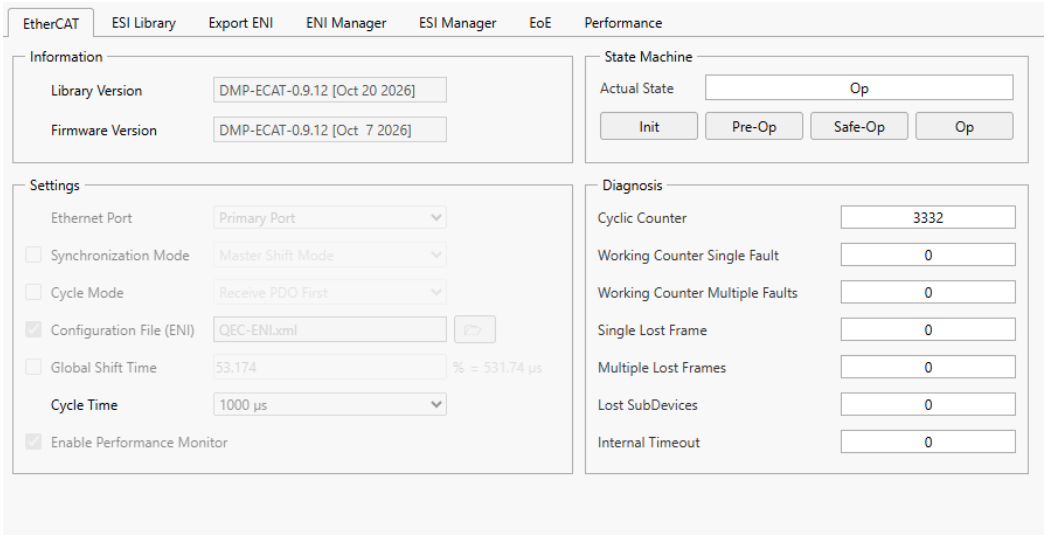


Figure 11-20-1 The EtherCAT tab of QEC-M Device: settings, state machine and diagnostic counters

ESI Manager and ESI Library tabs: once the ESI is loaded, Scan can obtain the object dictionary, SubDevice icons, Startup commands and Slots module definitions (loading is recommended). The controller does **not** load the ESI automatically at power-up.

1. **ESI Manager** tab: under Upload to Remote, click **Browse** to select a local ESI file (.xml), and click **Upload** to upload it to the controller. A progress window appears during the upload.
2. **ESI Library** tab: right-click an empty area of ESI Tree > **Load ESI** (in the Idle state). A Loading ESI window appears while loading.
3. The tree structure is Vendor -> Group -> Device (for modules, Vendor -> Modules -> Module); click a Device to see its Product Code, number of PDOs, Slots, Mailbox, DS402 Channels, etc.

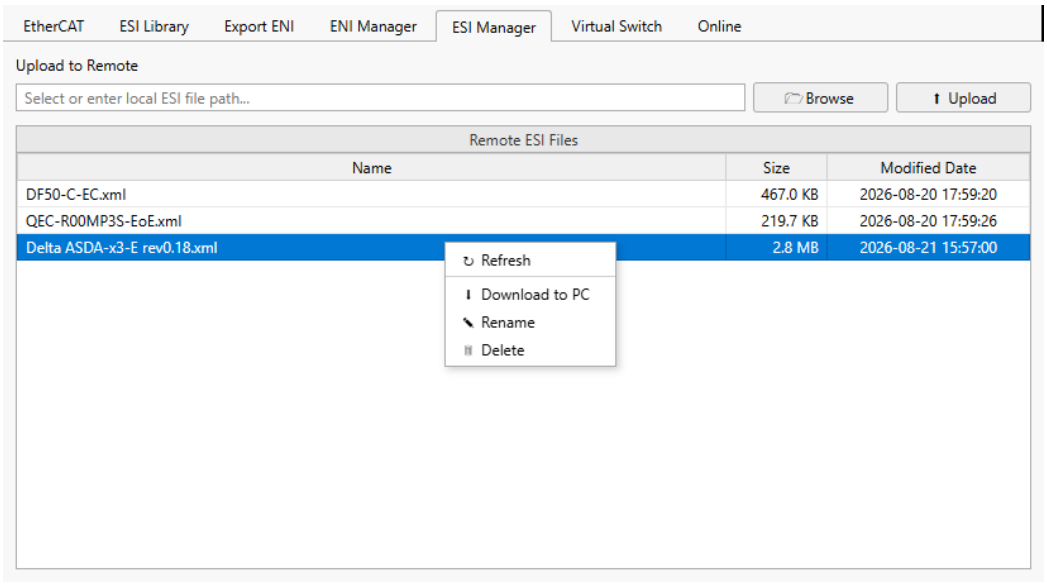


Figure 11-21 ESI Manager: the ESI files on the controller

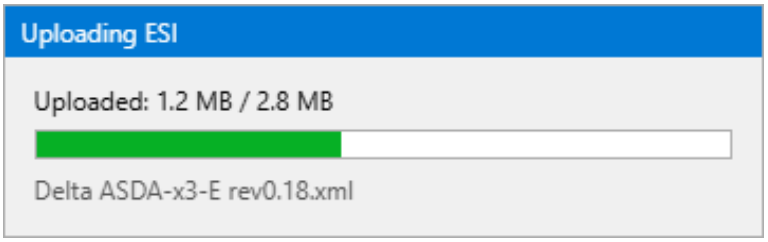


Figure 11-22 The upload progress window

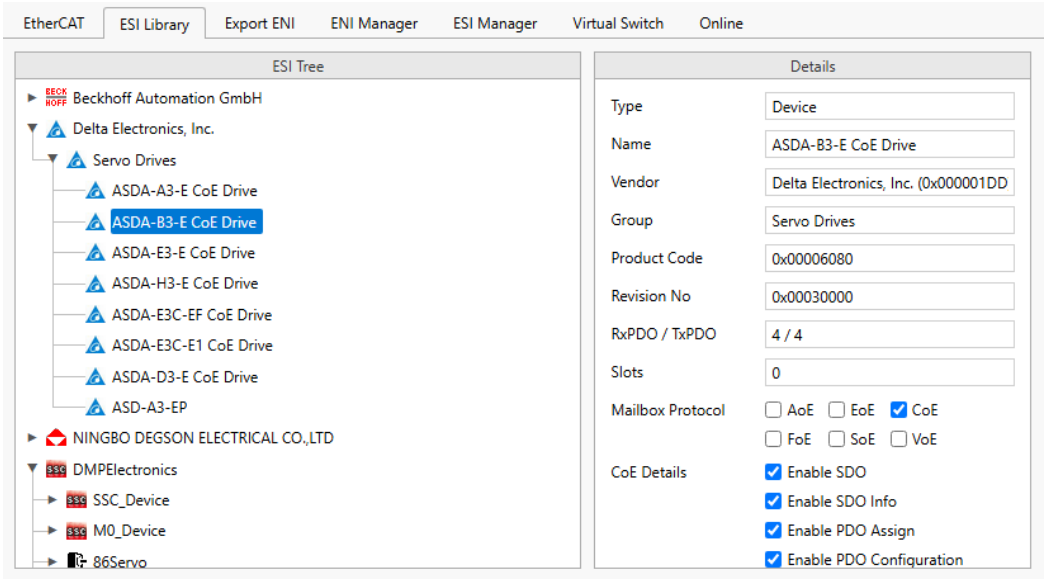


Figure 11-23 ESI Library: the loaded ESI

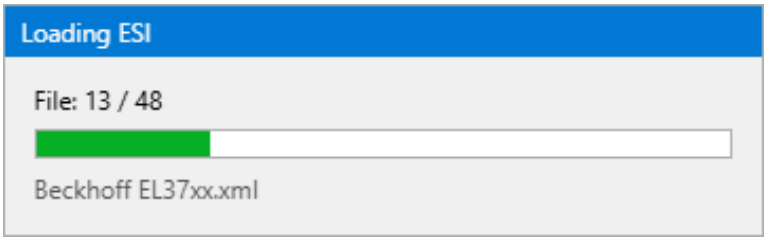


Figure 11-24 Loading ESI: loading progress

*__Tip:__ After uploading, renaming or deleting an ESI in ESI Manager, go to ESI Library and do **Unload ESI** and then **Load ESI** for the change to take effect.

Export ENI tab: makes the ENI from the current configuration and uploads it to the controller (11.2.4).

Button	Description
Open (🗁)	Selects an ENI file on the controller and shows it
Export (⚙)	Generates the complete ENI XML from the current device tree data (Slots, Process Data, Startup, DC, etc.)
Save As... (⭳)	Saves the XML shown as `QEC-ENI.xml` on the computer
Upload (⭱)	Enter a file name and upload it to the controller (`C:/eni/`)
Copy (⧉)	Copies the XML shown

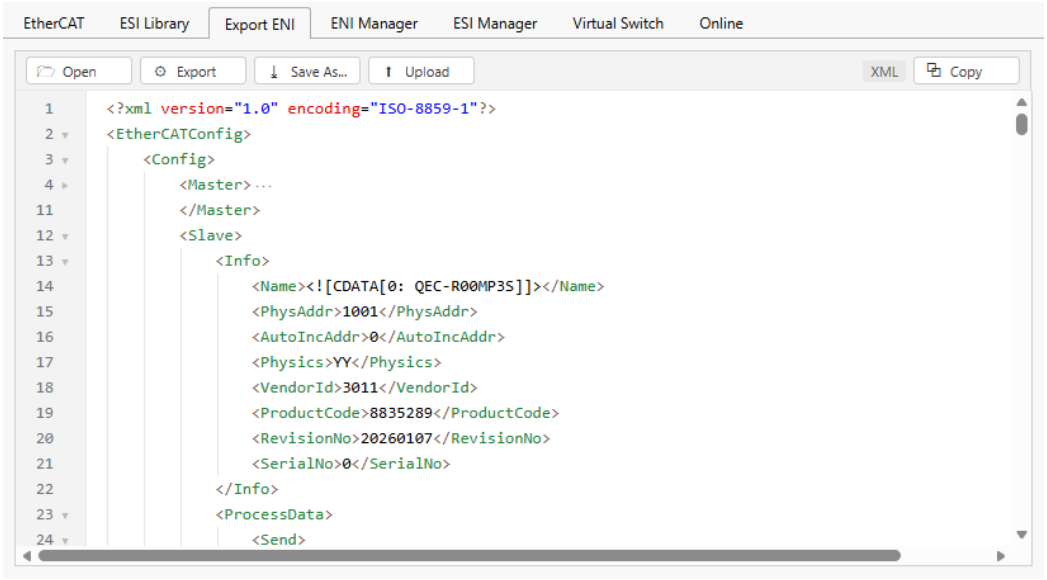


Figure 11-25 Export ENI tab: the generated ENI XML

ENI Manager tab: manages the ENI files on the controller: the table can be sorted by name, size and date; right-click for Refresh, Download to PC, Rename, Delete.

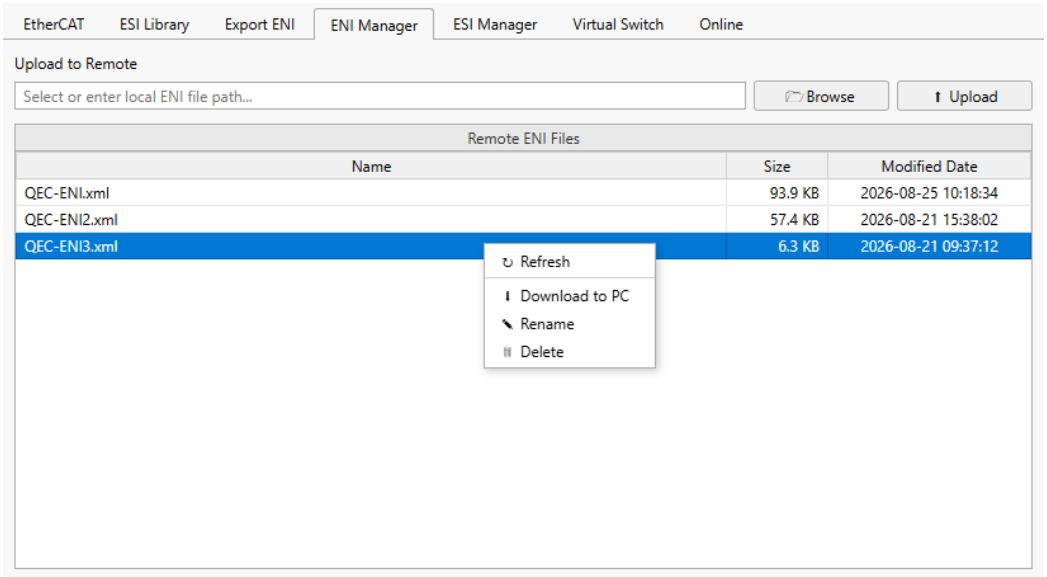


Figure 11-26 ENI Manager: the ENI files on the controller

EoE tab: appears when a SubDevice supports EoE (Ethernet over EtherCAT): tick it to add it to the virtual switch, set the MAC, IP, subnet mask, gateway and DNS, and click Set to write them (Config / Run state).

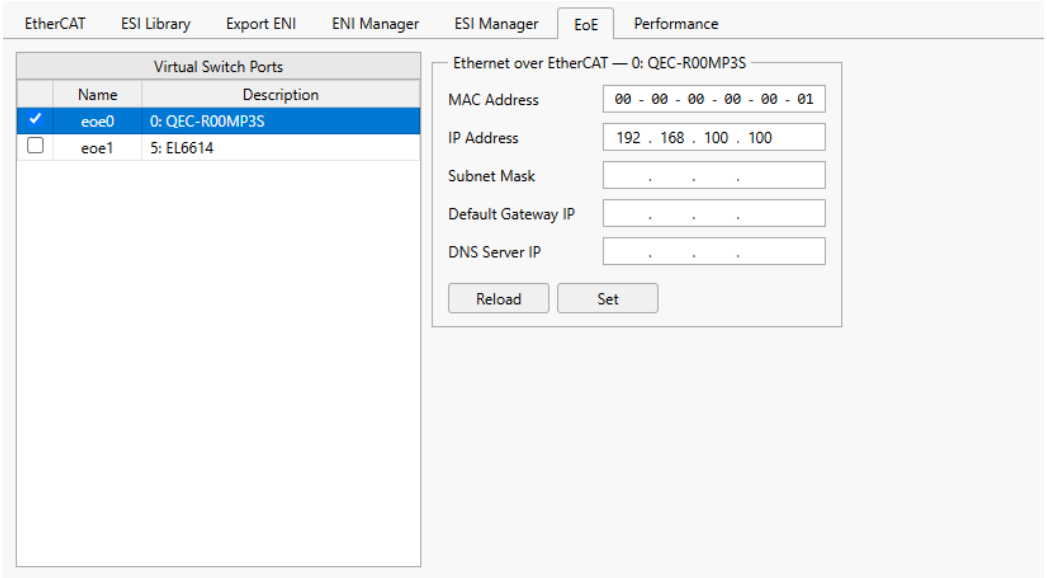


Figure 11-27 EoE tab: the virtual switch and the network settings of the SubDevices

Performance tab: appears when Enable Performance Monitor is ticked on the EtherCAT tab and the MainDevice is in Run, and is updated every 500 ms: the minimum / average / maximum time of each stage of every cycle (Receive PDO, User Application, Send PDO, Process Acyclic), the actual cycle time and the Callback Jitter, plus a cycle timeline and a jitter chart of about the last 60 seconds.

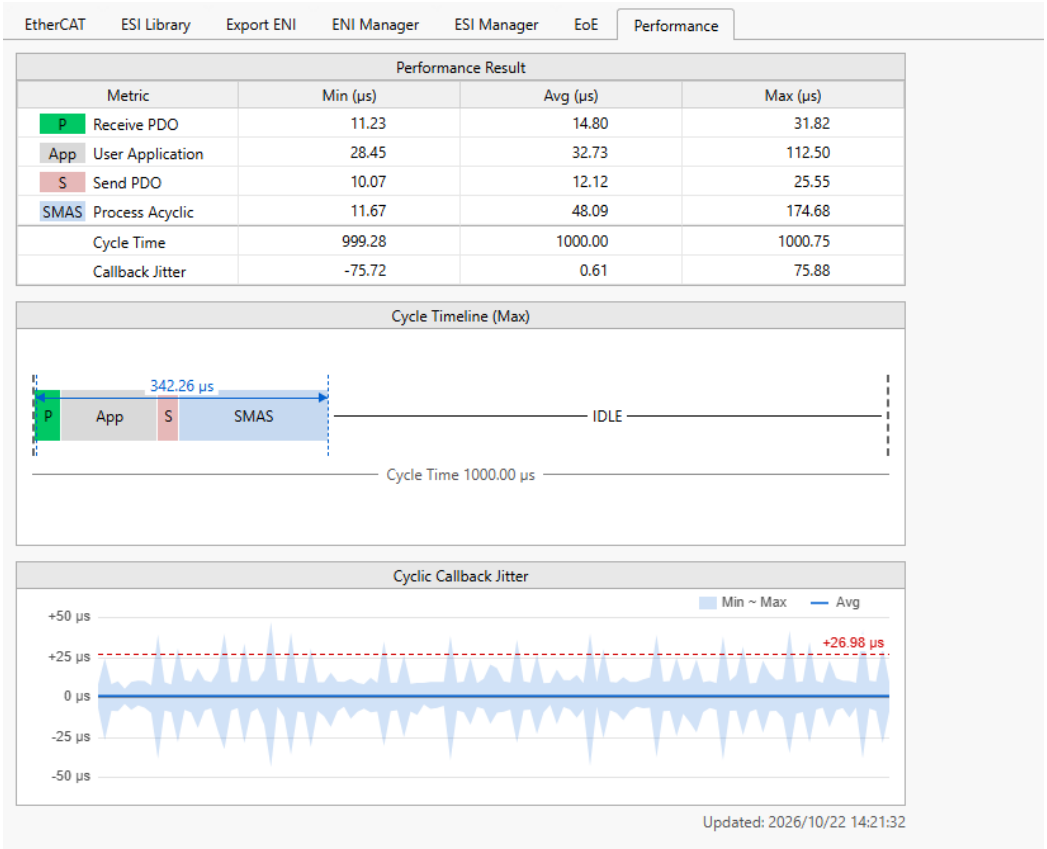


Figure 11-28 Performance tab: the time and jitter of each stage of the cycle

11.3.6 SubDevice tabs

When a SubDevice is selected, the tabs are EtherCAT, DC, E²PROM, ESC Access and Online; depending on the SubDevice there are also Slots (the ESI defines Slots), Process Data / Startup / CoE (it supports CoE) and CiA 402 (it supports CiA 402 and the MainDevice is in Config / Run).

Tab	Purpose
EtherCAT	SubDevice information (name, Vendor ID, Product Code, Revision, Serial, DS402 Channels, the matching ESI file), mailbox protocols, the state machine (Init / Pre-Op / Safe-Op / Op, Bootstrap), FoE file read / write
DC	Distributed clocks: Enable SYNC0, the SYNC0 cycle factor and Shift Time, SYNC1; written at Start (when an ENI is used, the ENI settings apply)
Slots	Modular SubDevices: right-click a Slot and choose Insert to select a module; the Startup commands are generated automatically
Process Data	PDO mapping: tick the RxPDOs (0x1C12) / TxPDOs (0x1C13) to exchange; right-click for Insert / Delete / Move / Edit; the right side shows the contents of the selected PDO
Startup	SDO commands written to the SubDevice at Pre-Op → Safe-Op (Index, CA, Data, Comment); written at Start when no ENI is used, and also output to the exported ENI
CoE	Object dictionary: Refresh, Upload, Download, Download (CA); the source can be switched between Load from ESI / Load from SDO Info
E²PROM	Station Alias; generate the EEPROM from the ESI, read it, write it; hex editor
ESC Access	ESC registers (advanced): read, write, bit-field decoding
Online	Live PDO values, updated every 200 ms in Run; right-click to change an RxPDO (output)
CiA 402	Drive test: select an axis, state machine Enable / Disable, servo settings, Mode of Operation, actual values, an operation panel for each mode

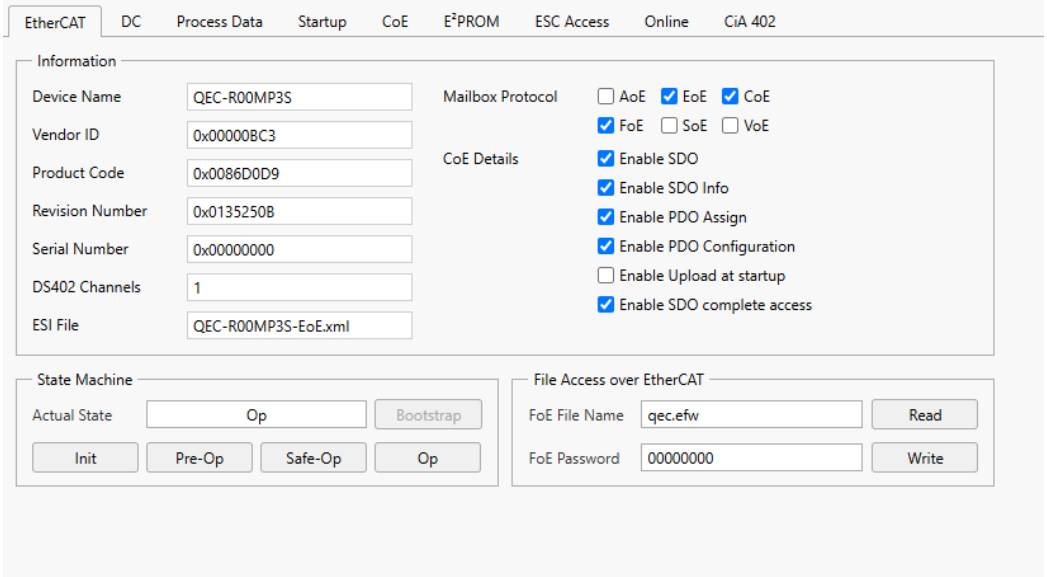


Figure 11-29 The EtherCAT tab of a SubDevice: information, state machine and FoE

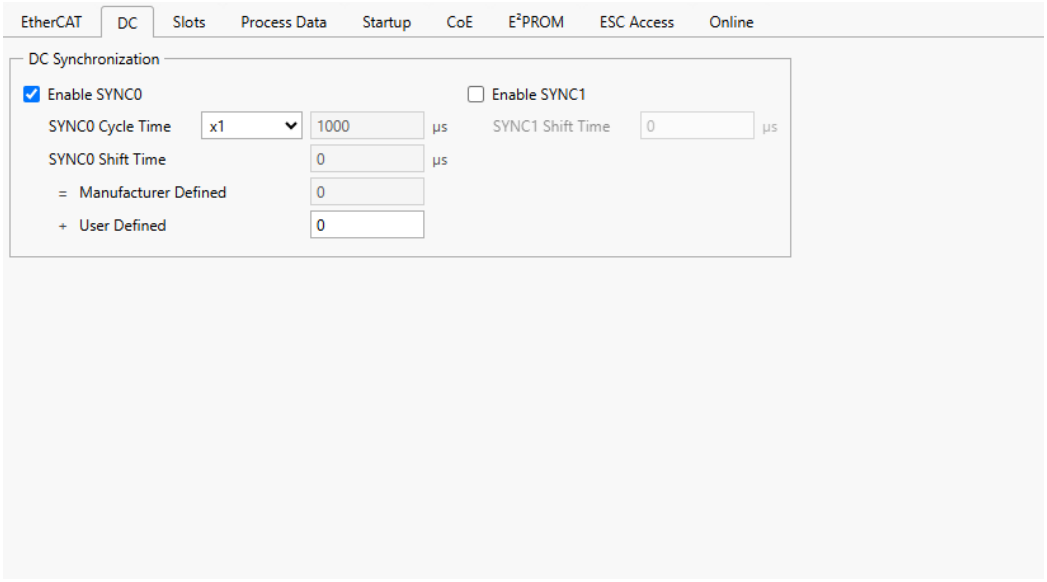


Figure 11-30 DC tab: SYNC0 / SYNC1 settings

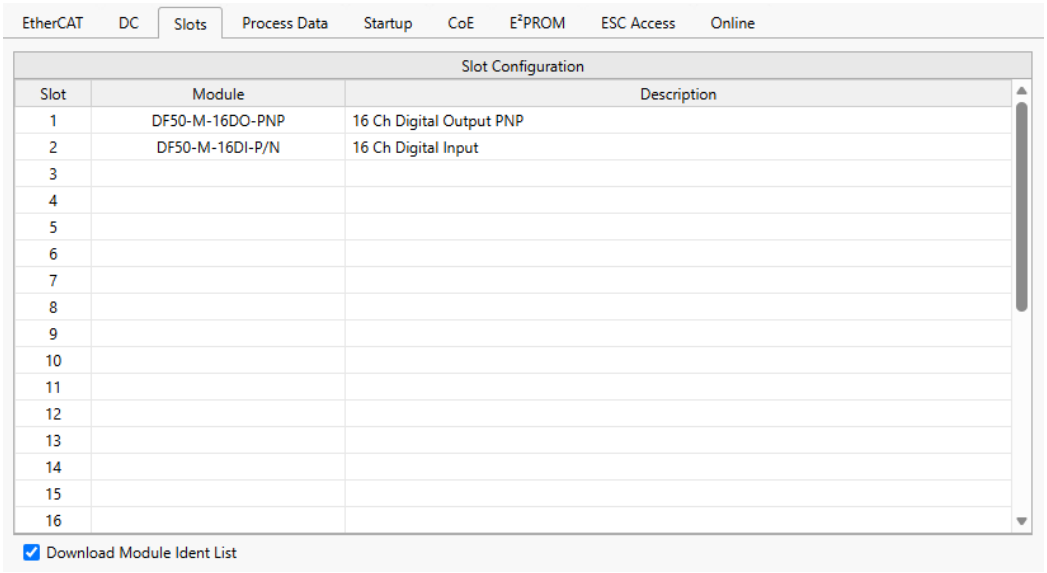


Figure 11-31 Slots tab: the Slots and modules of a modular SubDevice

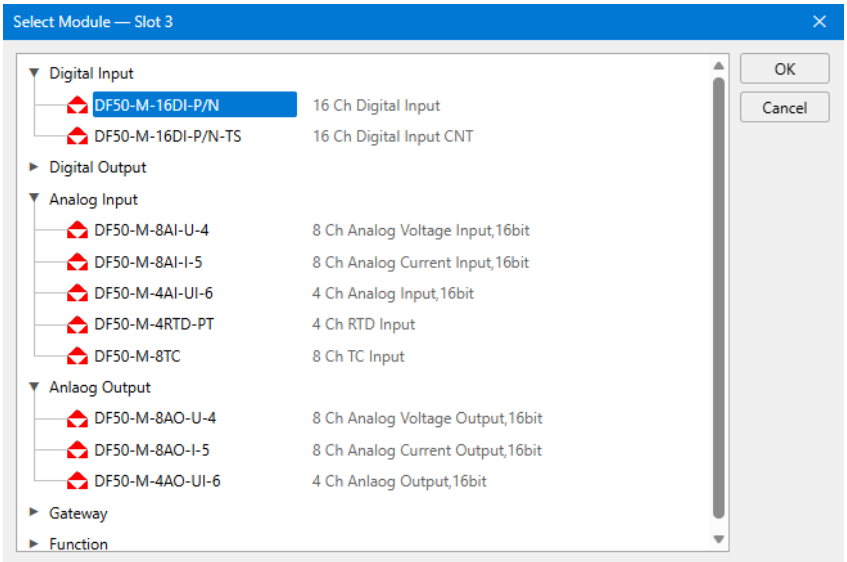


Figure 11-32 Select Module: choosing the module to insert in a Slot

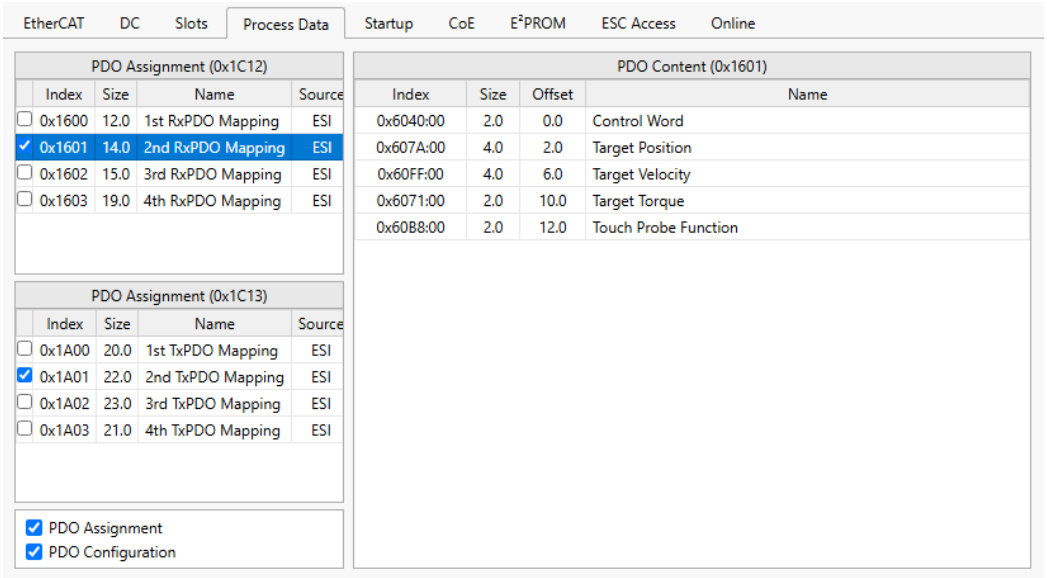


Figure 11-33 Process Data tab: the PDO assignment on the left, the mapping of the selected PDO on the right

The `Source` column of the Process Data tab tells where a PDO comes from: `ESI` (the loaded ESI or a module inserted in Slots), `SII` (the SubDevice's EEPROM), `SDO` (read by SDO at Configure), `User` (added by you). Fixed or mandatory entries are protected and cannot be deleted.

*__Important:__ When you use the CSP mode of Motion86 (Chapter 12), the PDOs of a CiA 402 drive must include the outputs 6040h (Controlword) and 607Ah (Target position) and the inputs 6041h (Statusword) and 6064h (Position actual value). For axis n of a multi-axis drive, add 800h×(n−1) to the address.

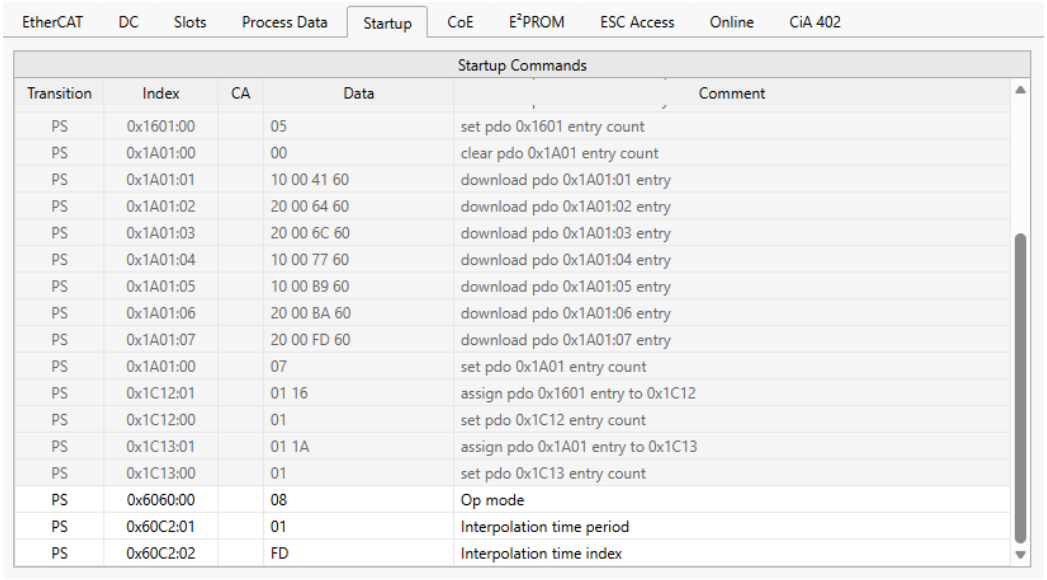


Figure 11-34 Startup tab: the SDO commands written at startup

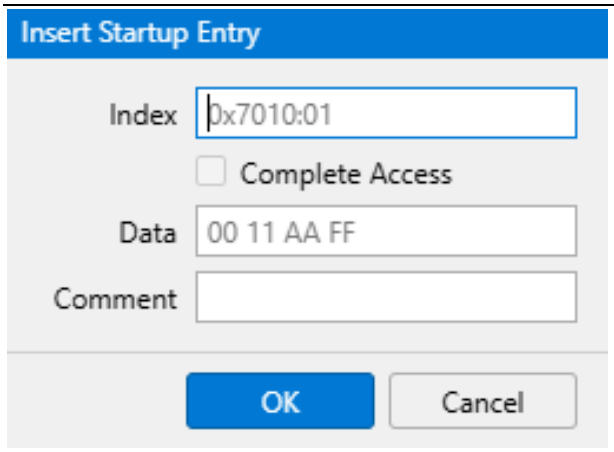


Figure 11-35 Insert/Edit Startup Entry: adding a Startup command

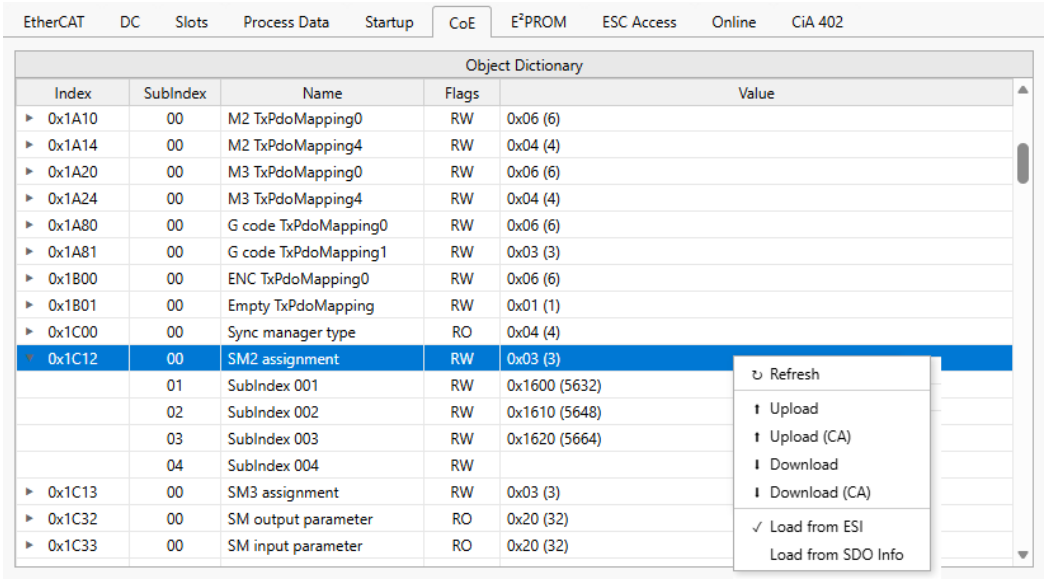


Figure 11-36 CoE tab: the object dictionary

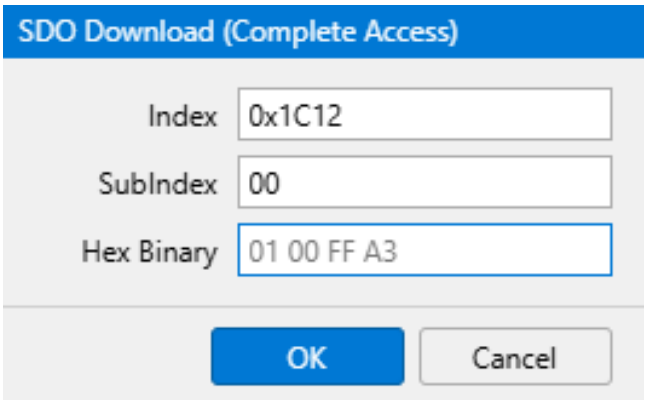


Figure 11-37 SDO Download (Complete Access): writing a whole object in hexadecimal

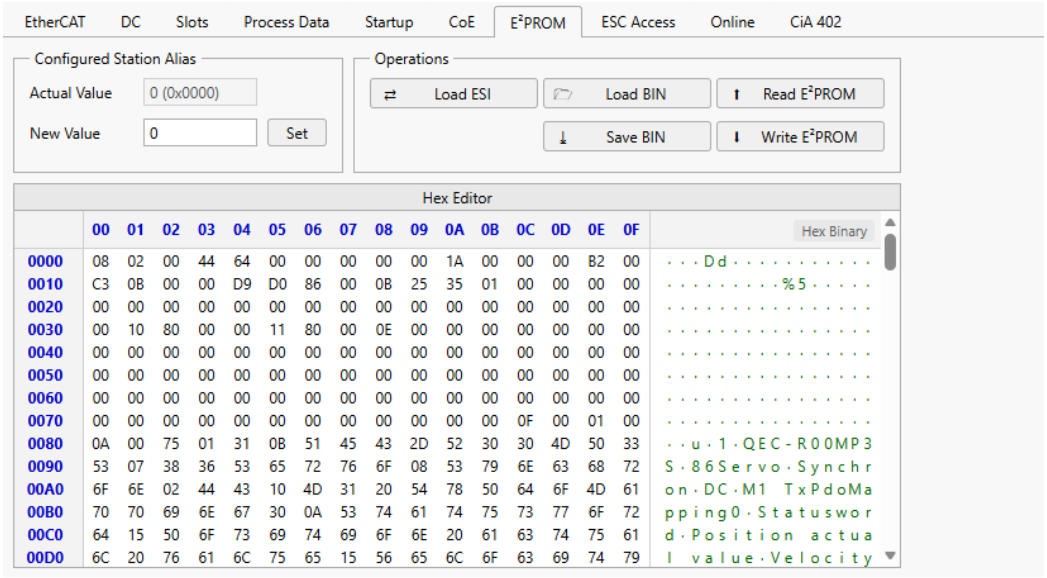


Figure 11-38 E²PROM tab: Station Alias, EEPROM operations and the hex editor

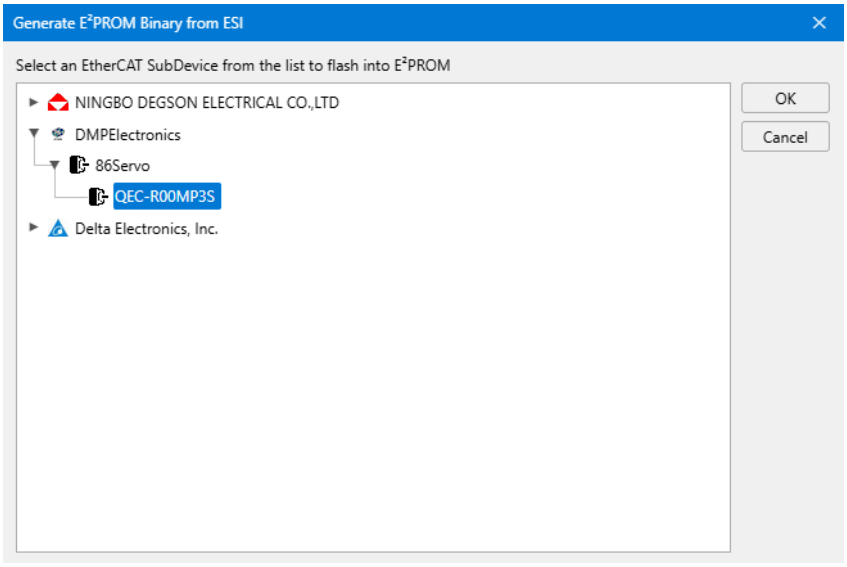


Figure 11-39 Generate E²PROM Binary from ESI: making the EEPROM contents from the ESI

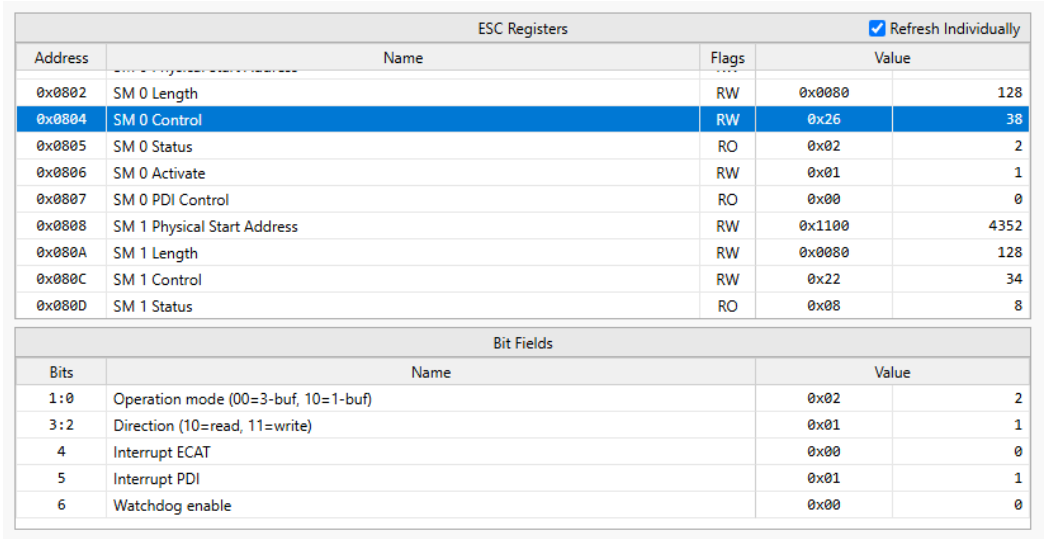


Figure 11-40 ESC Access tab: ESC registers

CiA 402 tab: tests a drive without writing a program. Recommended order: select the axis → set the Mode of Operation → click **Enable** to power it on → enter the parameters in the panel of that mode and click **Run** → click **Stop** on the panel, or simply **Disable**.

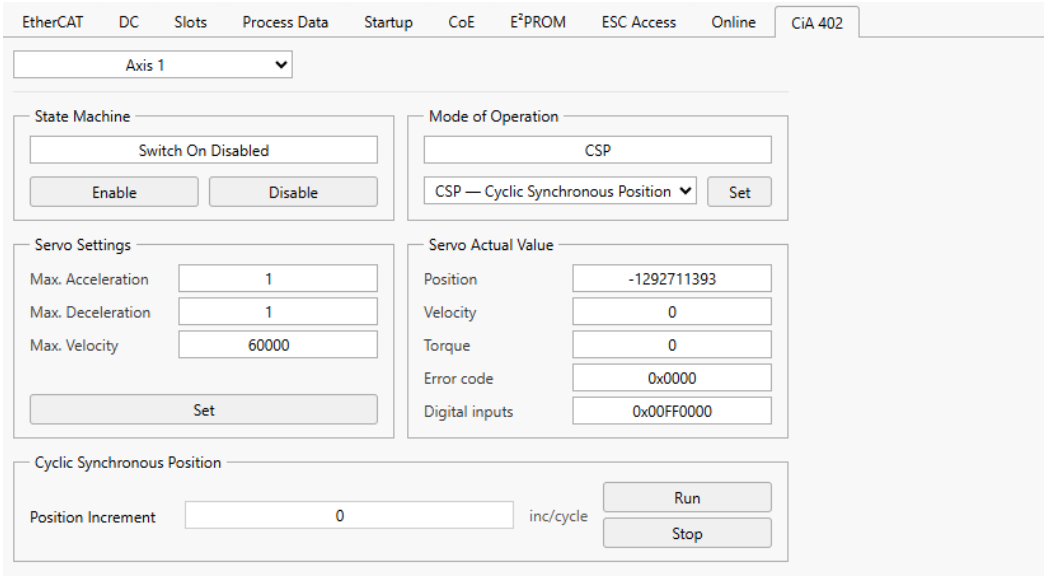


Figure 11-41 CiA 402 tab: state machine, servo settings, mode and actual values

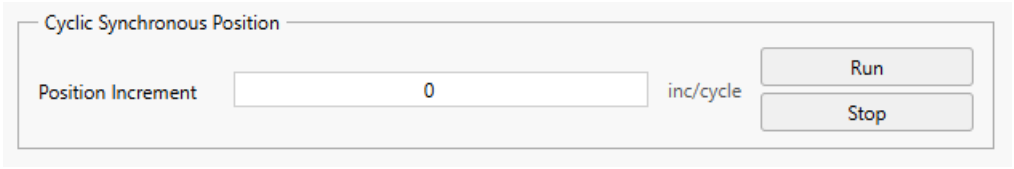


Figure 11-42 The Cyclic Synchronous Position (CSP) panel of the CiA 402 tab

Mode	Name
PP	Profile Position
PV	Profile Velocity
TQ	Profile Torque
HM	Homing
CSP	Cyclic Synchronous Position (the mode Motion86 uses)
CSV	Cyclic Synchronous Velocity
CST	Cyclic Synchronous Torque

*__Safety:__ The CiA 402 tab makes the motor turn directly. Before testing, make sure the mechanism can move freely and the emergency stop works, and start at a low speed.

11.3.7 Log panel

The log panel at the bottom of the window lists Timestamp, Device and Message: the success or failure of each operation, and the system events read back every 500 ms while the MainDevice is not Idle. Click rows (Shift to select several, Ctrl+A to select all) and right-click for Copy or Clear.

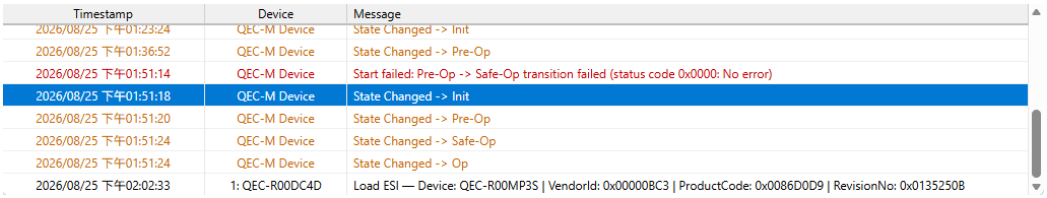


Figure 11-43 Log panel

11.3.8 Summary of the setup steps

1. (Recommended) Upload the ESI in the ESI Manager, then right-click the ESI Library and choose Load ESI.
2. On the EtherCAT tab of the QEC-M Device, check the Ethernet Port, Synchronization Mode and Cycle Mode.
3. Click **Scan** and wait until the device tree lists all the SubDevices.
4. (Optional) To use an existing ENI, tick Configuration File (ENI) on the EtherCAT tab and select the file; otherwise set up the Slots / Process Data / Startup of each SubDevice directly.
5. Click **Configure** and check that the PDOs and CoE of each SubDevice are loaded correctly; adjust the Cycle Time, Global Shift Time and DC as needed.
6. Click **Start** and check the values on the Online tab; CiA 402 drives can be tested on the CiA 402 tab.
7. Export ENI → **Upload** (for example `QEC-ENI-1.xml`).
8. Then, in order: **Stop** → **Deconfigure**.

11.4 Reading the ENI and mapping variables

11.4.1 Reading the ENI

- **Automatically**: when the EtherCAT page finds a new or updated ENI on the controller, it asks "The controller has a new ENI: … Take it into the project (its SubDevices and process data)?"; click **Take it into the project**.
- **Read the ENI...**: reads it from the controller (also under "Tools > EtherCAT > Read the ENI from the controller...").
- **Import ENI...**: select an ENI file on the computer (for example a file saved with Save As... in the Configurator).
- Command line: `86ldmicro-cli eni <project> <eni.xml> [name on the controller]` (section 15.5).

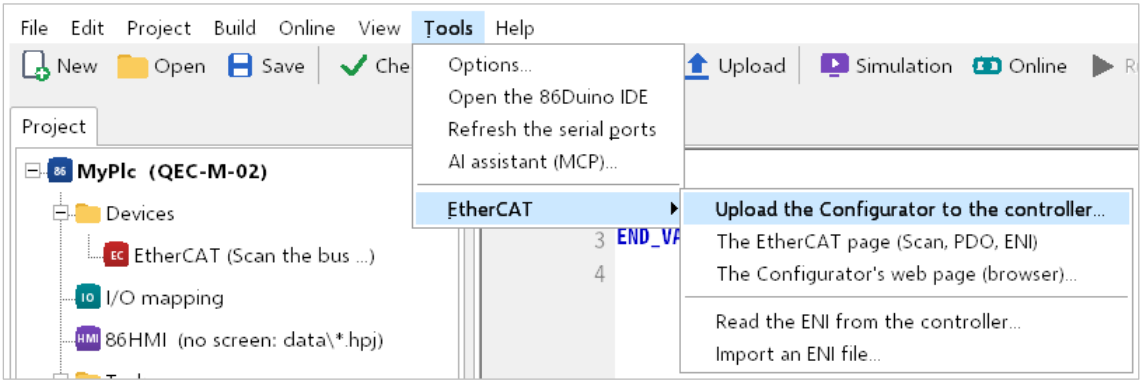


Figure 11-44 The "Tools > EtherCAT" submenu

After reading, the SubDevices and every PDO entry appear under "Devices > EtherCAT" in the project tree, and a copy of the ENI is kept in `data\86ld\ethercat\`. CiA 402 drives are shown with a motor icon; a multi-axis drive (for example QEC-R11MP3S) has one node per axis.

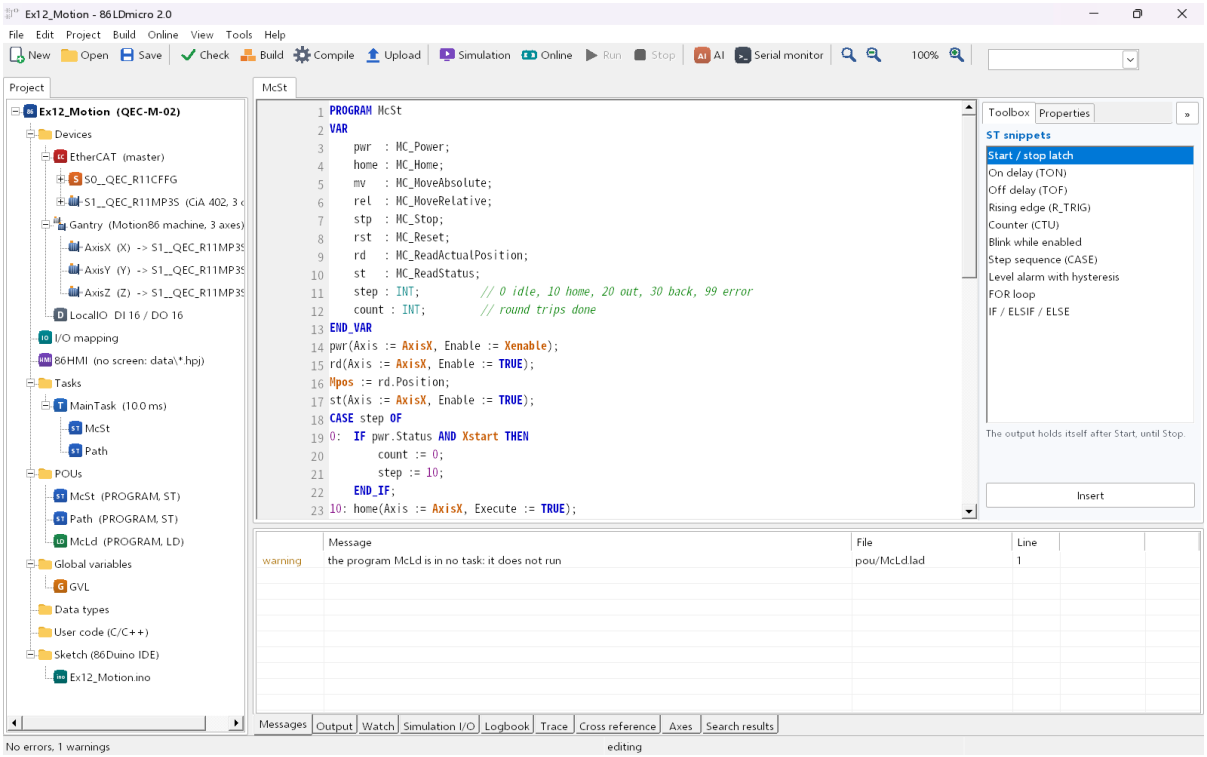


Figure 11-45 The project tree after reading the ENI: S0__QEC_R11CFFG (I/O module), S1__QEC_R11MP3S (3-axis drive)

- SubDevice names have the form `S<position>__<model>`, for example `S0__QEC_R11CFFG`.
- When a SubDevice has PDO entries with the same name (for example a multi-axis drive), the 2nd and 3rd are renamed "name #2" and "name #3" automatically; you can also specify them by Index (for example `6864:00`).
- A SubDevice whose PDOs have 6040h / 6041h is treated as a CiA 402 drive automatically, whatever the vendor; you can also set this by hand with **Profile (CiA 402)...** on the SubDevice page (section 11.5).
- When the ENI is read again, the existing mappings, the machines and the manually set profiles are kept.

11.4.2 Mapping PDOs to variables

A PDO entry must be mapped to a variable before the program can read or write it. There are three ways:

- On the EtherCAT page, click **Variable mapping**: select a row and click **Link variable...** (or double-click it); **Unlink** removes the mapping.
- In the project tree, double-click a PDO entry under a SubDevice in "Devices > EtherCAT".
- On the SubDevice page, select a row and click **Link to a variable...** (section 11.5).

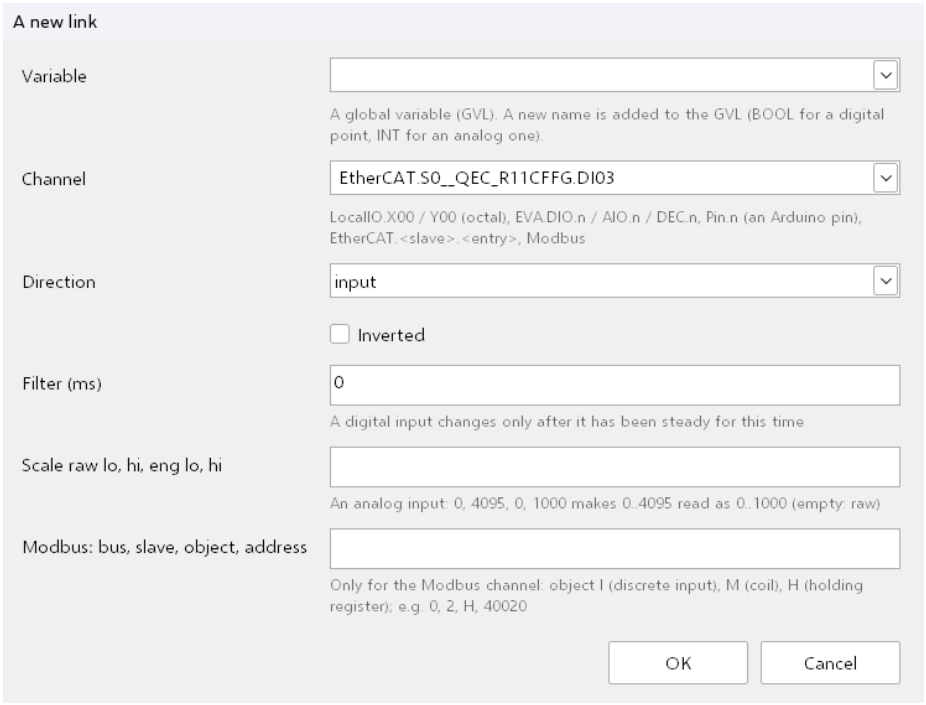
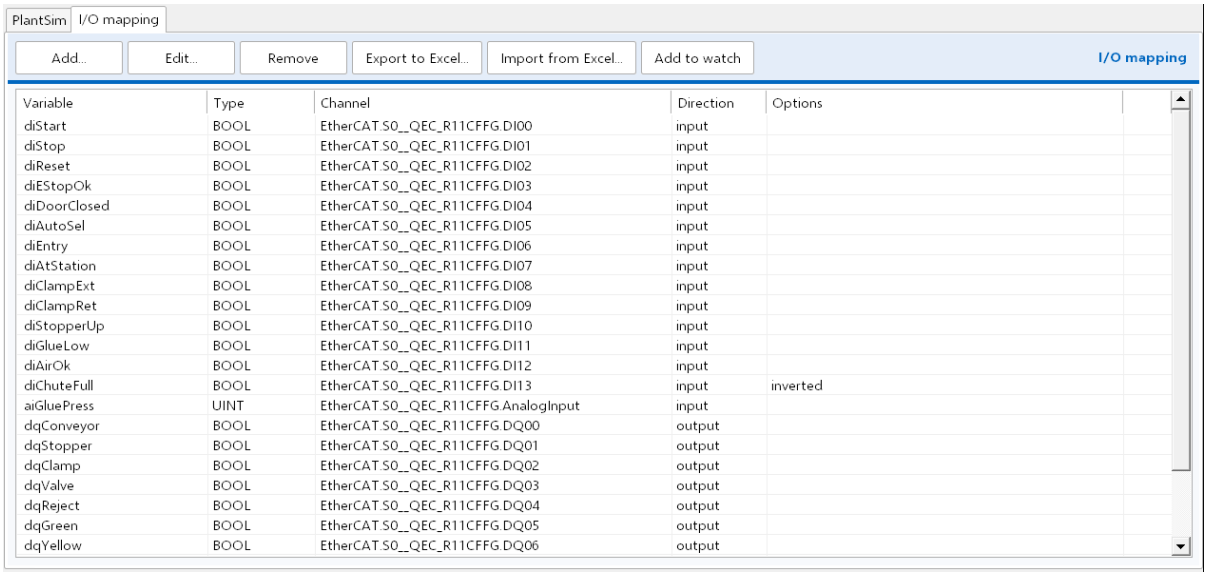


Figure 11-46 Link variable: the channel is filled in (for example EtherCAT.S0__QEC_R11CFFG.DI02); enter or select a variable

The channel name is `EtherCAT.<SubDevice>.<PDO entry>`. When you enter a new variable name, the variable is added to the GVL automatically (with the type of the PDO). Once mapped, the mappings are also listed on the "I/O mapping" page (section 4.6), and the variables can be used directly in LD, ST, SFC and C/C++ (`GVL(name)`).



Variable	Type	Channel	Direction	Options
diStart	BOOL	EtherCAT.S0_QEC_R11CFFG.DI00	input	
diStop	BOOL	EtherCAT.S0_QEC_R11CFFG.DI01	input	
diReset	BOOL	EtherCAT.S0_QEC_R11CFFG.DI02	input	
diEStopOk	BOOL	EtherCAT.S0_QEC_R11CFFG.DI03	input	
diDoorClosed	BOOL	EtherCAT.S0_QEC_R11CFFG.DI04	input	
diAutoSel	BOOL	EtherCAT.S0_QEC_R11CFFG.DI05	input	
diEntry	BOOL	EtherCAT.S0_QEC_R11CFFG.DI06	input	
diAtStation	BOOL	EtherCAT.S0_QEC_R11CFFG.DI07	input	
diClampExt	BOOL	EtherCAT.S0_QEC_R11CFFG.DI08	input	
diClampRet	BOOL	EtherCAT.S0_QEC_R11CFFG.DI09	input	
diStopperUp	BOOL	EtherCAT.S0_QEC_R11CFFG.DI10	input	
diGlueLow	BOOL	EtherCAT.S0_QEC_R11CFFG.DI11	input	
diAirOk	BOOL	EtherCAT.S0_QEC_R11CFFG.DI12	input	
diChuteFull	BOOL	EtherCAT.S0_QEC_R11CFFG.DI13	input	inverted
aiGluePress	UINT	EtherCAT.S0_QEC_R11CFFG.AnalogInput	input	
dqConveyor	BOOL	EtherCAT.S0_QEC_R11CFFG.DQ00	output	
dqStopper	BOOL	EtherCAT.S0_QEC_R11CFFG.DQ01	output	
dqClamp	BOOL	EtherCAT.S0_QEC_R11CFFG.DQ02	output	
dqValve	BOOL	EtherCAT.S0_QEC_R11CFFG.DQ03	output	
dqReject	BOOL	EtherCAT.S0_QEC_R11CFFG.DQ04	output	
dqGreen	BOOL	EtherCAT.S0_QEC_R11CFFG.DQ05	output	
dqYellow	BOOL	EtherCAT.S0_QEC_R11CFFG.DQ06	output	

Figure 11-47 I/O mapping page: the EtherCAT channels listed together with the local channels

***Tip:** If an input variable that the program reads has no mapping, "Messages" warns at build time: "… is only read here: it has no I/O mapping and nothing writes it -- it is always FALSE".

11.4.3 Settings of the EtherCAT device

Right-click "Devices > EtherCAT" in the project tree > "Edit" (or select EtherCAT on the devices page and click "Edit") to adjust how the MainDevice runs:

Setting	Description
mode	`auto`: DC synchronization when there are CiA 402 servos, otherwise free run (default); `sync`: DC synchronization (free run if it fails); `freerun`
master	`plc`: the program generated by 86LDmicro starts the MainDevice (default, `plc86_ecat.cpp`); `sketch`: your .ino does begin, attach and start itself, then sets `plc86_ecOk = true`
cycleUs	EtherCAT cycle (µs), default 1000

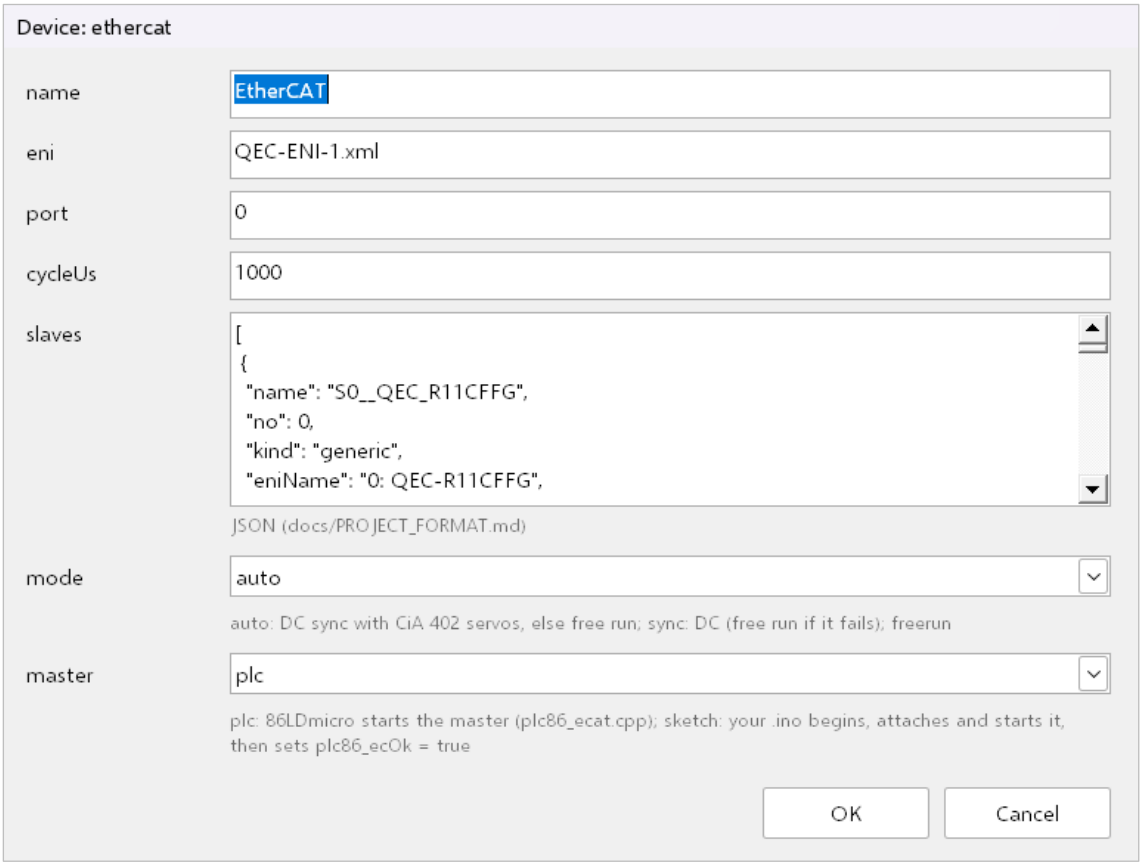


Figure 11-48 Settings of the EtherCAT device

11.4.4 Building, uploading and checking the MainDevice

1. Click **Upload** (F8). The PLC program replaces the Configurator; at power-up the MainDevice starts with `C:/eni/<file name>`, each SubDevice is attached, the drives are set to their mode, and then the cyclic exchange begins.
2. Open the "Serial monitor" (no need to go online, section 10.3): the MainDevice status is printed every 5 seconds (in normal operation only 3 times after startup; on failure it keeps printing), for example `EtherCAT: running, 2 slaves, 1000 us, sync, ENI C:/eni/QEC-ENI-1.xml`, or the reason for the failure.
3. When online, the "Logbook" also records the MainDevice status (section 10.5).

11.5 SubDevice page

Double-click a SubDevice under "Devices > EtherCAT" in the project tree to open its SubDevice page: the top shows its position on the bus, its name in the ENI, the vendor code / product code, the profile and the object name in the code; the bottom has two tabs, "**Process data (PDO)**" and "**CoE object dictionary (online)**".

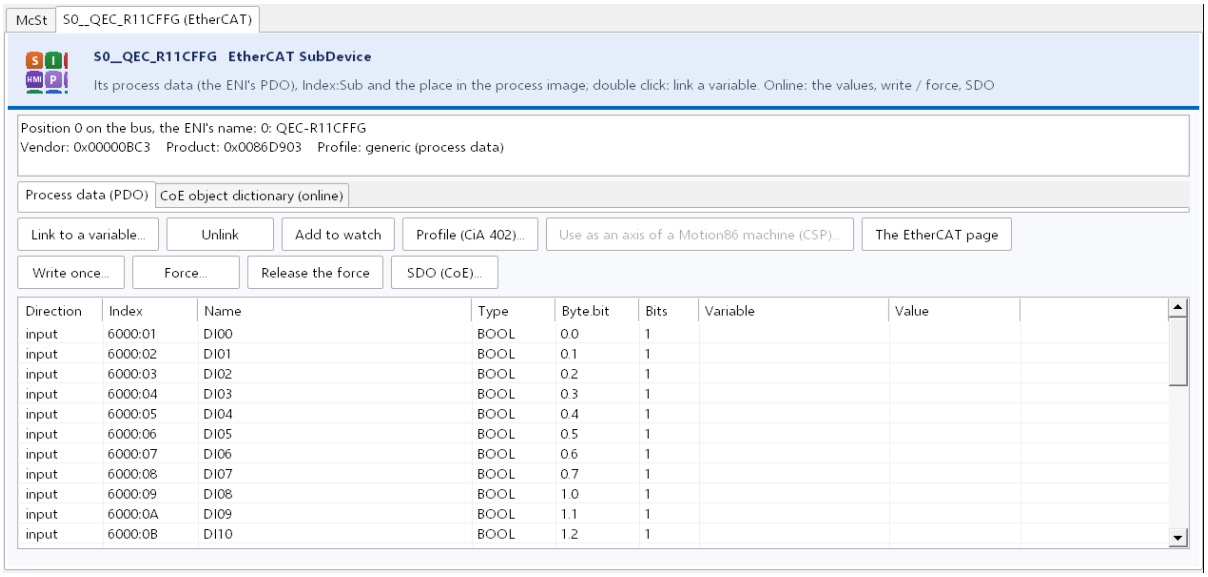


Figure 11-49 SubDevice page of QEC-R11CFFG: the direction, Index, name, type, Byte.bit, number of bits and variable of each PDO entry

Button	Description
Link to a variable... / Unlink	Links the selected PDO entry to a variable, or removes the link
Add to watch	Adds the linked variable to the Watch window
Profile (CiA 402)...	Sets the profile by hand (Auto / CiA 402 / Generic) and the number of drives (third-party drives)
Use as an axis of a Motion86 machine (CSP)...	Adds the selected drive to a Motion86 machine (section 12.2)
The EtherCAT page	Returns to the EtherCAT page
Write once... / Force... / Unforce	Online: writes or forces a PDO output directly (section 11.6)
SDO (CoE)...	Online: reads or writes a single object (section 11.6)

For a CiA 402 drive, the bottom of the SubDevice page also lists the Motion86 axis of each drive (#1, #2, #3…):

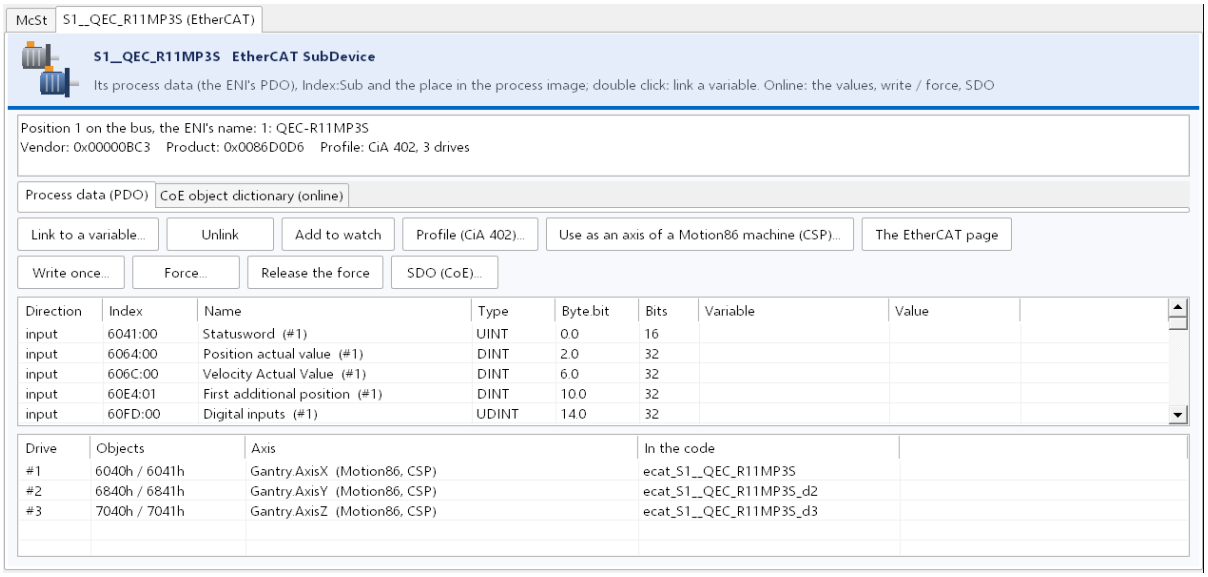


Figure 11-50 QEC-R11MP3S (3-axis drive): the Motion86 axis of each drive listed at the bottom

When the PDOs of a third-party drive do not have the standard 6040h / 6041h, click **Profile (CiA 402)...** to set it by hand:

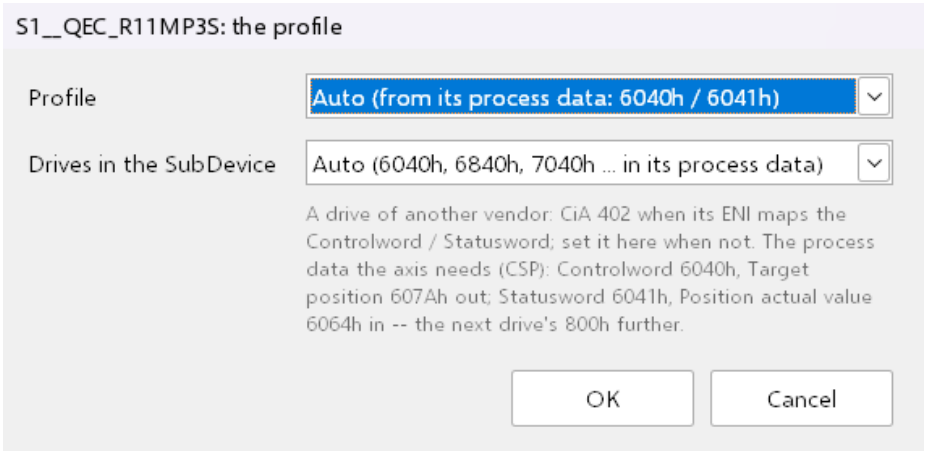


Figure 11-51 Profile settings: Auto, CiA 402 or Generic, and "Drives in the SubDevice"

The SubDevice page can be placed side by side with a program ("View > Side by side: the tab in front to the right / end it", Ctrl+\, section 3.7), so that you see the PDO values and the program at the same time:

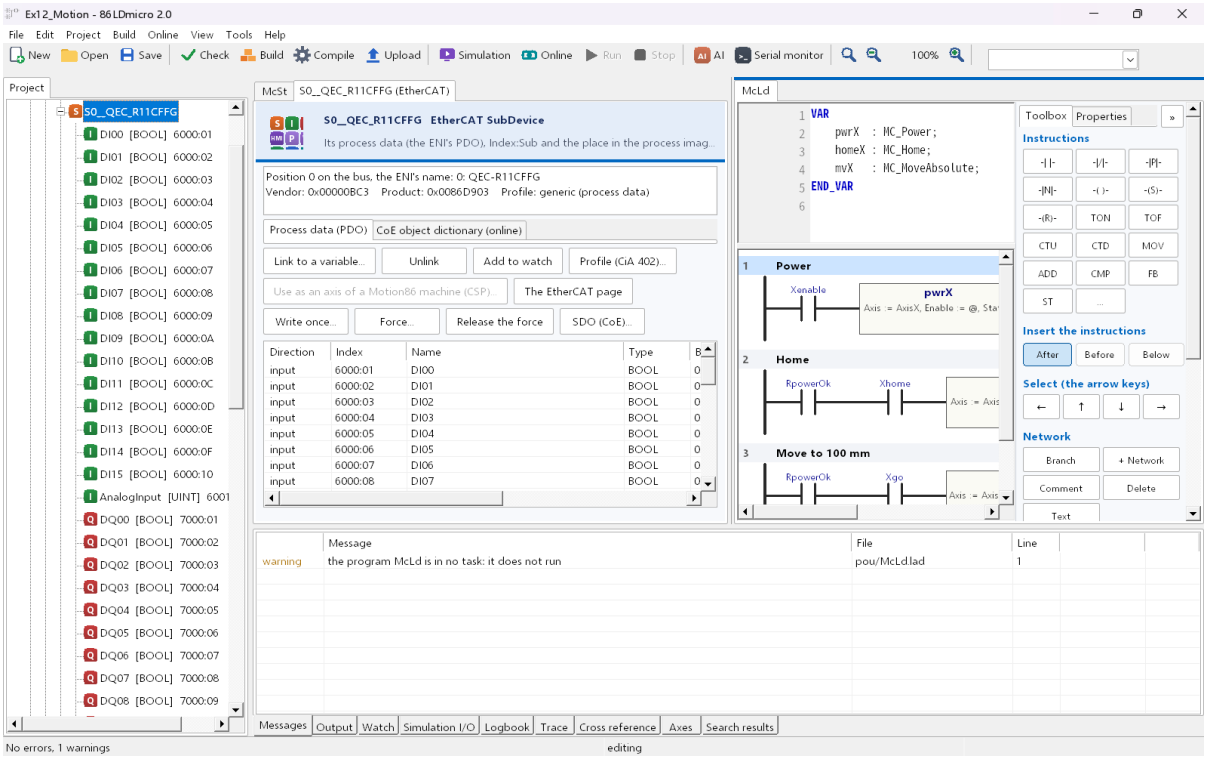


Figure 11-52 Side by side: the SubDevice page on the left, a ladder diagram on the right

11.6 PDO online values, SDO and CoE

11.6.1 PDO online values

Once online (Ctrl+F5), all PDO entries of the SubDevice page (linked to a variable or not) show their current values: the controller updates them every 100 ms and the page refreshes every 0.5 seconds. REAL shows a floating-point value; WORD / DWORD also show the hexadecimal value.

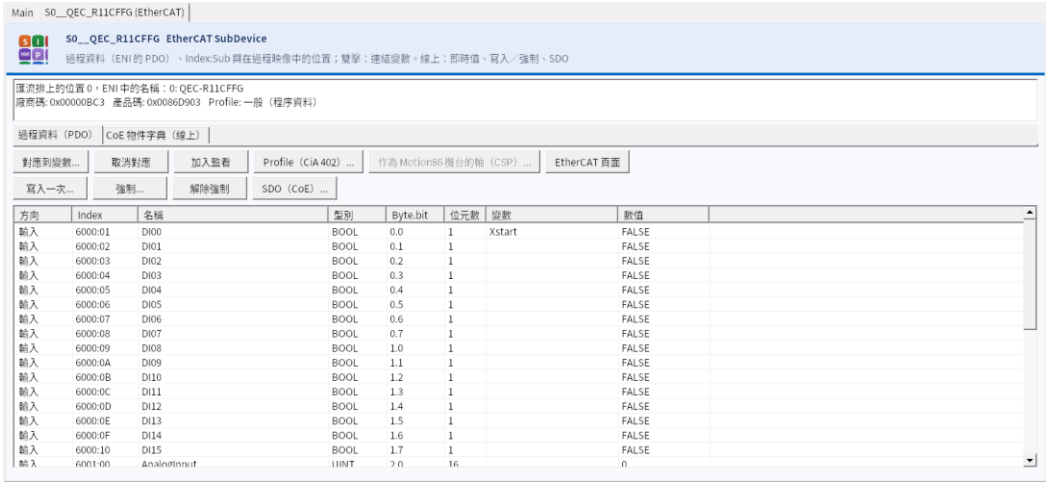


Figure 11-53 Online: each PDO entry shows its live value

11.6.2 Write once and force

Function	Description
Write once...	Writes once and then no more; the program can still change it
Force...	Writes it in every scan until "Unforce" or until the connection has been lost for 3 seconds; `[F]` is shown while it is forced
Unforce	Releases the force of the selected entries

- For an entry linked to a variable, it is **that variable** that is written or forced (in simulation too); an **output** not linked to a variable is written directly to the process data (online only).
- **Cannot be written by hand**: inputs (written by the SubDevice; to test the program, link a variable and force that variable), and the motion objects of CiA 402 drives (6040h, 607Ah, 60FFh, 6060h, etc., controlled by the axis).

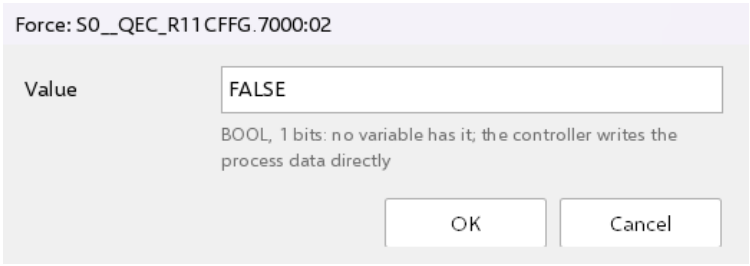


Figure 11-54 Forcing an output not linked to a variable (DQ01): the controller writes the process data directly

***Safety:** Forcing an output takes the equipment out of the program's control; make sure the site is safe before you do it.

11.6.3 SDO (CoE): reading and writing a single object

Select a PDO entry and click **SDO (CoE)...** (its Index is filled in), then choose the action (read or write), the Index:Sub (hexadecimal, for example `1018:01`), the size (1 / 2 / 4 / 8 bytes) and the value (for writing: decimal, or hexadecimal starting with 0x). On failure, the abort code and its description are shown.

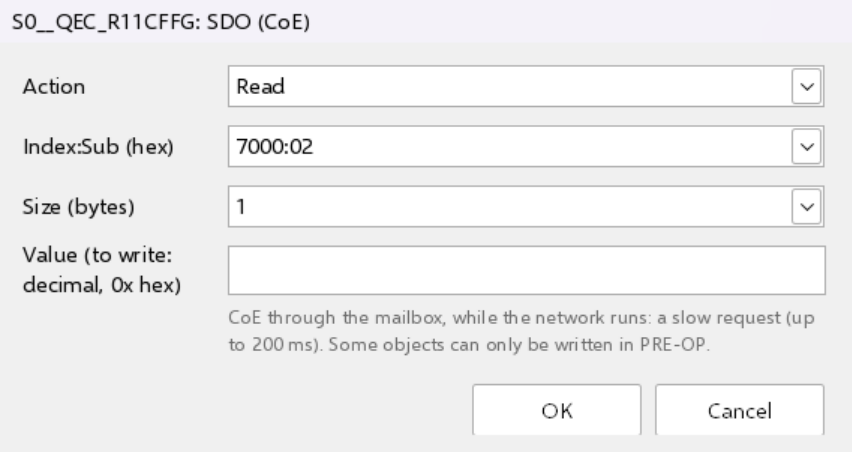


Figure 11-55 SDO (CoE): reading or writing a single object

11.6.4 CoE object dictionary (online)

Like CoE - Online in TwinCAT, it lets you browse and change all the parameters of a drive:

1. Once online, switch to the **CoE object dictionary (online)** tab of the SubDevice page and click **Read the object dictionary**: the complete list (name, type, access rights) is obtained from the SubDevice with SDO Info and stored in the project, and is shown directly the next time you open the page.
2. Click **Read all values**; or tick **Auto refresh** and choose an interval (0.5 / 1 / 2 / 5 seconds): the rows visible on the screen keep updating, and the background of a value that changes briefly turns yellow.
3. Use **Filter** at the top to filter by Index or name, for example `6098`, `Homing`.
4. Double-click an RW entry (or click **Write...**) to change the parameter; it is read back right after writing.

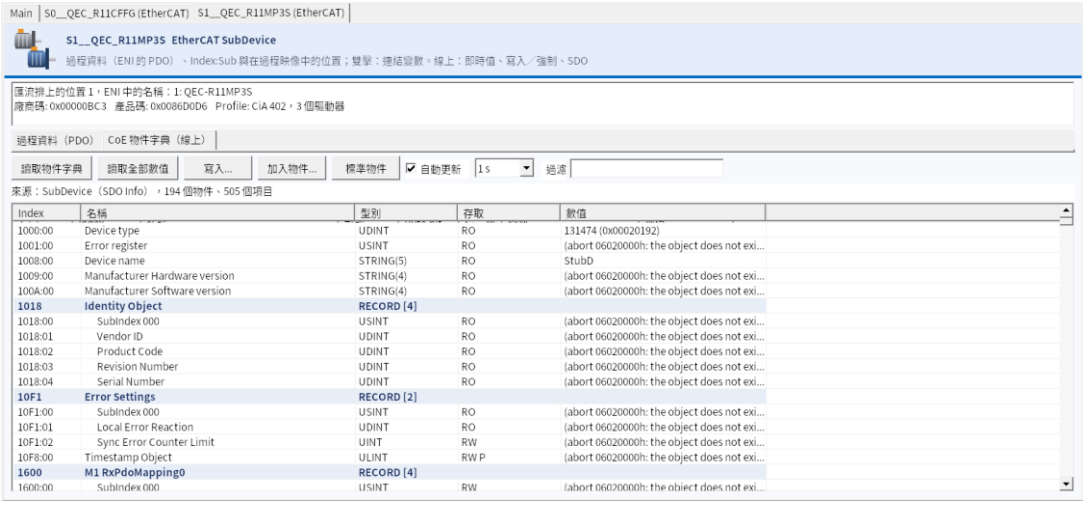


Figure 11-56 CoE object dictionary (online): the objects, types, access and values of QEC-R11MP3S

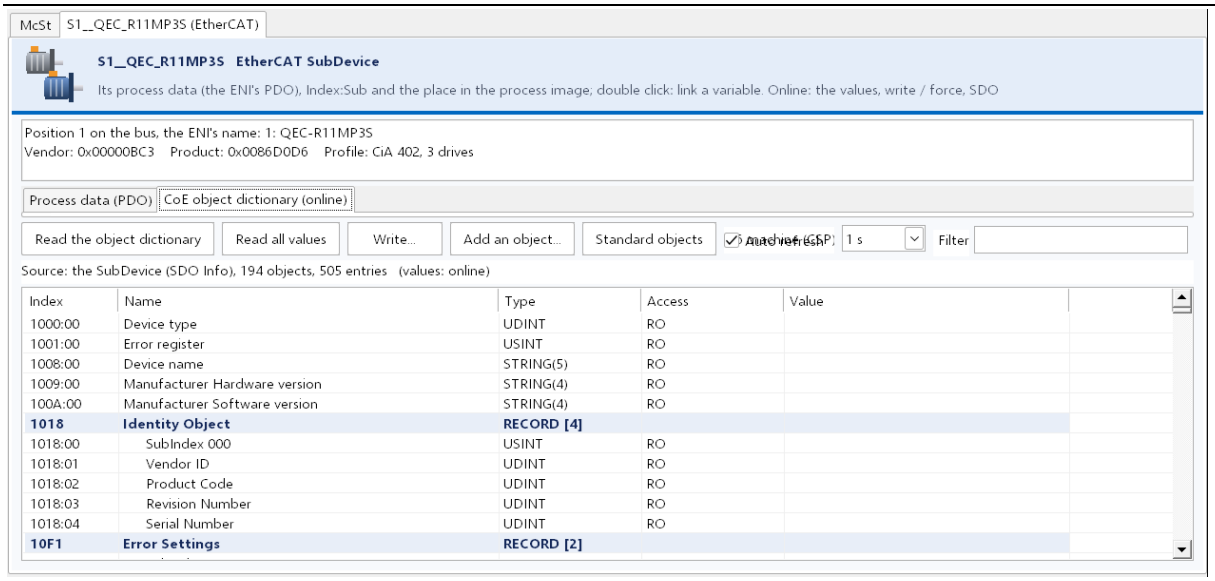


Figure 11-57 When not online: the object list stored in the project is shown (the values need a connection)

- Access column: RO read-only, RW read/write; "(PRE-OP)" means it can be written only in the PRE-OP state; "P" means it can be mapped to a PDO.
- When the SubDevice does not support SDO Info, the **standard objects** (1000h, 1018h, common CiA 402 objects) and the PDO entries are listed instead; **Add an object...** adds other Indexes by hand.
- Reads and writes are done without blocking, between the controller's scans, so they **do not stall the PLC scan**.
- Common abort codes: 06020000h object does not exist, 06010002h read-only, 06090030h value out of range, 08000022h not allowed in the current state (writable only in PRE-OP).
- Command line: `86ldmicro-cli online <project> <port> sdo …, … coe <SubDevice>` (section 15.5).

11.7 Troubleshooting

When EtherCAT does not work, first open the "Serial monitor" (no need to go online) to see the MainDevice status, then use the table below:

Symptom	Cause and remedy
The EtherCAT page shows no answer from the Configurator	The Configurator has not been uploaded, or the PLC program has replaced it: click **Upload Configurator**; over USB, check that the port selected in the toolbar is the controller's; over the network, check that the computer is connected to the controller's LAN port, and check the IP
`EtherCAT: begin failed`	The EtherCAT cable is not connected to the MainDevice's EtherCAT port, or the SubDevices are not powered; the .ino still calls `EVA.begin()` of 1.x (section 11.1)
`the ENI … cannot be used`	The controller does not have the project's ENI (`C:/eni/…`), or it does not match the actual network, so the MainDevice uses the scanned network instead: Scan again, export and upload the ENI, and read it back into the project
`no slave N`	The number or order of the SubDevices does not match the ENI, or the cable is connected to the OUT port of a SubDevice instead of IN: check the wiring, Scan again, export the ENI and read it back
An input is always FALSE	The variable is not linked (the build warns in "Messages"); or the PDO is not in the assignment of the ENI (tick it on the Process Data tab and export again)
EtherCAT does not work at all	Remove the 1.x `EVA.begin()` and `myeva.*`; check that master of the EtherCAT device is `plc` (11.4.3)
A CSP drive does not move	The PDOs must have 6040h / 607Ah / 6041h / 6064h (11.3.6); check the Statusword and the error code 603Fh in the CoE object dictionary; for the machine and axis settings, see Chapter 12
The PLC program is gone after uploading the Configurator	This is normal: the Configurator temporarily replaces the PLC program; when the setup is done, click **Upload** (F8)
The Configurator disconnects during an upload or while online	In USB mode, uploading and online monitoring take priority and the Configurator pauses automatically; it resumes when you return to the EtherCAT page afterwards

Chapter 12 Motion86 motion control

86LDmicro manages EtherCAT servo drives with TwinCAT-style "machines and axes". This chapter describes how to create and set up Motion86 machines and axes, how to commission them without writing a program, how to write motion programs with the PLCopen MC function blocks and the M86 machine function blocks, and the safety mechanisms. Before you start, complete Chapter 11 so that the CiA 402 drives appear under "Devices > EtherCAT" in the project tree.

12.1 Concepts

A **Motion86 machine** is an axis group of the Motion86 library, with at most **9 axes** (X, Y, Z, U, V, W, A, B, C). Each **axis** is an EtherCAT CiA 402 drive controlled in **CSP (cyclic synchronous position)** mode: in every EtherCAT cycle, Motion86 calculates the target position of each axis and sends it to the drive. The machine **Gantry** of the example `tools\example\ex30_dispense` uses the three axes of a QEC-R11MP3S (AxisX, AxisY, AxisZ).

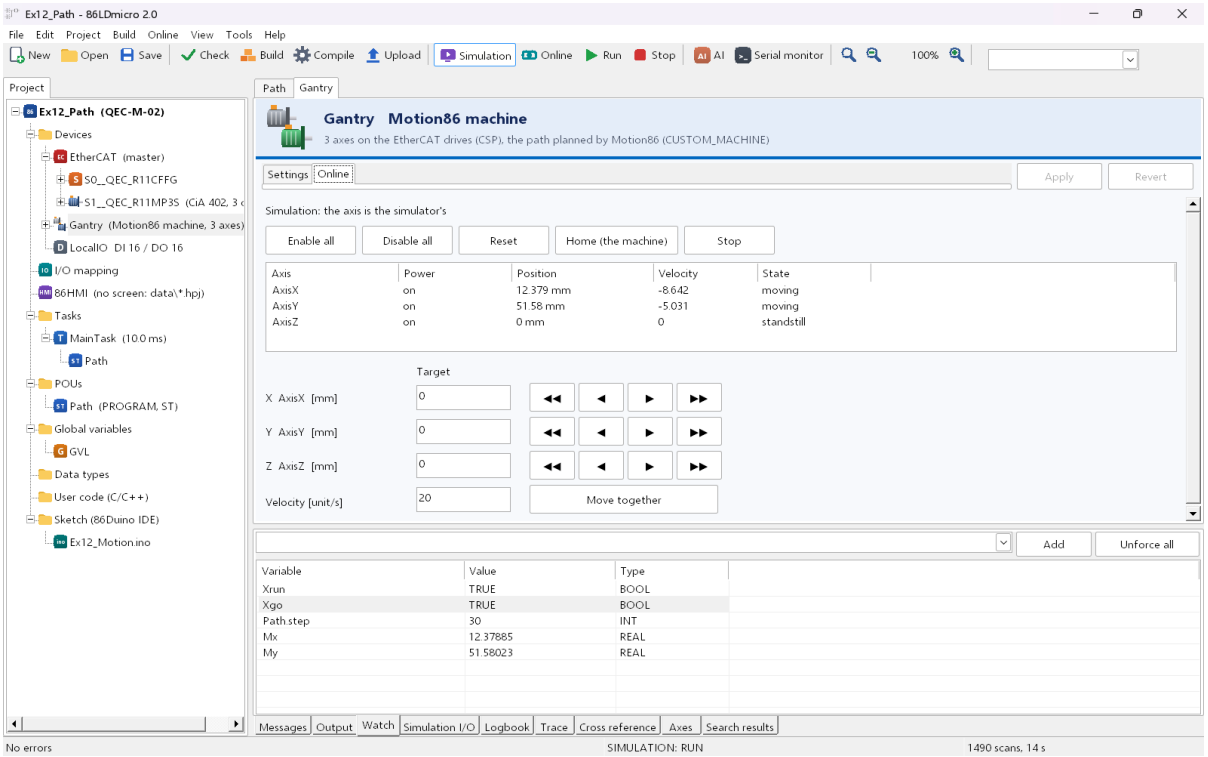


Figure 12-1 The three axes of the Gantry machine moving in simulation: the machine and axes in the project tree, the "Online" tab of the machine page

Term	Description
Machine	A group of axes planned together; in the program it is specified with `Machine := Gantry`
Axis	A drive (or a virtual axis); the axis name is a constant in the program, for example `Axis := AxisX`
Axis letter	Axes 1 to 9 of a machine are X, Y, Z, U, V, W, A, B, C in order; G-code and the M86_* function blocks use these letters to specify axes
User units	Positions in the program are in units such as mm or deg, and velocities in unit/s; the PPU of the axis converts them to the drive's counts
PPU	Pulses per unit = pulses per revolution ÷ travel per revolution; for example 6400 counts/rev and a 10 mm screw lead → 640 counts/mm
Interpolated	One path for the whole machine (lines, arcs, G-code); all axes arrive at the same time
Independent	Each axis is planned on its own; the axes can run their own moves at the same time
CSP	Cyclic Synchronous Position: in every cycle the MainDevice sends the target position (607Ah) and the drive reports the actual position (6064h)

12.2 Creating a machine

Any of these three ways opens the "A new Motion86 machine" window:

- Right-click "Devices" in the project tree > "Add a device" > `motion86`.
- Right-click a CiA 402 drive (motor icon) in the project tree > "Use as an axis of a Motion86 machine".
- On the SubDevice page of the drive, click **Use as an axis of a Motion86 machine (CSP)...** (section 11.5).

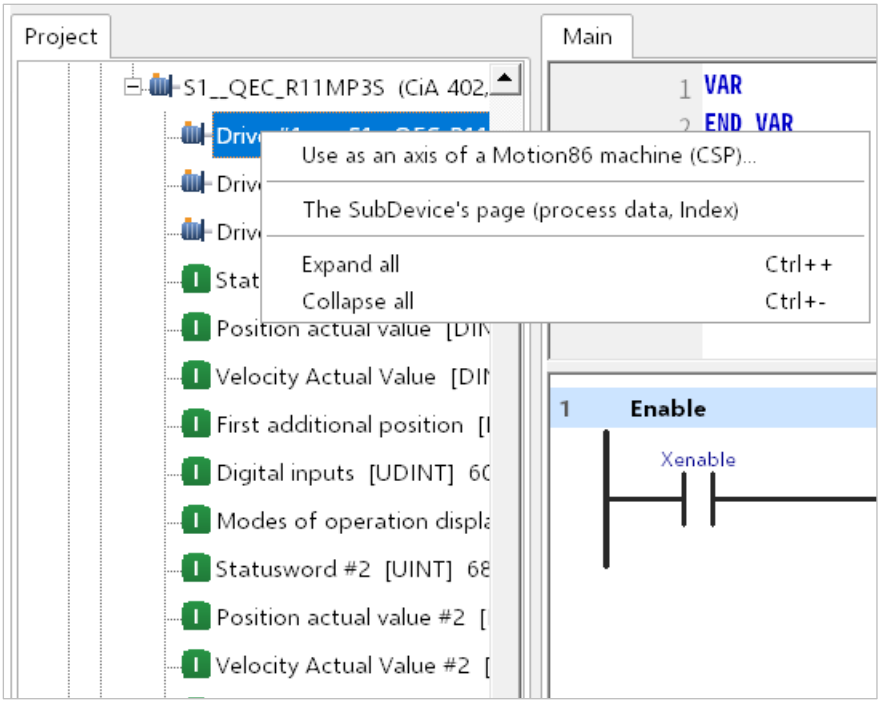


Figure 12-2 Right-clicking a drive: Use as an axis of a Motion86 machine

1. Enter the machine name (for example `Gantry`; letters, digits and _, not the name of another device or axis).
2. Select the number of axes (1 to 9). The available CiA 402 drives become the X, Y, Z, U, V, W, A, B, C axes in order; when no drive is available, the axes are first created as **virtual axes**, and you select the drive later in "Drive" on the axis page.
3. Click **OK**. The machine and its axes (motor icons) appear in the project tree; each axis is followed by its axis letter and its drive, for example `AxisX (X) -> S1__QEC_R11MP3S #1`.

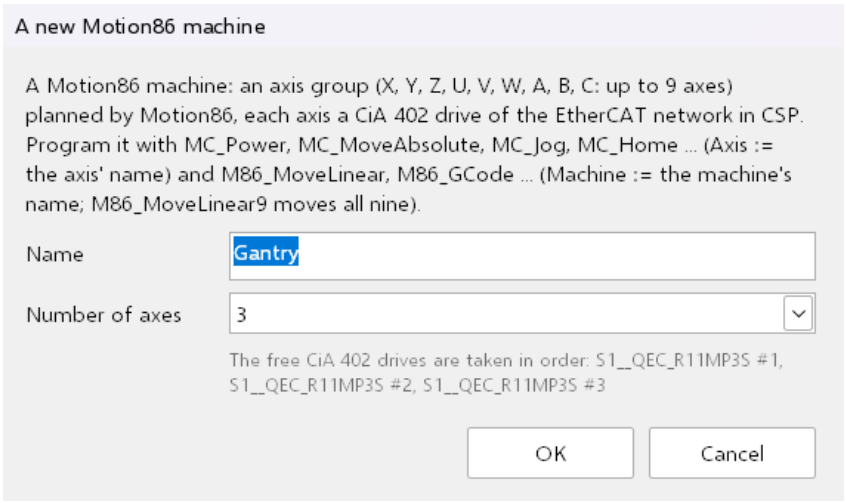


Figure 12-3 A new Motion86 machine: name and number of axes (1 to 9)

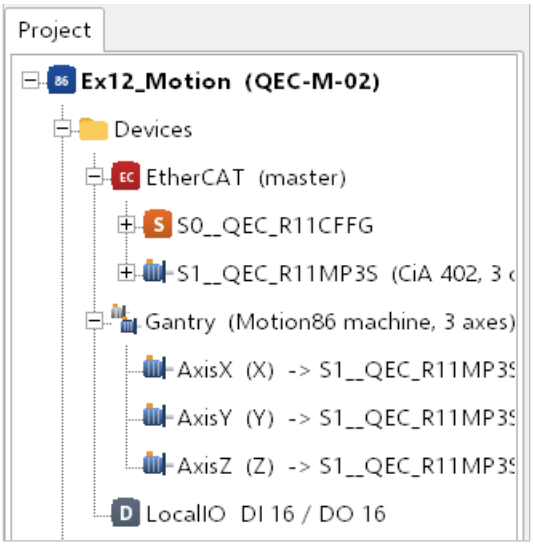


Figure 12-4 Project tree: the machine Gantry and its three axes, each followed by its drive

*__Tip:__ Axis names and machine names are constants (DINT) in the program, for example `Axis := AxisX`, `Machine := Gantry`. After renaming, change the program too, and click **Check** (Shift+F7) to find anything you missed.

12.3 Machine settings

Double-click the machine in the project tree to open the "**Settings**" tab of the machine page. After making changes, click **Apply** (stores them in the project; they reach the controller only after you build and upload); **Revert** discards the changes.

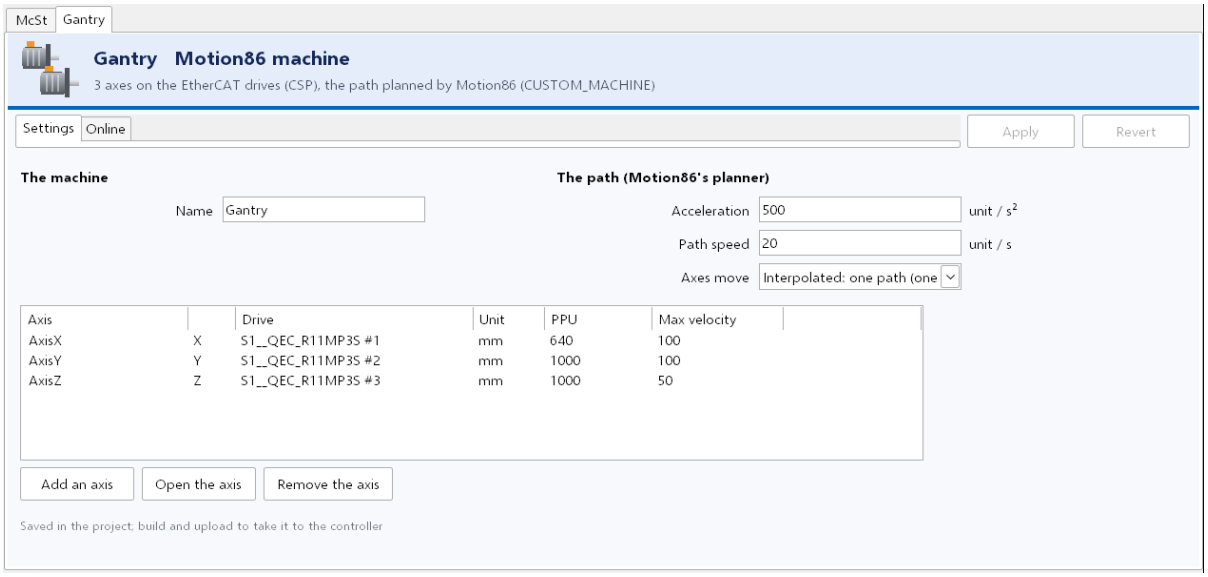


Figure 12-5 "Settings" of the machine page: name, path acceleration, path speed, axes move mode and axis list

Field	Description
Name	The machine name; in the program: `Machine := Gantry`
Acceleration (unit/s²)	The path acceleration / deceleration of the Motion86 planner; it determines the Acceleration / Deceleration of MC_*
Path speed (unit/s)	Used when the Feedrate of M86_MoveLinear is 0
Axes move	**Interpolated**: one path (one Motion86 machine); the moves of the same machine are queued in order; **Independent**: each axis moves on its own (one Motion86 machine per axis), so the axes can move at the same time
Axis list	Axis, letter, drive, unit, PPU, maximum velocity; click **Add an axis**, **Open the axis**, **Remove the axis** (the drive stays in the EtherCAT network)

***Note:** There can be at most **3 Motion86 machines** on EtherCAT drives; in independent mode each axis takes one, so the whole project can have at most 3 independent axes (or 1 interpolated machine + 2 independent axes). For a machine with more than 3 axes, use **Interpolated** mode (one machine can carry 9 axes). Beyond the limit, "Apply" and the build report an error. In independent mode, M86_MoveArcXY / M86_GCode are not executed.

12.3.1 Multi-axis machines (up to 9 axes)

A machine can have 1 to 9 axes, corresponding to X, Y, Z, U, V, W, A, B, C in order. Both the project tree and the axis list show the letter of each axis.

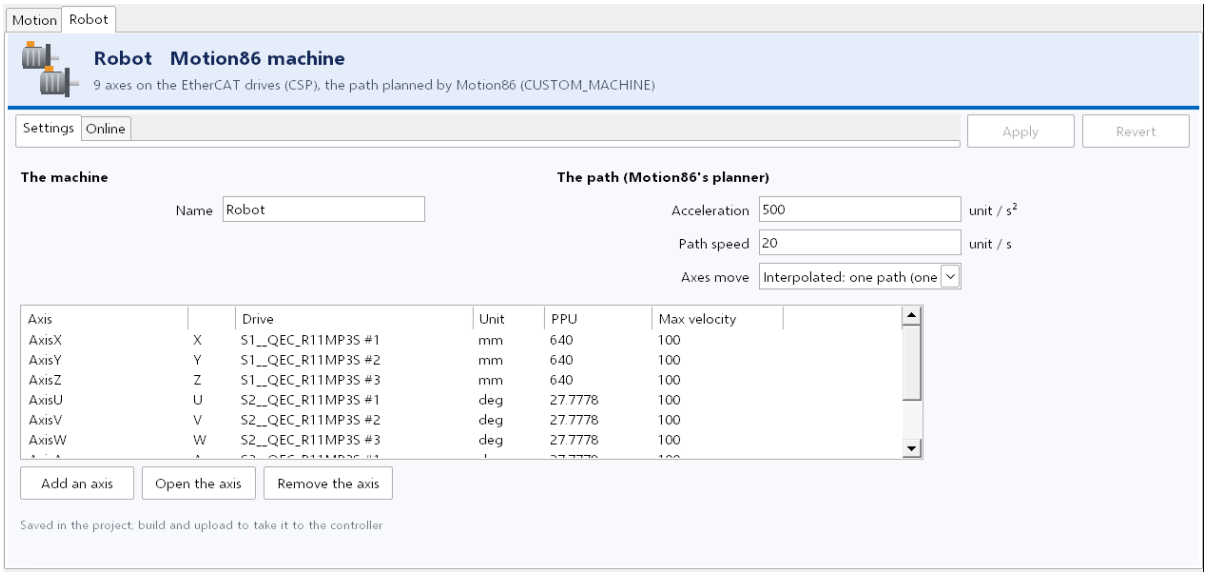


Figure 12-6 "Settings" of the 9-axis machine Robot: the second column of the axis list is the axis letter (X to C); the list scrolls up and down

1. Select 9 in "Number of axes" when you create the machine; or click **Add an axis** on the machine page to add one axis at a time (up to 9 axes).
2. Double-click each axis, select its CiA 402 drive in "Drive" on the axis page, and set the unit, the counts per revolution and the travel per revolution (section 12.4). Use mm for linear axes and deg for rotary axes.
3. Keep "Axes move" at **Interpolated**, click **Apply**, then build and upload.

12.4 Axis settings

Double-click an axis in the project tree to open the "**Settings**" tab of the axis page; after making changes, click **Apply**.

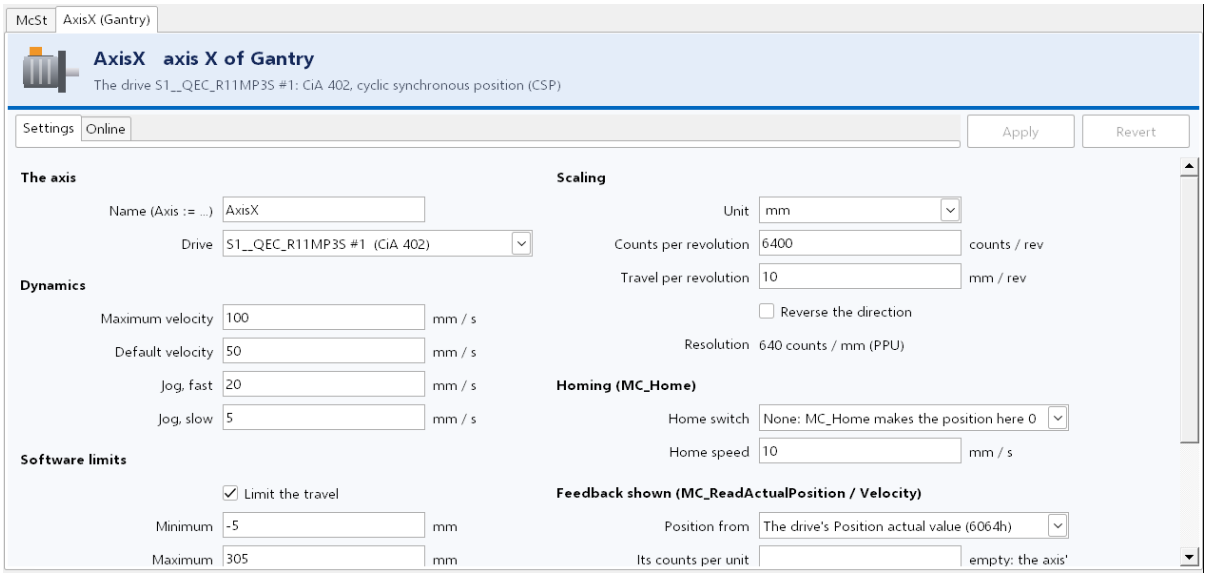


Figure 12-7 "Settings" of the axis page: drive, dynamics, software limits, scaling and homing

Section	Field	Description
Axis	Name (Axis := ...)	The axis name in the program
Axis	Drive	For example `S1__QEC_R11MP3S #1`; for a multi-axis drive select #1 / #2 / #3; you can also select a virtual axis (no drive, the position follows the command)
Dynamics	Max velocity, Default velocity	unit/s; the default velocity is used when the Velocity of MC_* is 0
Dynamics	Jog, fast / Jog, slow	The speeds of the jog buttons on the axis page and of MC_Jog (when Velocity is 0)
Software limits	Limit the travel, Minimum, Maximum	When ticked, moves beyond the range are rejected
Scaling	Unit, Counts per revolution, Travel per revolution	The Resolution (PPU) is calculated and shown automatically
Scaling	Reverse the direction	Tick it when the motor direction is opposite to the coordinates
Homing (MC_Home)	Home switch	**none**: MC_Home sets the current position to 0; **The drive's home switch (60FDh bit 2)**; or **a BOOL of the GVL**
Homing (MC_Home)	Home speed	The speed for searching the home switch (1/10 of it when leaving the switch)

***Note: The simulator does not apply the software limits: in simulation, commands such as MC_MoveAbsolute still move normally beyond the limits. When built for the controller, the limits are handled by the position limits of Motion86 (with the minimum and maximum of "Limit the travel").** Check how the controller actually reacts when a limit is exceeded on the real machine, at low speed and over a short distance, then design the error handling accordingly.

12.4.1 Feedback shown

The "**Feedback shown (MC_ReadActualPosition / Velocity)**" section determines where the position and velocity shown on the axis page, and the values read by MC_ReadActualPosition / MC_ReadActualVelocity, come from:

Field	Description
Position from	Default: **The drive's Position actual value (6064h)**. You can select any TxPDO entry of the EtherCAT network instead (for example the encoder position of QEC-R11MP3S); value shown = `entry value ÷ its counts per unit + its offset`
Its counts per unit	The counts of the selected entry per unit; **empty = the axis settings** (this axis' PPU, including Reverse the direction); enter a negative number to reverse it
Its offset	An offset added to the value shown, in the axis' unit
Velocity from	Default: **The axis' (Motion86, from the moves)**. You can select a TxPDO entry instead (for example 606Ch velocity actual value); MC_ReadActualVelocity = `entry value ÷ its counts/s per unit/s`
Its counts/s per unit/s	The conversion of the velocity entry; empty = PPU

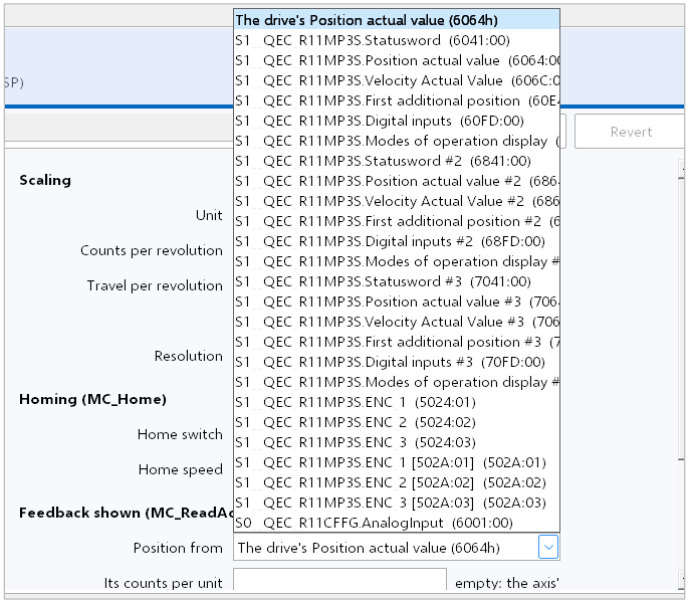


Figure 12-8 "Feedback shown" in "Settings" of the axis page: a TxPDO entry selected as the position source

*__Important:__ The position / velocity source **affects only what is shown** and **MC_ReadActualPosition / MC_ReadActualVelocity**. CSP control, the in-position check, homing, the software limits, M86_ReadPosition and the coordinates on the machine page still use the drive's 6064h.

MC_Home and MC_SetPosition do not change the reading of an external entry: after homing, the feedback shown does not return to zero (a limitation of 86LDmicro 2.0.0). Use "Its offset" to align it when needed.

An external encoder (for example 60E4h) usually has a different resolution from the motor, so enter the counts per unit of that encoder in "Its counts per unit"; if it is empty, the motor's PPU is used, and with the wrong conversion the value looks magnified and jumps.

In simulation there are no EtherCAT values, so the position of the drive (the simulator) is shown. If the selected entry is no longer in the ENI, the build warns and the drive's value is shown instead.

12.5 Commissioning the machine

After you start the simulation (F5) or go online (Ctrl+F5), switch to the "**Online**" tab of the machine page to operate the whole machine without writing a program. When neither simulating nor online, the page says "Offline: start the simulation (F5) or go online (Ctrl+F5) to try the axis".

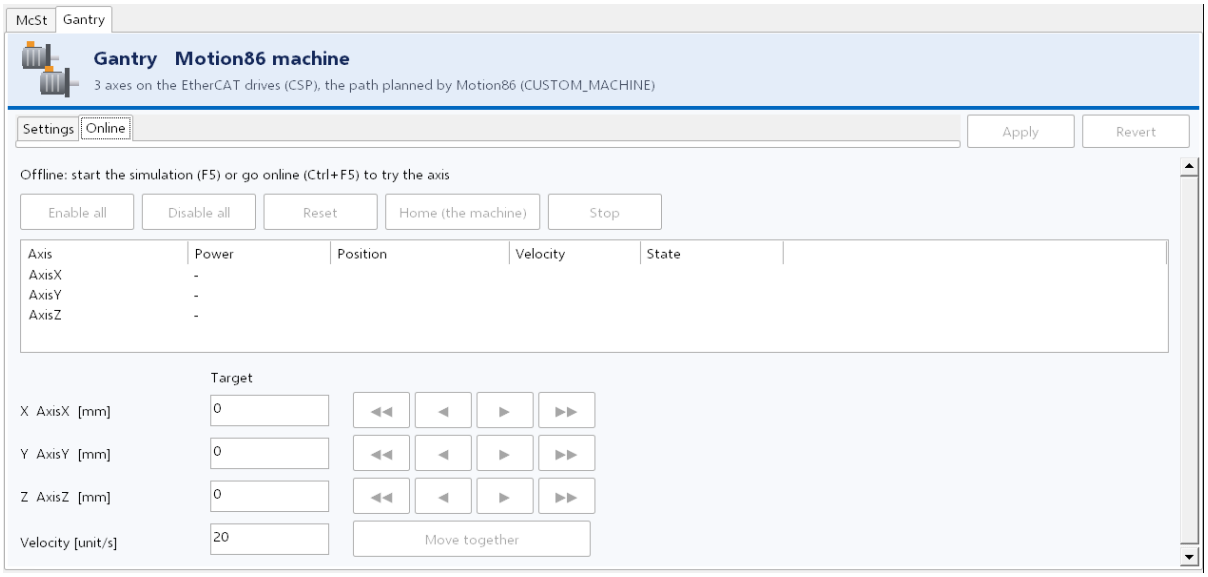


Figure 12-9 The "Online" tab of the machine page: the status of each axis at the top, the target, velocity and jog at the bottom

Button / field	Description
Enable all	The drives of all axes go to Operation Enabled, and the "Power" column shows ON
Disable all	Stops first, then disables all drives
Jog ◀◀ ◀ ▶ ▶▶	One row per axis: fast backward, slow backward, slow forward, fast forward; **moves only while held, stops when released**
Target, Velocity, Move together	Enter the position in the "Target" field and unit/s in the "Velocity" field of each axis, then click **Move together**: in interpolated mode the axes make one linear move and arrive at the same time; in independent mode the axes start at the same time and each reaches its position on its own
Home (the machine)	The axes with a home switch set search for home one after another; axes without a switch keep their position
STOP	Stops all axes immediately
Reset	Clears the errors

1. Click **Enable all** and check that "Power" is ON for each axis.
2. Use the **slow jog** (◀ ▶) to check the direction of each axis.
3. Enter X 80, Y 40, Z 20 in "Target" and 20 in "Velocity", and click **Move together**.
4. When you are done, click **Disable all**.

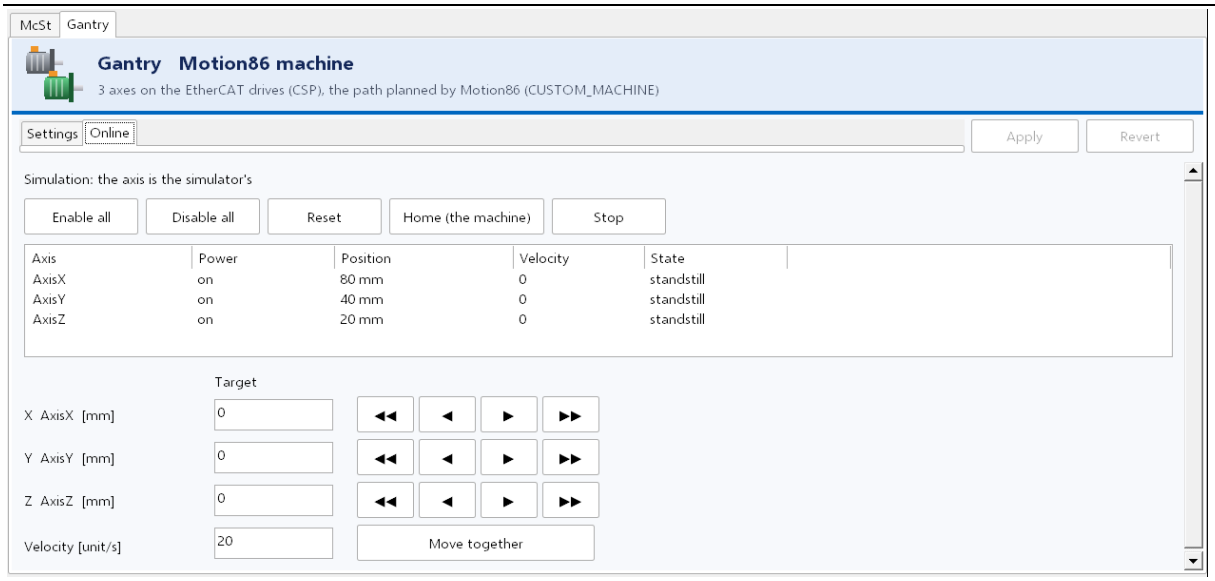


Figure 12-10 Gantry online: the three axes powered on and moved to X 80, Y 40, Z 20 mm

With more than 3 axes, each axis has its own row of target and jog buttons, and the number of rows grows with the number of axes (the layout depends on the window size); when it does not fit on the screen, scroll down with the mouse wheel or the scroll bar on the right to find "Velocity" and Move together:

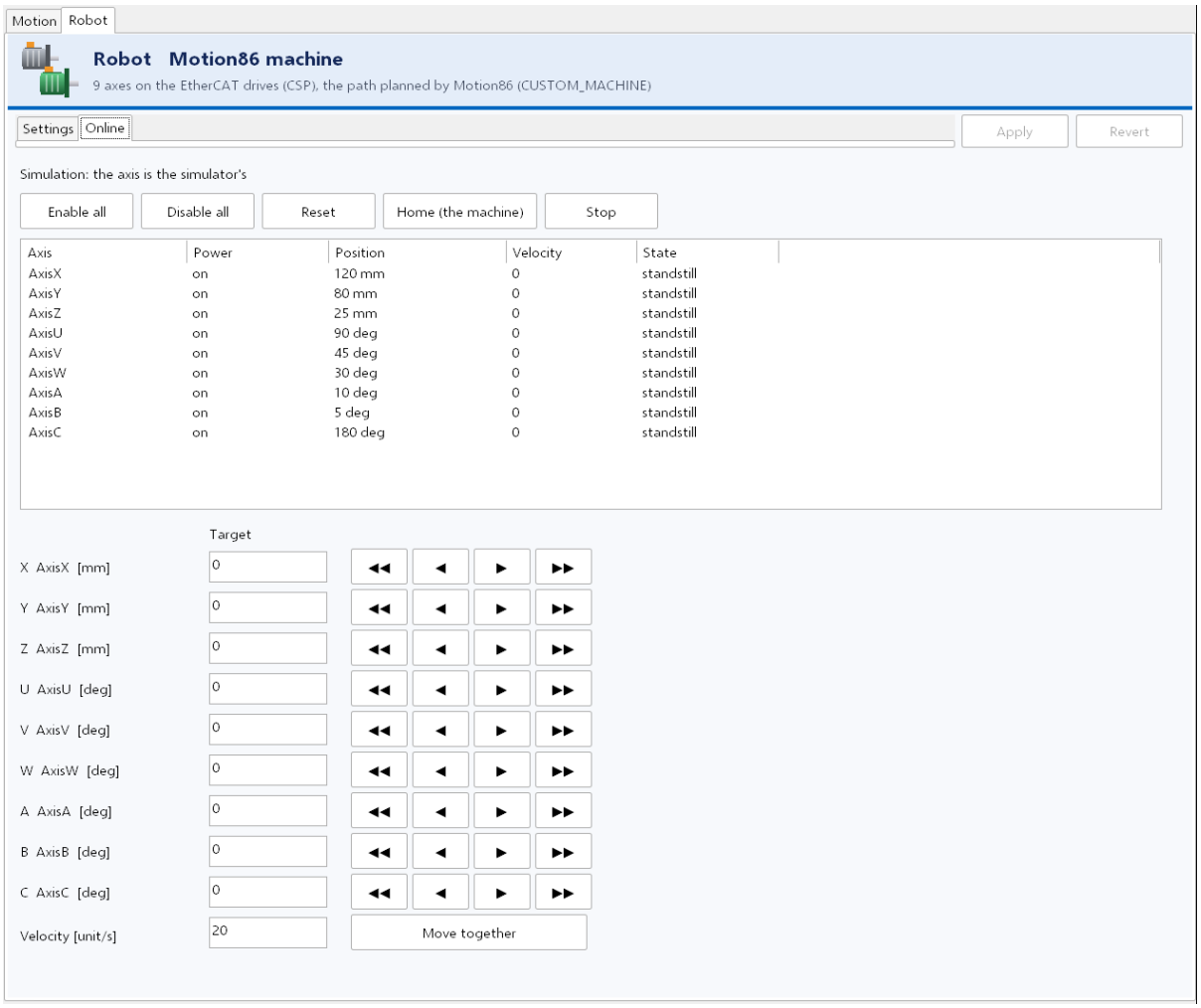


Figure 12-11 "Online" of a 9-axis machine: the status of each axis at the top, the target, jog and "Move together" of each axis at the bottom

12.6 Commissioning a single axis

The "**Online**" tab of the axis page operates a single axis and shows its position, velocity, following error, command and a trend of the last 30 seconds:

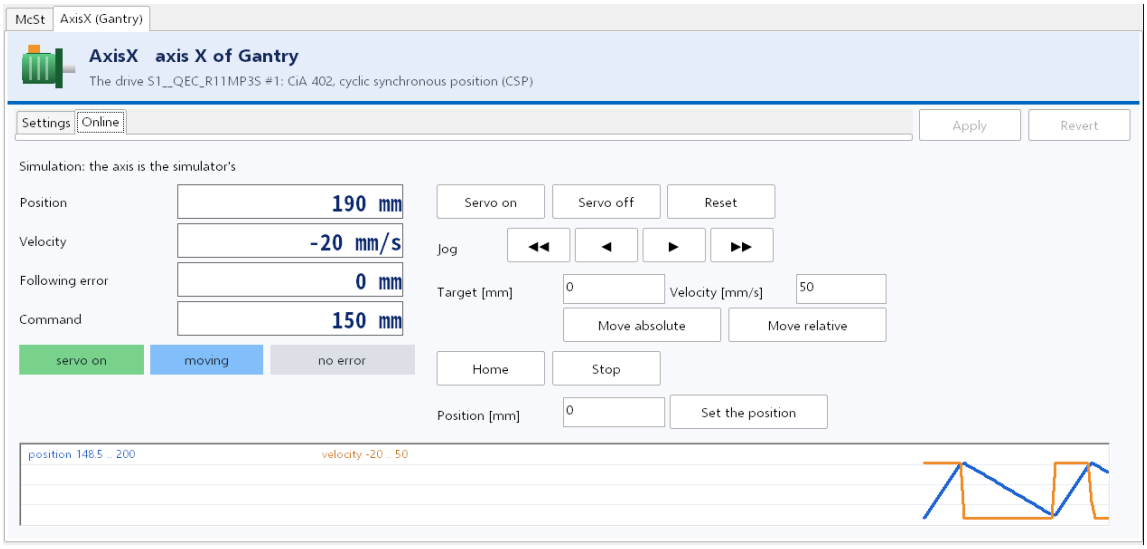


Figure 12-12 "Online" of the axis page: position, velocity, following error, command, status lights, power, jog, move, home, set position and trend chart

Button / field	Description
Enable / Disable / Reset	Powers the drive on or off; Reset clears the errors
Status lights	Enabled, standstill / moving, error
Jog ◀◀ ◀ ▶ ▶▶	Moves only while held (the command is renewed every 200 ms; the controller stops automatically when it receives nothing for 600 ms)
Move absolute / Move relative	Enter the target (or the distance) and the velocity
Home	According to the home switch setting of the axis (section 12.4)
STOP	Stops immediately
Set the position	Sets the current position to the value entered, without moving the drive (like MC_SetPosition)
Trend	Position (blue) and velocity (orange) of the last 30 seconds, shown in simulation or online

12.6.1 Axes window

"View > Axes" lists the power, position and command status of all the axes of the project, and lets you power on, jog, move, home and STOP them directly; double-click a Motion86 axis to open its axis page. You can also open it by right-clicking a machine in the project tree > "The Axes panel (commissioning)".

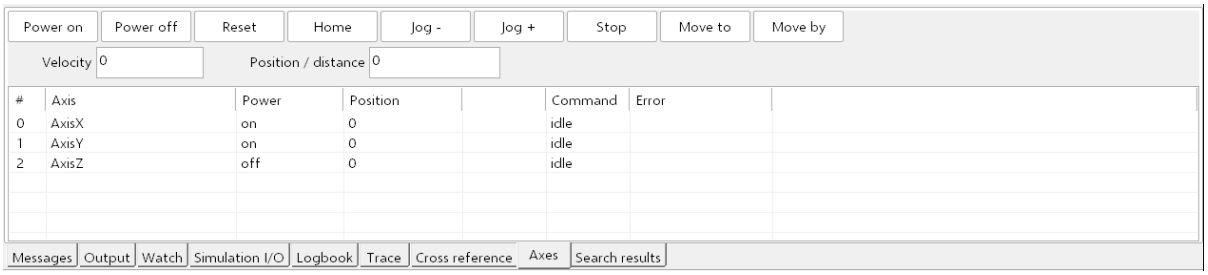


Figure 12-13 Axes window: the status and operation of all axes

12.7 PLCopen MC function blocks

In the program, the PLCopen MC_* function blocks control a **single axis**. Call them directly in ST and SFC; in a ladder diagram, insert them with **CALL** (sections 7.8, 7.11).

12.7.1 Basic rules

- `Axis :=` takes the axis name (for example `AxisX`); the axis name is a DINT constant.
- Positions and distances are in the axis' unit (mm…), velocities in unit/s; all parameters are of type **REAL**. When Velocity is 0, the default velocity of the axis is used.
- Acceleration / Deceleration are determined by the acceleration of the machine (the values passed in are kept for compatibility).
- Declare the function block instances in VAR, and **call them in every scan** (or at least while Execute changes and while waiting for Done).
- In interpolated mode, the MC_* moves of the same machine are queued in order; for a simultaneous linear move of several axes, use M86_MoveLinear (section 12.8).

12.7.2 Common behavior of the function blocks

Type	Trigger	Outputs
Execute type (MoveAbsolute, MoveRelative, MoveVelocity, Home, Stop, Halt, Reset, SetPosition)	The **rising edge** of `Execute` sends the command; to send it again, first set Execute back to FALSE	`Busy` (executing) → `Done` (completed) or `Error` (`ErrorID`); `CommandAborted` when replaced by a new command. Done / Error stay on while Execute is TRUE
Enable type (Power, ReadActualPosition, ReadActualVelocity, ReadStatus, ReadAxisError)	Active as long as `Enable` is TRUE	`Valid` (or `Status` of MC_Power) and the values read
MC_Jog	Moves while `JogForward` / `JogBackward` is TRUE (stops when both are TRUE or both are FALSE)	`Busy`, `Done`, `Error`

12.7.3 Function blocks at a glance

Function block	Inputs	Outputs	Description
MC_Power	Axis, Enable	Status, Valid, Error, ErrorID	Powers on the axis; Status TRUE means it is powered on
MC_MoveAbsolute	Axis, Execute, Position, Velocity, Acceleration, Deceleration	Done, Busy, Active, CommandAborted, Error, ErrorID	Moves to an absolute position
MC_MoveRelative	Axis, Execute, Distance, Velocity, Acceleration, Deceleration	Same as above	Moves by a distance
MC_MoveVelocity	Axis, Execute, Velocity, Acceleration, Deceleration	InVelocity, Busy, Active, CommandAborted, Error, ErrorID	Moves continuously at a velocity until MC_Stop
MC_Jog	Axis, JogForward, JogBackward, Velocity, Acceleration, Deceleration	Done, Busy, CommandAborted, Error, ErrorID	Jog (must be called continuously)
MC_Home	Axis, Execute, Method, Velocity, Acceleration	Done, Busy, Active, CommandAborted, Error, ErrorID	Homing (according to the home switch setting of the axis)
MC_Stop	Axis, Execute, Deceleration	Done, Busy, CommandAborted, Error, ErrorID	Stop (stops immediately)
MC_Halt	Axis, Execute, Deceleration	Done, Busy, Active, CommandAborted, Error, ErrorID	Halt (stops immediately)
MC_Reset	Axis, Execute	Done, Busy, Error, ErrorID	Clears the errors
MC_SetPosition	Axis, Execute, Position	Done, Busy, Error, ErrorID	Sets the current position to Position (the drive does not move)
MC_ReadActualPosition	Axis, Enable	Valid, Busy, Error, ErrorID, Position	Current position (REAL)
MC_ReadActualVelocity	Axis, Enable	Valid, Busy, Error, ErrorID, Velocity	Current velocity
MC_ReadStatus	Axis, Enable	Valid, Busy, Error, ErrorID, ErrorStop, Disabled, Stopping, Homing, StandStill, DiscreteMotion, ContinuousMotion, SynchronizedMotion	Status of the axis
MC_ReadAxisError	Axis, Enable	Valid, Busy, Error, ErrorID, AxisErrorID	Error code of the axis

12.7.4 Example: power on a single axis, home it, move back and forth three times

Scenario: The X axis of Gantry: once the enable switch Xenable is on, pressing Xstart homes the X axis, moves it to 200 mm, then back 50 mm, and after 3 round trips it stops at 150 mm; Xstop stops the axis and returns to standby (pressing Xstart again starts over from homing); when an error occurs, the program stays at the error step and Yerror lights up, and pressing Xreset resets it and returns to standby. Holding Xstart down runs the sequence only once.

Variable	Type	I/O mapping	Purpose
Xenable	BOOL	Digital input	Enable switch
Xstart	BOOL	Digital input	Start button
Xstop	BOOL	Digital input	Stop button
Xreset	BOOL	Digital input	Error reset button
Ybusy	BOOL	Digital output	Busy lamp
Yerror	BOOL	Digital output	Error lamp
Mpos	REAL	—	Current position of the X axis (mm)
step, count	INT	— (local variables of McSt)	Step, number of round trips done

```
PROGRAM McSt
VAR
    pwr  : MC_Power;
    home : MC_Home;
    mv   : MC_MoveAbsolute;
    rel  : MC_MoveRelative;
    stp  : MC_Stop;
    rst  : MC_Reset;
    rd   : MC_ReadActualPosition;
    st   : MC_ReadStatus;
    ae   : MC_ReadAxisError;
    startEdge : R_TRIG;      // a new press of Xstart
    resetEdge : R_TRIG;      // a new press of Xreset
    resetReq  : BOOL;        // MC_Reset requested
    step  : INT;             // 0 standby, 10 home, 20 forward, 30 back, 90 stopping, 99 error
    count : INT;             // round trips done
    lastError : DINT;        // ErrorID of the last error
END_VAR
pwr(Axis := AxisX, Enable := Xenable);
rd(Axis := AxisX, Enable := TRUE);
Mpos := rd.Position;
st(Axis := AxisX, Enable := TRUE);
ae(Axis := AxisX, Enable := TRUE);
startEdge(CLK := Xstart);
resetEdge(CLK := Xreset);

(* 1. While moving, errors and stop come first: end the current command *)
IF step >= 10 AND step <= 30 THEN
    IF home.Error OR mv.Error OR rel.Error OR st.ErrorStop THEN
        IF home.Error THEN lastError := home.ErrorID;
        ELSIF mv.Error THEN lastError := mv.ErrorID;
        ELSIF rel.Error THEN lastError := rel.ErrorID;
        ELSE lastError := ae.AxisErrorID;     (* only the axis itself went to ErrorStop *)
        END_IF;
        step := 99;
    ELSIF Xstop THEN
        step := 90;
    END_IF;
END_IF;

(* 2. Steps *)
CASE step OF
```

```
0:  IF pwr.Status AND startEdge.Q AND NOT Xstop THEN   (* a new press, and stop is not held *)
        count := 0;
        step := 10;
    END_IF;
10: IF home.Done THEN step := 20; END_IF;
20: IF mv.Done THEN step := 30; END_IF;
30: IF rel.Done THEN
        count := count + 1;
        IF count < 3 THEN step := 20; ELSE step := 0; END_IF;
    END_IF;
90: IF stp.Error THEN
        lastError := stp.ErrorID;
        step := 99;
    ELSIF stp.Done AND NOT Xstop THEN    (* stopped: back to standby; Xstart starts again from homing
*)
        step := 0;
    END_IF;
99: IF resetEdge.Q THEN resetReq := TRUE; END_IF;   (* reset by the operator *)
    IF rst.Error THEN                   (* reset failed: record the error; Xreset can be pressed again
*)
        lastError := rst.ErrorID;
        resetReq := FALSE;
    ELSIF rst.Done THEN
        resetReq := FALSE;
        IF NOT st.ErrorStop THEN step := 0; END_IF;
    END_IF;
END_CASE;

(* 3. Commands: Execute is TRUE only in its own step; leaving the step ends the command,
      and returning to the step gives a new rising edge *)
home(Axis := AxisX, Execute := step = 10);
mv(Axis := AxisX, Execute := step = 20, Position := 200.0, Velocity := 50.0);
rel(Axis := AxisX, Execute := step = 30, Distance := -50.0, Velocity := 20.0);
stp(Axis := AxisX, Execute := step = 90);
rst(Axis := AxisX, Execute := resetReq AND step = 99);
Ybusy := step >= 10 AND step <= 30;
Yerror := step = 99;
END_PROGRAM
```

#	Operation
1	Read the ENI as in Chapter 11 (for example import `data\86ld\ethercat\QEC-ENI-1.xml` of ex30_dispense), and create the 3-axis machine Gantry (AxisX, AxisY, AxisZ) as in section 12.2.
2	Declare Xenable, Xstart, Xstop, Xreset, Ybusy, Yerror (BOOL) and Mpos (REAL) in the GVL.
3	"Project > Add a PROGRAM (ST)...", name it `McSt`, and enter the program above.
4	Click **Check** (Shift+F7) and make sure there are no errors.
5	Click **Simulate** (F5). Add Xenable, Xstart, Mpos, Ybusy, McSt.step, McSt.count to the Watch window.
6	Write TRUE to Xenable, then write TRUE to Xstart (and write FALSE back after about 1 second). Open the "Online" tab of the AxisX axis page to watch.

Expected result: The X axis homes (without a home switch, the current position is set to 0), moves to 200 mm at 50 mm/s (about 4 seconds), then back to 150 mm at 20 mm/s; after 3 repetitions it stops at 150 mm, McSt.step returns to 0 and Ybusy goes off. Holding Xstart down does not run the sequence again automatically after it finishes; pressing Xstart while Xstop is held does not start it. Writing TRUE to Xstop during a move stops the axis, and step goes through 90 back to 0; release Xstop and press Xstart again to start over from homing.

***Note: On an error, the program stays in step 99 (Yerror lights up, lastError records the ErrorID) until Xreset is pressed to reset it, and returns to step 0 after a successful reset; errors are checked only in the move steps, so it does not keep returning to 99; if the reset fails (rst.Error), the ErrorID is recorded and you can press Xreset again to retry.**

The simulator does not produce axis errors and does not apply the software limits (section 12.4); check the error and reset sequence on the real machine in a controllable way (for example at low speed and over a short distance), and first make sure that stop and emergency stop work.

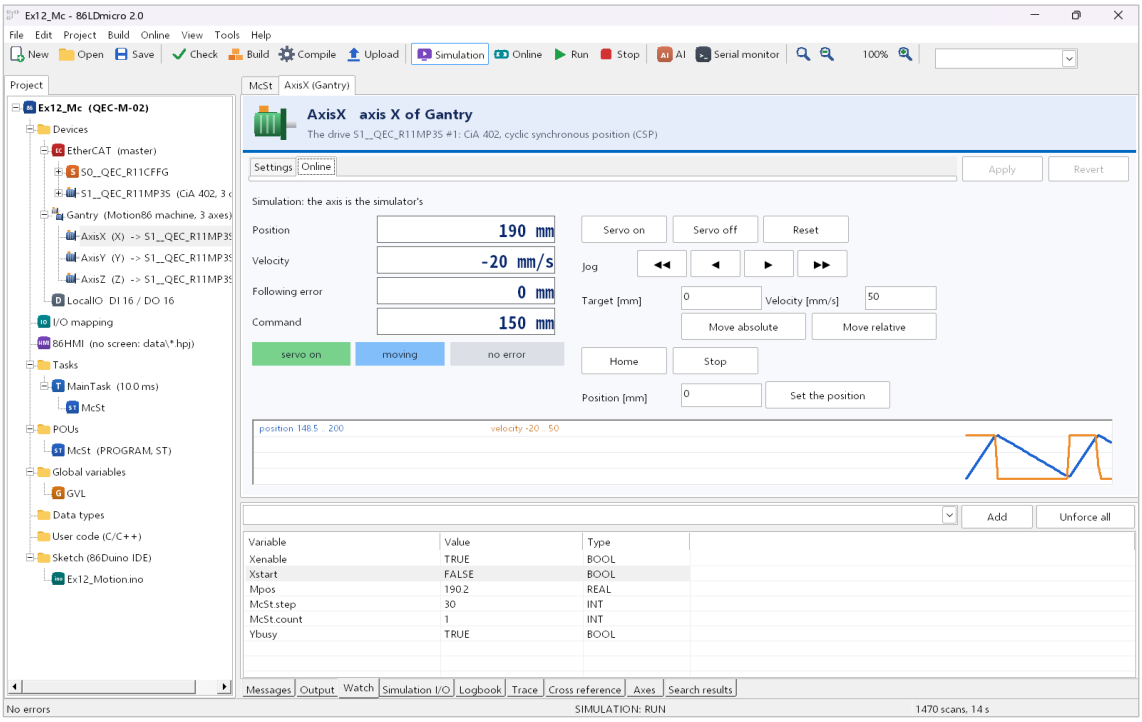


Figure 12-14 McSt in simulation: step, count, Mpos in the Watch window and the trend on the AxisX axis page

***Tip:** When you use MC blocks in SFC, the actions of a step are no longer executed once the step is left; set the Execute of the previous move back to FALSE in the next step, so that there is a new rising edge the next time the step is reached. It is recommended to call MC_Power permanently from another ST or LD program (section 8.6).

12.8 M86 machine function blocks

The M86_* function blocks control the **whole machine**: simultaneous multi-axis linear and circular interpolation, and G-code. `Machine :=` takes the machine name.

Function block	Inputs	Outputs	Description
M86_Power	Machine, Enable	Status	Powers on all axes of the machine
M86_MoveLinear	Machine, Execute, X, Y, Z, Feedrate	Done, Busy, Error	Linear interpolation to X, Y, Z (Feedrate in unit/min; 0 uses the path speed of the machine); the U to C axes do not move
M86_MoveLinear9	Machine, Execute, X, Y, Z, U, V, W, A, B, C, Feedrate	Done, Busy, Error	One linear path for all axes of the machine (up to 9 axes), all arriving at the same time; with fewer axes, the extra inputs are not used
M86_MoveArcXY	Machine, Execute, CX, CY, X, Y, Clockwise, Feedrate	Done, Busy, Error	Arc in the XY plane (center CX, CY; end point X, Y)
M86_Home	Machine, Execute	Done, Busy, Error	Homes the whole machine
M86_GCode	Machine, Execute, Line	Done, Busy, Error	Executes one line of G-code, for example `'G1 X50 Y0 F3000'` (axes not written do not move)
M86_Stop	Machine, Execute, Emergency	Done	Stops the machine
M86_ReadPosition	Machine, Enable	Valid, X to C, Moving	Current position of each axis and whether it is moving
M86_Override	Machine, Ratio	—	Velocity override (0 to 2)

- `Feedrate` is in **unit/min** (for example 1200 = 20 mm/s); 0 means the path speed of the machine is used.
- A move is sent only once, when `Execute` changes from FALSE to TRUE; to move again, first set Execute back to FALSE.
- The coordinates of M86_* are the same as those of MC_* (including the offset of MC_SetPosition); G-code strings are in machine coordinates. M86_MoveLinear, M86_MoveLinear9 and M86_MoveArcXY switch back to absolute coordinates (G90) before executing; M86_GCode does not change the mode: once G91 has been sent, G-code runs in relative coordinates, so send G90 first.
- A machine in independent mode does not execute M86_MoveArcXY and M86_GCode (section 12.3).

12.8.1 Example: line, arc and G-code

Scenario: After Gantry is powered on (`Xrun`), pressing `Xgo` moves XY in a straight line to (50, 30), draws a counterclockwise semicircle around the center (25, 30) to (0, 30), then moves to (10, 10) with G-code; `Mx` and `My` show the current position.

Variable	Type	I/O mapping	Purpose
Xrun	BOOL	Digital input	Machine power on
Xgo	BOOL	Digital input	Run the path
Mx, My	REAL	—	Current X, Y (mm)

```
PROGRAM Path
VAR
    mp  : M86_Power;
    lin : M86_MoveLinear;
    arc : M86_MoveArcXY;
    gc  : M86_GCode;
    rp  : M86_ReadPosition;
    step : INT;
END_VAR
mp(Machine := Gantry, Enable := Xrun);
rp(Machine := Gantry, Enable := TRUE);
CASE step OF
0:  IF mp.Status AND Xgo THEN step := 20; END_IF;
20: lin(Machine := Gantry, Execute := TRUE, X := 50.0, Y := 30.0, Z := 0.0, Feedrate := 1200.0);
    IF lin.Done THEN lin(Machine := Gantry, Execute := FALSE); step := 30; END_IF;
30: arc(Machine := Gantry, Execute := TRUE, CX := 25.0, CY := 30.0, X := 0.0, Y := 30.0,
        Clockwise := FALSE, Feedrate := 600.0);                    (* semicircle: (50,30) → (0,30) *)
    IF arc.Done THEN arc(Machine := Gantry, Execute := FALSE); step := 40; END_IF;
40: gc(Machine := Gantry, Execute := TRUE, Line := 'G1 X10 Y10 F3000');   (* the lin and arc before
have switched back to G90: absolute coordinates *)
    IF gc.Done THEN gc(Machine := Gantry, Execute := FALSE); step := 0; END_IF;
END_CASE;
Mx := rp.X;
My := rp.Y;
END_PROGRAM
```

#	Operation
1	Use the project of 12.7.4 (the machine Gantry in interpolated mode); declare Xrun, Xgo (BOOL) and Mx, My (REAL) in the GVL.
2	"Project > Add a PROGRAM (ST)...", name it `Path`, and enter the program above; remove McSt in the task settings (the two programs must not control Gantry at the same time).
3	Click **Check**, then **Simulate** (F5); add Xrun, Xgo, Mx, My, Path.step to the Watch window.
4	Write TRUE to Xrun, then write TRUE to Xgo; open the "Online" tab of the Gantry machine page to watch.

Expected result: The straight move to (50, 30) takes about 3 seconds (20 mm/s), the semicircle to (0, 30) about 8 seconds (10 mm/s), and after the G-code move to (10, 10) Path.step returns to 0. If Xgo stays TRUE, the same path runs again.

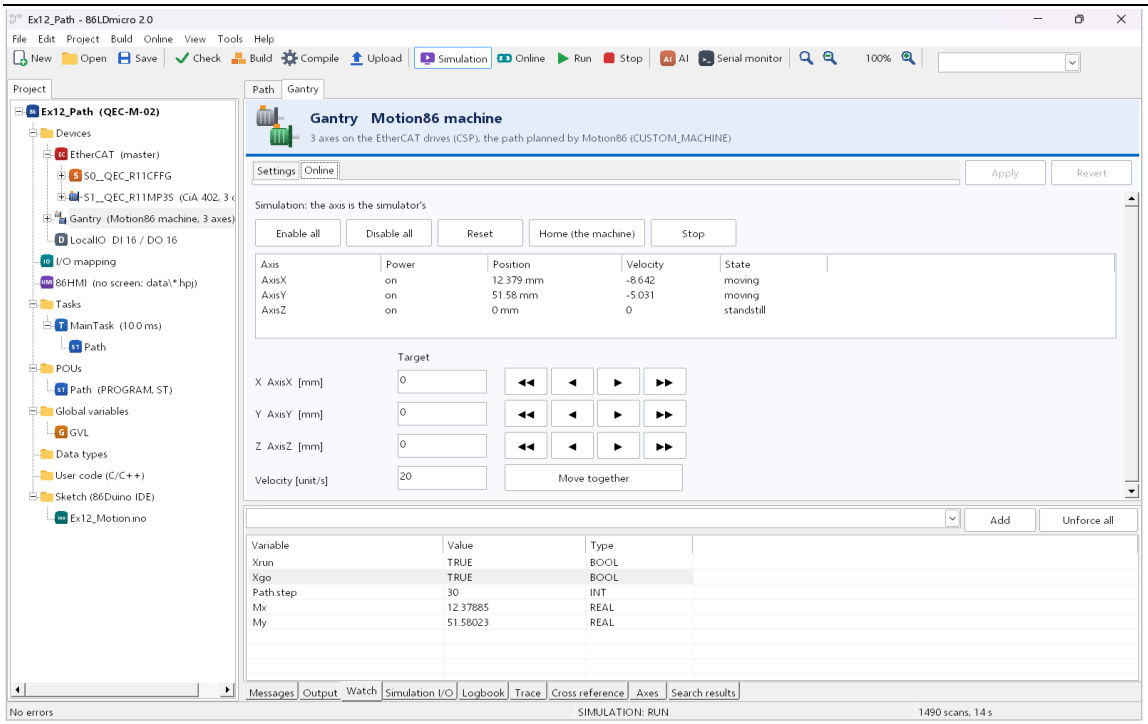


Figure 12-15 Path in simulation: the X, Y position shown on the machine page while the arc is drawn

12.8.2 Example: moving 9 axes together

The 9-axis machine Robot (AxisX to AxisC): it is powered on while Xpower is TRUE; pressing Xgo9 uses M86_MoveLinear9 to make the 9 axes reach the target at the same time, and M86_ReadPosition reads back the position of each axis. Xpower and Xgo9 are BOOLs of the GVL (initial value FALSE), so the program does not power on or move by itself after it is loaded. The target coordinates are in the unit of each axis (mm for linear axes, deg for rotary axes); before the first run, make sure the targets are within the travel and lower the Feedrate; to stop, use M86_Stop or set Xpower to FALSE.

```
PROGRAM Motion
VAR
    pw  : M86_Power;
    all : M86_MoveLinear9;
    rd  : M86_ReadPosition;
END_VAR
pw(Machine := Robot, Enable := Xpower);                  (* power on: Xpower *)
all(Machine := Robot, Execute := Xgo9 AND pw.Status,    (* press Xgo9 to send one move *)
    X := 120.0, Y := 80.0, Z := 25.0, U := 90.0, V := 45.0, W := 30.0,
    A := 10.0, B := 5.0, C := 180.0, Feedrate := 6000.0);
rd(Machine := Robot, Enable := TRUE);   (* rd.X .. rd.C: current positions *)
END_PROGRAM
```

G-code can also specify the U to C axes, for example `M86_GCode` with `Line := 'G1 X10 U90 C180 F3000'`.

12.8.3 Controlling axes in C/C++

User code (C/C++, Chapter 14) can also control axes:

Function	Description
`AXIS(AxisX)`	The axis number (passed to the functions below)
`mc_power(ax, on)`, `mc_is_on(ax)`	Power on / off, whether it is powered on
`mc_move_abs(ax, pos, vel)`, `mc_move_rel(ax, dist, vel)`	Absolute / relative move (vel 0 uses the default velocity)
`mc_jog(ax, dir, vel)`	Jog: dir is +1 / −1 / 0; must be called continuously; stops automatically 0.6 seconds after the last call
`mc_stop(ax)`, `mc_home(ax)`, `mc_set_position(ax, pos)`, `mc_reset(ax)`	Stop, home, set the position, clear the errors
`mc_position(ax)`, `mc_velocity(ax)`	Current position, velocity
`mc_state(ax)`, `mc_done`, `mc_busy`, `mc_error`, `mc_moving`, `mc_error_code`	State (0 idle, 1 busy, 2 done, 3 error)
`M86_MACHINE(Gantry)`	The Motion86 `Machine *` (bypasses the coordinate offset and the jog management of 86LDmicro)

***Note:** Call mc_* in `plc86_user_loop` / `plc86_user_before` / `plc86_user_after`, not in `plc86_user_ethercat` (interrupt).

12.9 Safety mechanisms

86LDmicro and Motion86 have the following built-in protections, but they cannot replace the hardware emergency stop and safety circuit.

Situation	Action
PLC STOP, watchdog	The program no longer runs; all machines stop and jogging ends; ordinary outputs keep their last values (sections 10.4 and 10.5)
Warm restart / cold restart	Warm restart: RETAIN variables are kept and other variables (including the variables used for outputs) return to their initial values; cold restart: all variables (including RETAIN) return to their initial values. Both remove all forcing and leave the RUN / STOP state unchanged: from the next scan the outputs are written from the initial values (in RUN they are then recalculated by the program). Stop the machine first (STOP or MC_Stop / M86_Stop) before restarting
Jog not kept up to date	The buttons on the axis page, MC_Jog and `mc_jog` stop automatically when they are not updated for 600 ms
The drive leaves Operation Enabled while powered on (fault, Quick stop, STO, network loss)	The whole machine stops and **does not power on again by itself**; the Logbook records `Motion86: an axis' drive left operation enabled: the machine stopped (MC_Reset, MC_Power again)`; MC_Reset and then MC_Power are required
Powering off	Stop the machine first, then power off the drive
The axis is moved by hand while the drive is not powered on	The position reference is offset; the Logbook records `Motion86: an axis moved while its drive was off: …`; home again or use MC_SetPosition
Homing finds no home switch	The Logbook records `Motion86: homing found no home switch`
Software limits	When "Limit the travel" is checked, a move beyond the minimum / maximum is rejected

***Safety: MC_Stop / MC_Halt are immediate stops in Motion86, without a deceleration ramp.** On a real machine, design an external emergency stop and safety circuit (cutting the drive's main power or STO).
The first time you run on a real machine, use a low speed, check the direction and the travel, and have the emergency stop ready.

12.9.1 First test-run checklist

Before turning a motor on a real machine for the first time, check each item and tick it off. Two people are recommended: one operates the computer and one stands by the emergency stop.

#	Before power-up	Check
A1	The emergency stop is wired into the hardware safety circuit (cutting the drive's main power or STO); when it is pressed, the motor can definitely produce no torque	☐
A2	No people, tools or foreign objects are within the mechanism's range of movement; both ends of the travel have mechanical stops or limit switches	☐
A3	Motor power cables, encoder cables and EtherCAT cables are connected as described in the drive manual, and grounding is secure	☐
A4	The drive parameters (motor model, electronic gear, limits) are set as described in the drive manual	☐

#	Power-up, communication	Check
B1	The serial monitor shows `EtherCAT: running, N slaves, … sync`, and N matches the actual number of SubDevices	☐
B2	The online values on the SubDevice page update; the drive's Statusword (6041h) can be read	☐
B3	The ENI's PDOs include the 6040h / 607Ah outputs and the 6041h / 6064h inputs (11.3.6)	☐
B4	The machine and axis settings are correct: unit, pulses per revolution, travel per revolution, drive assignment	☐

1. First, in **simulation** (F5), operate once with "Online" on the machine page to get familiar with the button positions and directions.
2. Upload the program and connect, then open the "Axes" window or "Online" on the axis page.
3. Power on a **single axis** and check that the "enabled" lamp lights and the position does not jump.
4. **Jog slowly** (◄ ►), moving a short distance each time, to confirm the positive direction; if it is reversed, power off, check "Reverse the direction" on the axis page, then build and upload again.
5. Use a **relative move** of a short distance (for example 5 mm) to confirm that the actual travel equals the command; if it differs, check the travel per revolution.
6. Run **homing** and check that the axis stops at the correct position.
7. Only after every axis has been checked, use "Move together" on the machine page, raising the speed step by step from low to high.
8. Run your own MC / M86 program only at the end.

#	First motion	Check
C1	Each axis moves in the correct direction	☐
C2	Each axis moves the correct distance (command 5 mm, actual 5 mm)	☐
C3	The axes stop when you press STOP, use PLC STOP or unplug the EtherCAT cable	☐
C4	The hardware emergency stop stops all axes	☐
C5	Homing works correctly	☐

Chapter 13 86HMI human-machine interface

86HMI is the 86Duino touch-screen HMI. The screens (pages, buttons, lamps, numbers ...) are made with the **86HMI Editor** of the 86Duino IDE; 86LDmicro **binds** the widgets on the screen to PLC variables and links the screen with the program in simulation and online. This chapter explains, in order, enabling, binding, how each widget interacts, text / trends / alarms / page switching, simulation, and uploading resources and the language buttons.

Task	Where	This chapter
Draw pages, place widgets, set the appearance and languages	86HMI Editor (86Duino IDE "Tools > 86HMI", or **86HMI Editor** on the 86HMI page)	13.1, 13.6
Widget ↔ variable binding	The 86HMI page of 86LDmicro, "86HMI" in the project tree, right-click on a ladder element	13.2 to 13.4
Try the operation on the computer	86HMI simulator window (opens automatically when simulating with F5)	13.5
Send the screen to the controller	Upload (F8), or upload only the resources	13.6

13.1 Enabling 86HMI

The 86HMI screen is stored in the sketch's `data\_project.hpj`. When you press **Code Generation** in the 86HMI Editor, it writes `myhmi.cpp` / `myhmi.h` and the pictures and fonts used by the screen (`data\Resource`), and adds `#include "myhmi.h"` and `Hmi.begin();` to the `.ino`. 86LDmicro then reads this `.hpj` and lists all the pages and widgets.

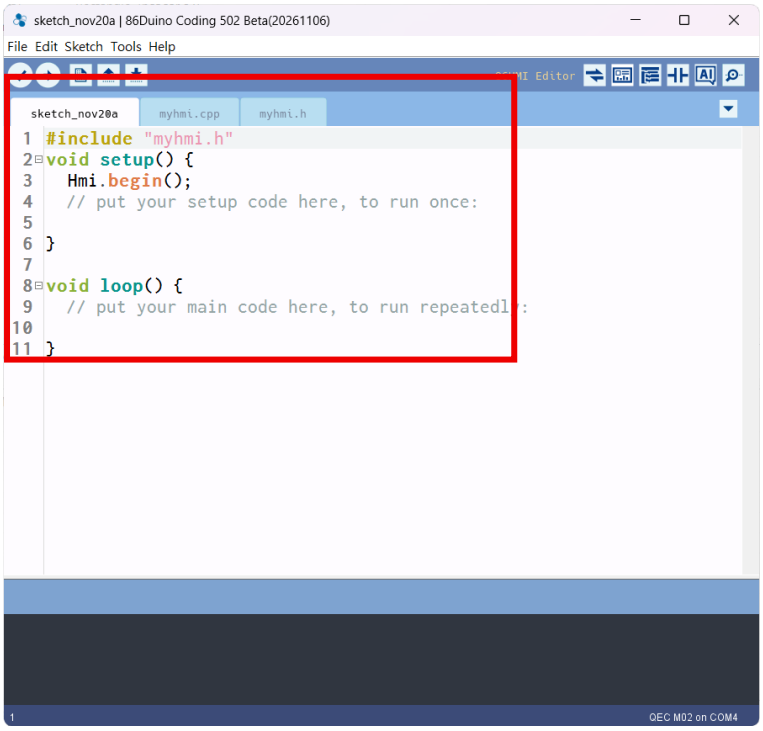


Figure 13-0 86HMI Code Generation

13.1.1 Opening the 86HMI Editor from the 86Duino IDE

After opening the sketch in the 86Duino IDE, click "**Tools > 86HMI**" to open the 86HMI Editor. This is the same as in the 1.1 days.

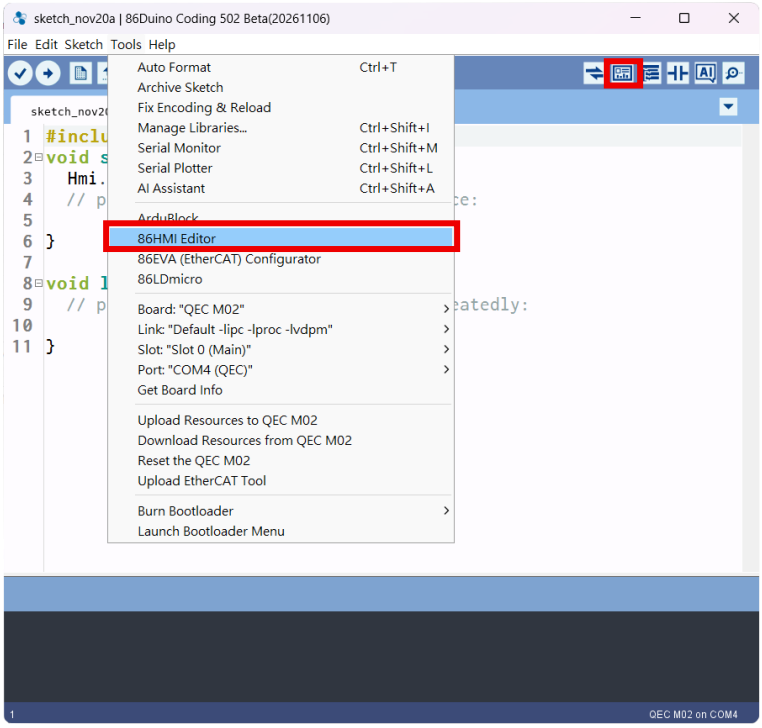


Figure 13-1 The "Tools" menu of the 86Duino IDE: 86HMI and 86LDmicro

13.1.2 Opening the 86HMI Editor from 86LDmicro (recommended)

1. Double-click "**86HMI**" in the project tree, or use "**Project > 86HMI screen and bindings**", to open the 86HMI page.

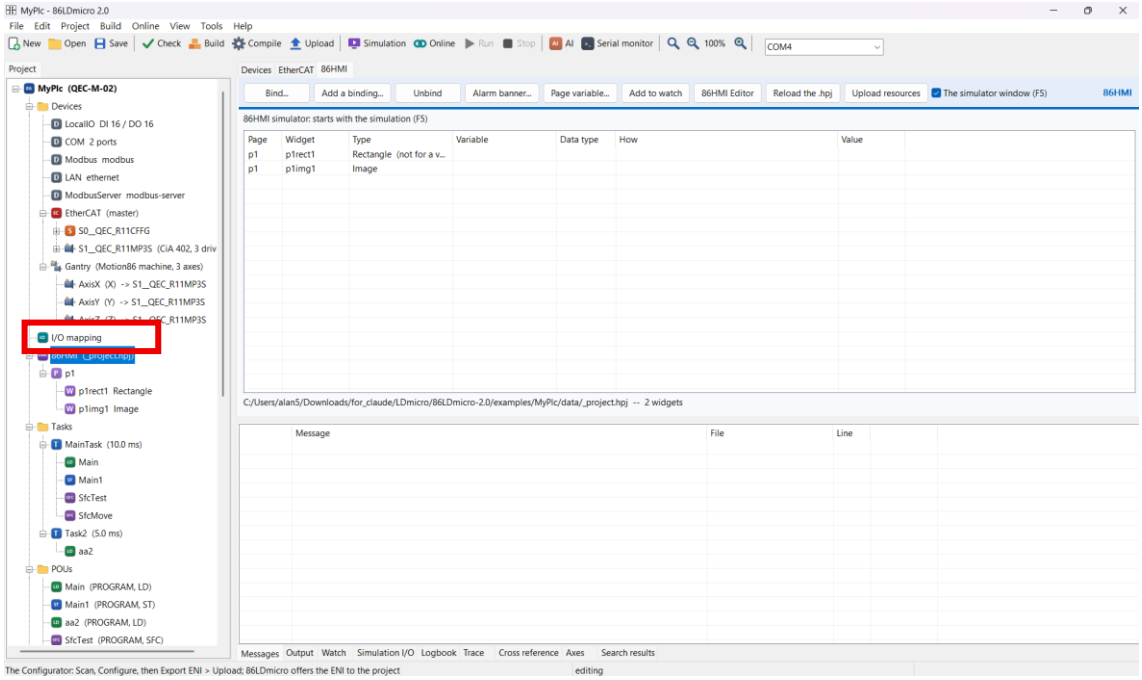


Figure 13-2-1 86LDmicro: Project > 86HMI screen and bindings

2. Press **86HMI Editor**: this opens the editor in the same way as "Tools > 86HMI" in the 86Duino IDE, with the `data\_project.hpj` of this project's sketch; if the sketch has no screen yet, a new screen is created.

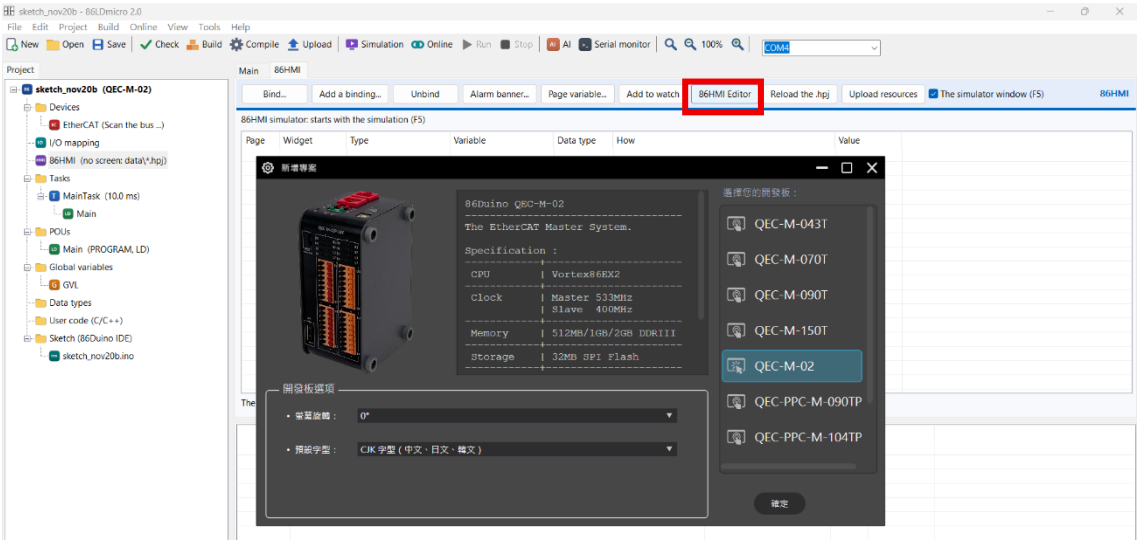


Figure 13-2-2 86LDmicro and the 86HMI Editor, opening a new project

3. In the editor, add pages, place widgets (Button, Switch, LED, Number, Label, Image ...), and set their names (for example `p1b1`, `p1led1`) and appearance.

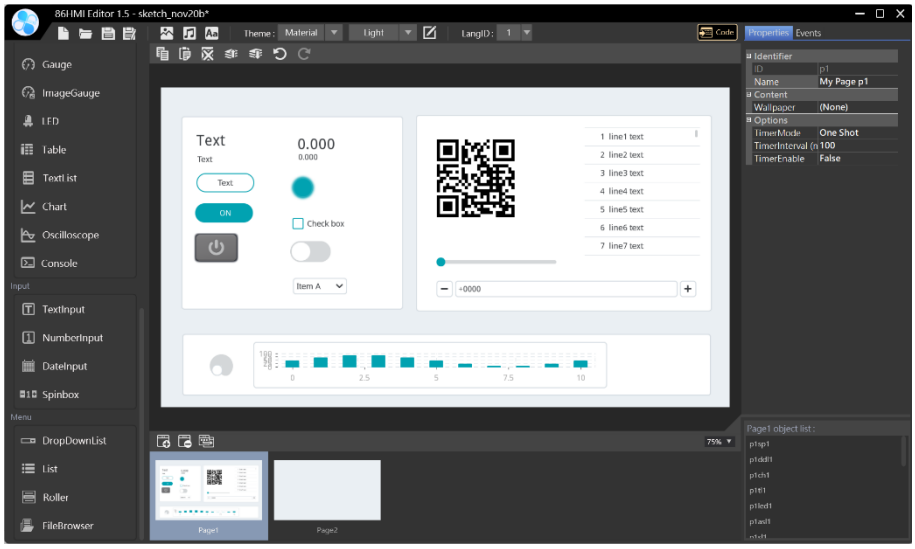


Figure 13-2-3 86HMI Editor

4. Press **Code Generation**: this saves `data\_project.hpj`, `myhmi.cpp`, `myhmi.h` and `data\Resource`, and patches the `.ino`.

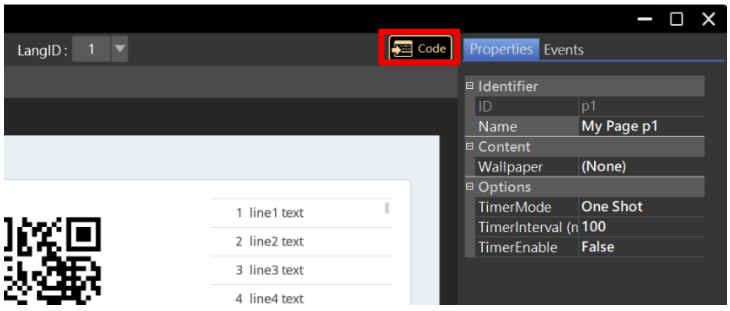


Figure 13-2-4 86HMI Editor; Press Code Generation

5. Back in 86LDmicro: the 86HMI page and the project tree reload automatically (the status bar shows "86HMI: the screen is saved in the sketch (data_project.hpj, myhmi.cpp / myhmi.h): build to use it").

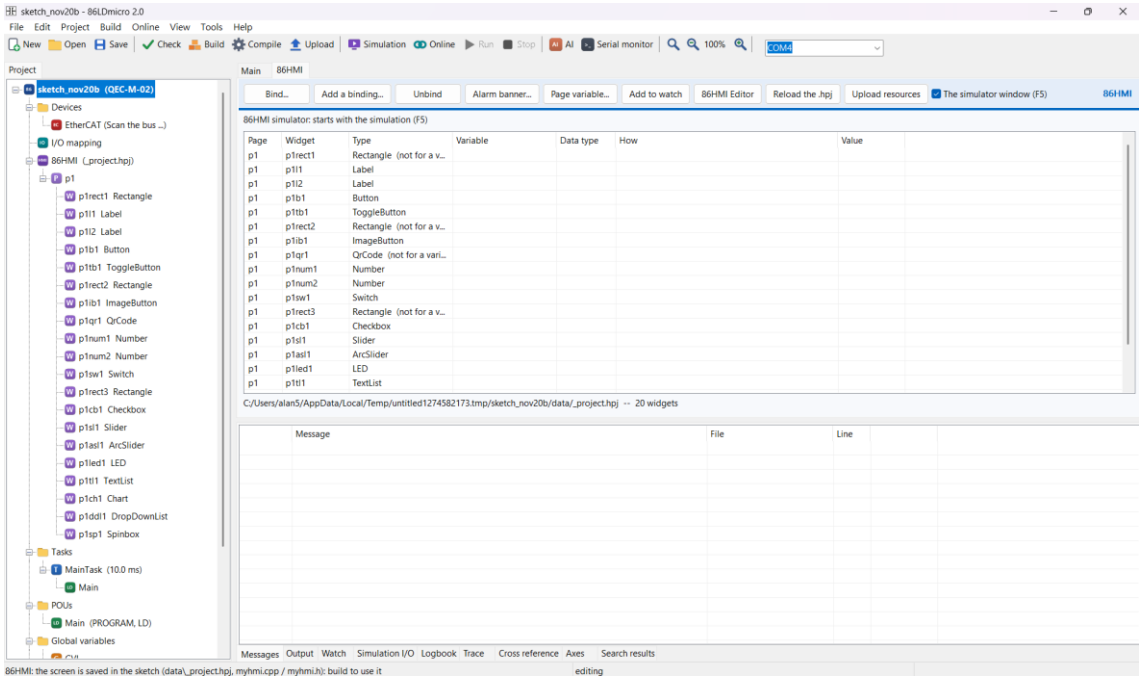


Figure 13-2-5 Back in 86LDmicro: the 86HMI page and the project tree reload automatically

If it does not update, press **Reload the .hpj**.

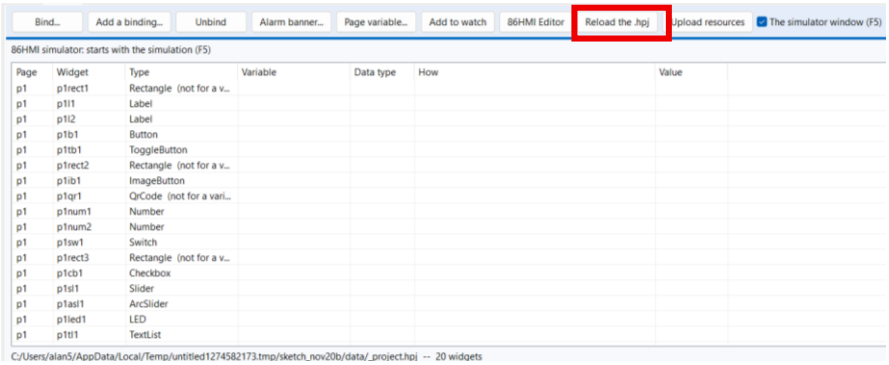


Figure 13-2-6 Back in 86LDmicro: if the 86HMI page does not update, press Reload the .hpj

***Tip:** When 86LDmicro starts the 86HMI Editor, it uses the editor that comes with the 86Duino IDE, so the IDE folder must be set correctly (section 2.6, "Options"). When 86LDmicro is started from the 86Duino IDE, it uses that IDE automatically.

13.1.3 The 86HMI page

The 86HMI page lists every widget in the `.hpj`; bound widgets have **one row per binding**. The buttons at the top and the table columns are as follows:

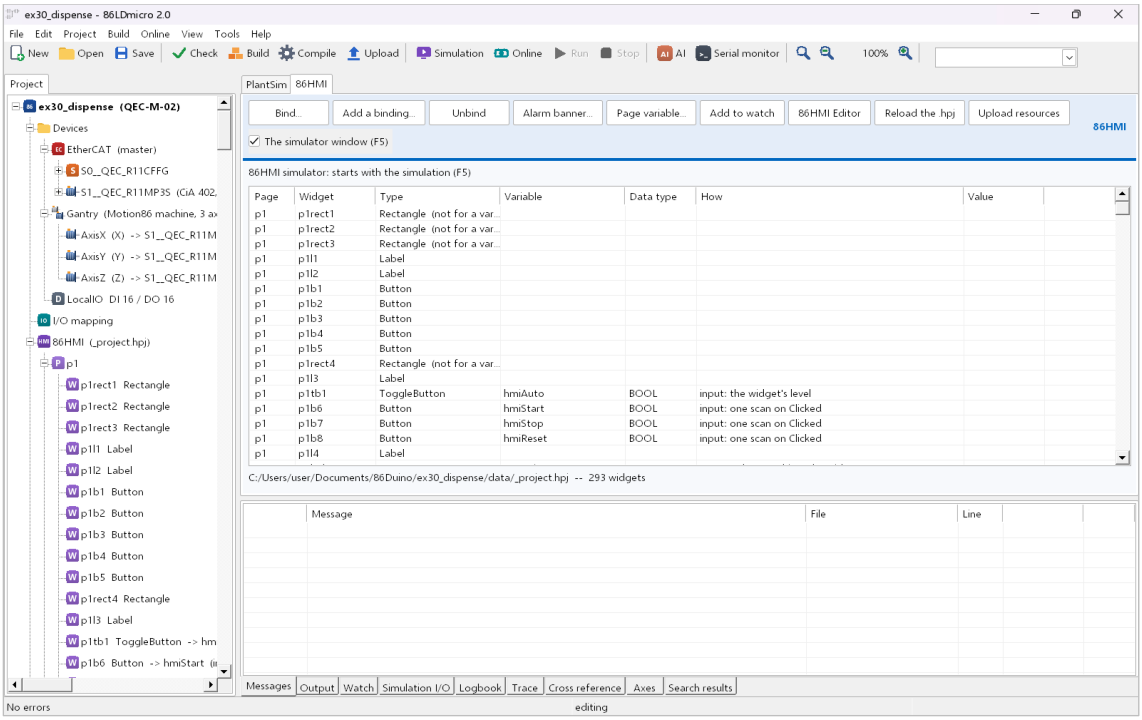


Figure 13-3 The 86HMI page: the button bar and the widget / binding list

Button	Use
Bind...	Changes the binding of the selected row; adds one when the widget is not bound yet (section 13.2)
Add a binding...	Adds another binding to the same widget (section 13.2.3)
Unbind	Removes only the binding of the selected row
Alarm banner...	An alarm banner controlled by BOOL variables (section 13.4.3)
Page variable...	An integer variable decides the page currently shown (section 13.4.4)
Add to watch	Adds the variable of the selected row to the "Watch" window (section 9.3)
86HMI Editor	Opens this project's `data\_project.hpj` (section 13.1.2)
Reload the .hpj	Reloads the pages and widgets after the screen has been changed outside
Upload resources	Uploads only `data\Resource` (section 13.6)
The simulator window (F5) (check box)	When checked, simulation (F5) opens the 86HMI simulator window automatically (section 13.5)

Column	Contents
Page	The page the widget is on (p1, p2 ...)
Widget	The widget's name in the 86HMI Editor (for example p1b1)
Type	The widget type (Button, Switch, LED, Number, Image (x) ...); widgets that cannot be bound to a variable show "(not for a variable)"
Variable	The bound global variable
Data type	The variable's type (BOOL, INT, REAL ...)
How	Input or output and how (section 13.3)
Value	The live value of the variable in simulation or online

The bottom of the page shows the path of the `.hpj` and the number of widgets. When the sketch has no screen, the project tree shows "86HMI (no screen: data*.hpj)"; first create a screen as described in section 13.1.2.

13.2 Binding widgets

Binding specifies which global variable a widget is linked to and in which direction the data flows: an **input** sets the variable from the screen (buttons, switches, entering numbers), an **output** shows the variable on the screen (lamps, numbers, text, picture positions). Bindings are stored in `bindings` in `project.86ld`; the program uploaded to the controller and the simulator window use the same settings.

13.2.1 Ways to open the binding form

- On the 86HMI page, select a widget (or one of its binding rows) and press **Bind...**, or double-click the row.
- Double-click "86HMI > page > widget" in the project tree.
- Starting from the variable: right-click the variable on a ladder element, a watch row or in ST > "**86HMI binding of <variable>...**", then choose a widget, the alarm banner or the page shown (section 13.4.5).

13.2.2 The binding form (by widget type)

The form's title is "86HMI Binding", and its first line tells you which widget it is (name, type, page). The fields depend on the widget type:

Field	Description
Variable	A global variable. Choose it from the drop-down list or type a new name: a new name is added to the GVL automatically (BOOL for buttons / LEDs, INT for numbers)
How	Input (the screen sets the variable) or output (the screen shows the variable); the choices depend on the widget type (section 13.3). An event is TRUE for one scan only
Scale	Number widgets: variable = widget value × scale (÷ scale for an output); 1 means no conversion
Text (Label / Table / TextList)	A template (`Temperature %d C`, where %d stands for the value), or one message per value, one per line: `0=Stopped`, `1=Running`, `2=Alarm` (section 13.4.1)
Column, row / line (Table / TextList)	The cell (Table) or line (TextList) to write to
Channel (1..4), period (ms) (Chart / Oscilloscope)	Trend: which channel the variable is sampled into and how many ms between samples (section 13.4.2)
Enable (a BOOL, optional)	Text and trend widgets: blank = always update; when a BOOL is given, the widget is updated only while it is TRUE

Button: the form only has "Variable", "How" and "Scale". When adding, "Variable" is blank and "How" defaults to "Input: the variable follows the widget's level".

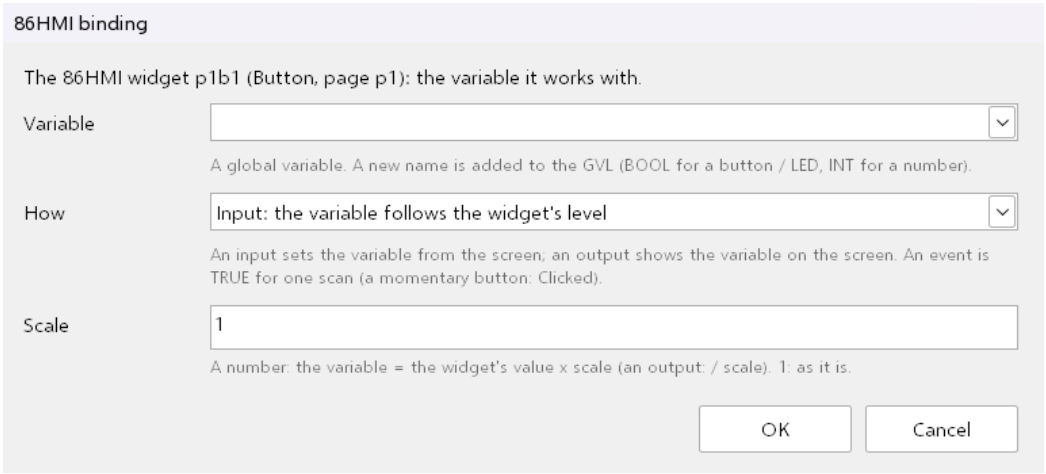


Figure 13-4 Binding a Button: Variable, How, Scale

Switch: stays in place after switching ON / OFF, usually bound to an internal flag (for example `Rstart`).

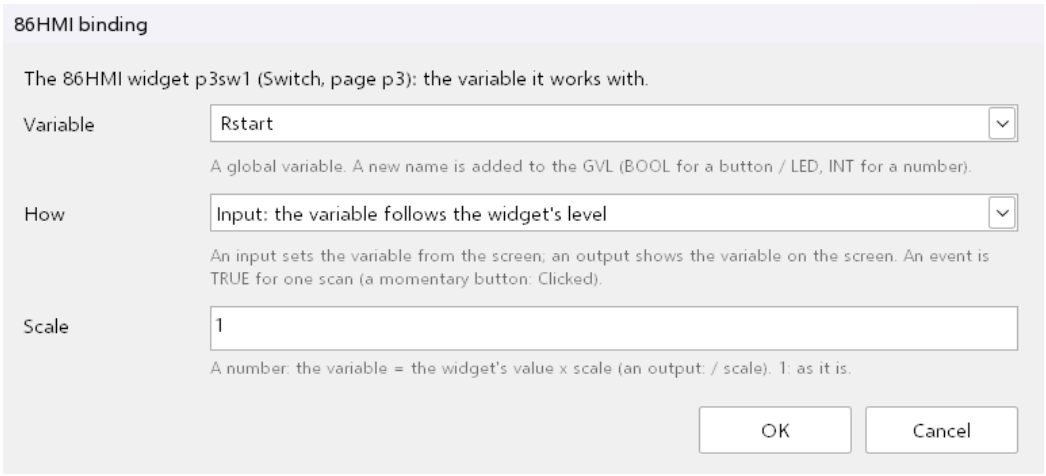


Figure 13-5 Binding a Switch: p1sw1 → Rstart, Input: the variable follows the widget's level

Number: output only; shows the variable's value on the screen; the scale can convert units.

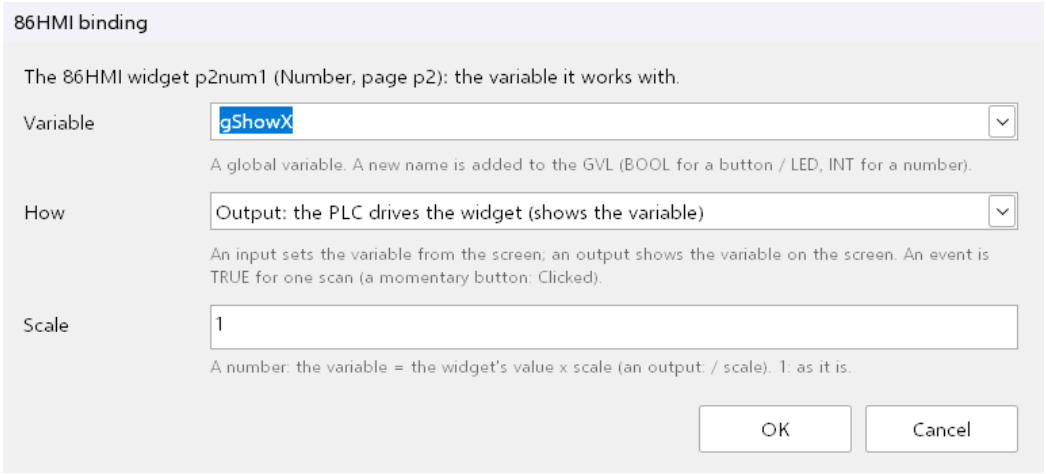


Figure 13-6 Binding a Number: output, the PLC drives the widget (shows the variable)

13.2.3 Illustrated: binding button p1b1 to Rstop

1. On the 86HMI page, select widget **p1b1** (Button) and press **Bind...**.
2. In "Variable", type or choose `Rstop`; in "How", choose **Input: the variable follows the widget's level** (TRUE while pressed).
3. Press **OK**. The "Variable" column of p1b1 in the table shows Rstop, and it also shows under 86HMI in the project tree (for example `p1b1 Button -> Rstop`).
4. In the ladder diagram, use `Rstop` together with the physical stop button (see the note in section 13.3).

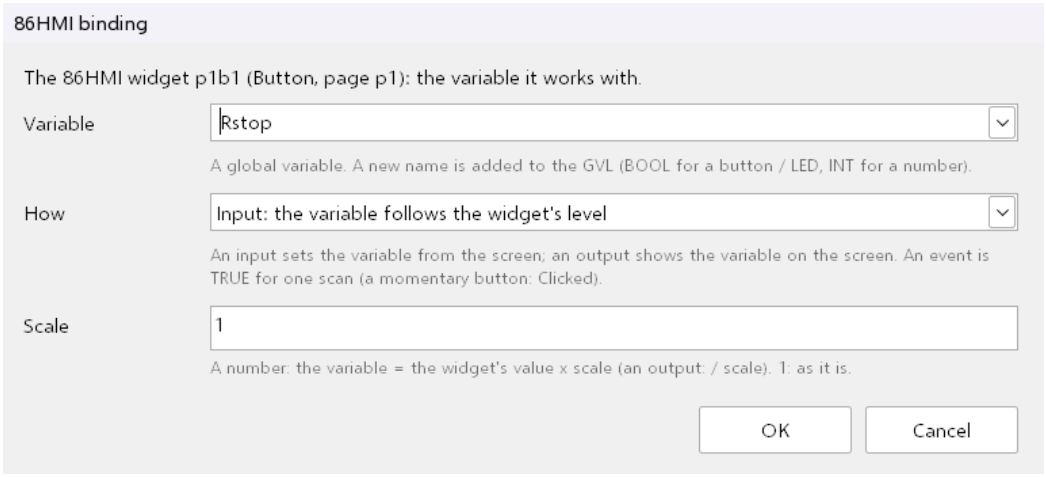


Figure 13-7 The binding form: p1b1 → Rstop, Input: the variable follows the widget's level

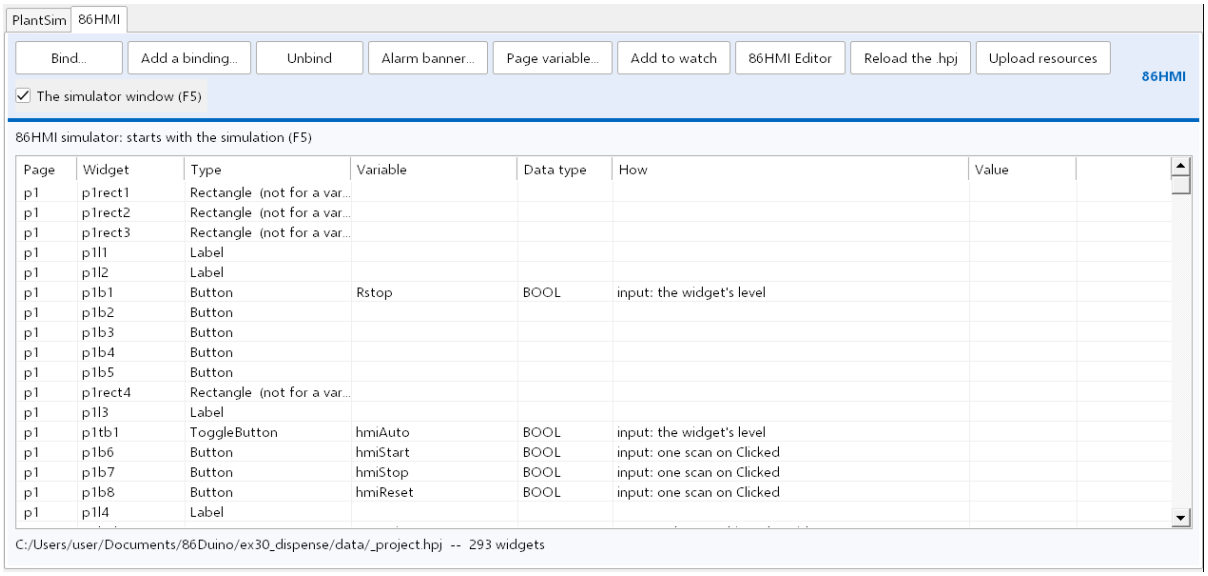


Figure 13-8 Binding done: the table and the project tree both show p1b1 → Rstop, p1led1 → Yout, p1sw1 → Rstart, p2num1 → Mposition

13.2.4 Several bindings on one widget

One widget can be bound to several variables at the same time, for example an Image with "show / hide", "x position" and "y position"; each channel of a Chart; each cell of a Table; plus an "Enable" on any widget. The 86HMI page has **one row per binding**, so the same widget appears in several rows.

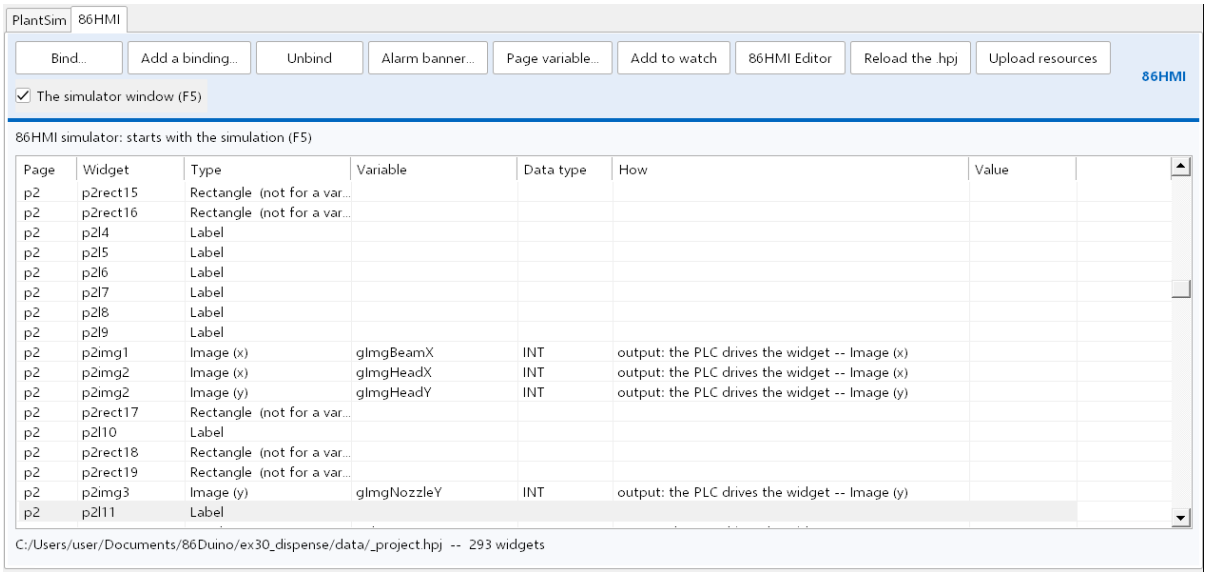


Figure 13-9 The 86HMI page of ex30: p2img1 and p2img2 each have two bindings, Image (x) and Image (y), one row per binding

1. Select any row of the widget and press **Add a binding...**.
2. Choose "Variable" and "How". The top of the form tells you how many other bindings the widget already has ("It has N other binding(s): a new one is added next to them").
3. Press **OK**; the list gets one more row, and a new variable is added to the GVL automatically.
4. To change a binding, select its row and press **Bind...** (or double-click). To remove one, select its row and press **Unbind**; only that one is removed.

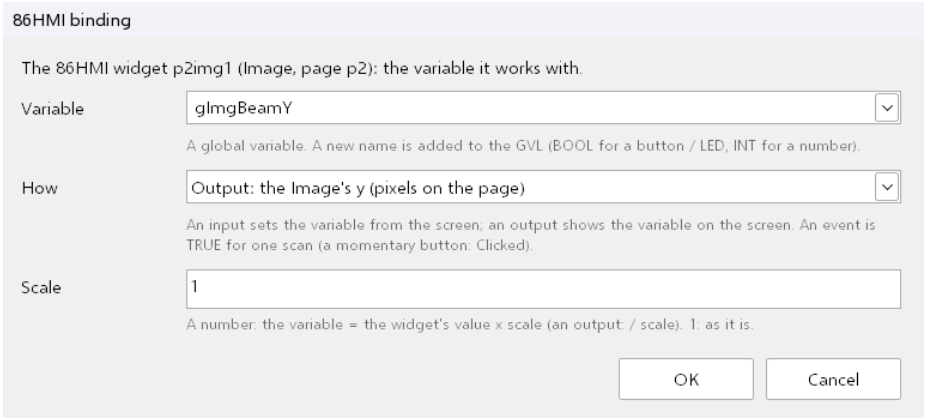


Figure 13-10 Adding an "Output: the Image's y (pixels on the page)" binding to p2img1 (variable glmgBeamY)

To avoid conflicts, the following cases are blocked with a message:

- The same widget repeats the same usage (for example two x positions, the same channel of a Chart, the same cell of a Table).
- The same value is both an input and an output (for example a Switch bound as an input and also bound to an output).
- A variable is already bound to another widget: unbind it on that widget first (except the BOOL used for enabling, which can be bound to several widgets).

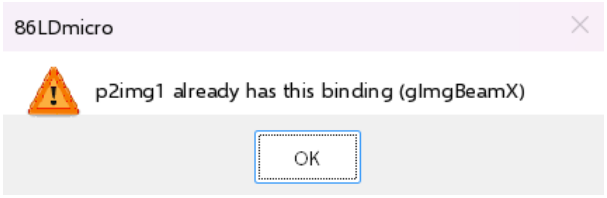


Figure 13-11 Adding another x position binding is blocked when p2img1 already has one

***Tip:** In the project tree, all of a widget's bindings are listed after it; double-clicking a widget with several bindings lets you choose which one to change, or "(a new binding)"; "Unbind" in the project tree asks whether to remove all of them.

13.3 Widgets and how they interact

The "How" list shows only the ways that the widget supports. The table below summarizes the common widgets (the widget names are the types in the 86HMI Editor):

Widget	Available "How" choices	Interaction
Button, ImageButton	Input: the variable follows the widget's level Input: one scan TRUE on Clicked (event) Input: one scan TRUE on Pressing (event)	Follow the level: TRUE while pressed, FALSE when released (like a physical button; ImageButton has only events) Event: the variable is TRUE for **one scan** when clicked; suited to momentary commands such as "start" and "reset"
ToggleButton, ImageToggleButton, Checkbox	Input: the variable follows the widget's level Input: the level, inverted Input: event (Checked / Unchecked) Output: the PLC drives the widget	Stays in place after switching; the PLC can also set the state
Switch	Input: the variable follows the widget's level Input: the level, inverted (Off) Input: event (On / Off) Output: the PLC drives the widget	Stays in place after switching ON / OFF (for example p1sw1 → Rstart)
LED	Output: the PLC drives the widget (shows the variable)	Lights when the variable is TRUE (for example p1led1 → Yout)
Number	Output: the PLC drives the widget	Shows the variable's value; the scale can convert it
NumberInput, Spinbox	Input: the variable follows the widget's level Input: event (Applied / Canceled, NumberInput only) Output: the PLC drives the widget	Can have one output and one input at the same time (two variables): when the PLC changes the output variable, it is written to the screen (for example loading a recipe value), and the value the operator enters is read back through the input variable; the output is written only when its value changes, so it does not overwrite a value being entered
Slider, ArcSlider, ImageSlider	Input: the variable follows the widget's level, event (Dropped / Dragging) Output: the PLC drives the widget	Drag to set the value
Bar, ArcBar, Gauge, ImageBar, ImageGauge	Output: the PLC drives the widget Input: the variable follows the widget's level	Shows the value as a bar, an arc or a needle (usually used as an output)
DropDownList, List, Roller	Input: the variable follows the widget's level (the selected item, an integer), event (Selected)	The operator chooses an item
Image	Output: the PLC drives the widget (show / hide) Output: the Image's x (pixels on the page) Output: the Image's y (pixels on the page)	Show: shown when the variable is TRUE, hidden when FALSE x / y: the variable (INT) is the picture's pixel position on the page; when the PLC changes the variable, the picture moves (for example the position page of ex30)
Label, Table, TextList	Output: the value shown as text	A template or one message per value (section 13.4.1)

Widget	Available "How" choices	Interaction
Chart, Oscilloscope	Output: a trend (the variable sampled into a channel)	One binding per channel (section 13.4.2)
Any widget (including Label, Table, Chart)	Output: the widget enabled (TRUE) or greyed out (FALSE)	When the BOOL is FALSE the widget is greyed out and cannot be operated, so the PLC decides which buttons can be pressed at the moment; the same BOOL can be bound to several widgets; written only when the value changes

13.3.1 The three kinds of input

How	Value of the variable	Suited to
Input: the variable follows the widget's level	The same as the widget's state (TRUE while pressed, FALSE when released; switch ON = TRUE)	Jogging, "move only while held", switch-type selections
Input: the level, inverted	The opposite of the widget's state	Normally closed buttons (pressed = FALSE)
Input: one scan TRUE on Clicked (event)	TRUE for one scan when the event occurs, then back to FALSE automatically	Momentary commands such as start, reset and alarm acknowledge; no need to make a rising edge yourself

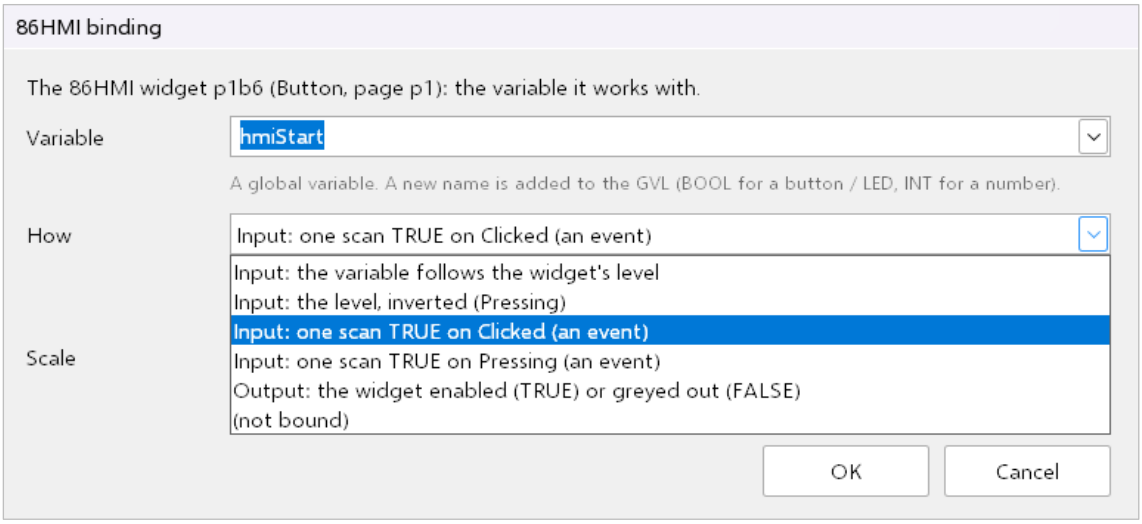


Figure 13-12 The "How" list of a Button: follow the level, Clicked event, Pressing event

*__Tip:__ An event (TRUE for one scan) is cleared after **the task that reads it** has scanned once, so it can also be read in a slower task (for example a 100 ms SlowTask). When the same event is read by two tasks, only the first task is guaranteed, and the build gives a warning: read it in one task only, and latch it with SET when necessary.

*__Note:__ A variable already mapped to a physical input cannot be used as an HMI input (the next scan overwrites it with the physical input). Bind an internal variable (for example `Rstart`, `Rstop`) and use it together with the physical button in the program.

Example: the physical start `Xstart` (local X00) in parallel with the HMI switch `Rstart`, and the physical stop `Xstop` in series with the HMI stop button `Rstop`:

```
NETWORK 'start/stop'
    [ NO(Xstart) | NO(Rstart) | NO(Rrun) ] NC(Xstop) NC(Rstop) OUT(Rrun)
NETWORK 'lamp'
    NO(Rrun) OUT(Yrun)
```

Check in simulation: when `Rstart` is TRUE, `Rrun` and `Yrun` become TRUE and stay TRUE; when `Rstop` is TRUE, `Rrun` becomes FALSE.

13.3.2 Enable: letting the PLC decide which buttons can be pressed

Any widget can have an "**Output: the widget enabled (TRUE) or greyed out (FALSE)**" binding. For example, grey out the "Home" button while running to avoid pressing it by mistake:

1. Select the widget and press **Add a binding...** (when the widget already has other bindings) or **Bind...**.
2. In "Variable", enter a BOOL (for example `gCanHome`), in "How" choose **Output: the widget enabled (TRUE) or greyed out (FALSE)**, and press OK.
3. In the program, write `gCanHome := NOT gRunning;` (or the equivalent ladder diagram). The same BOOL can be bound to several widgets.

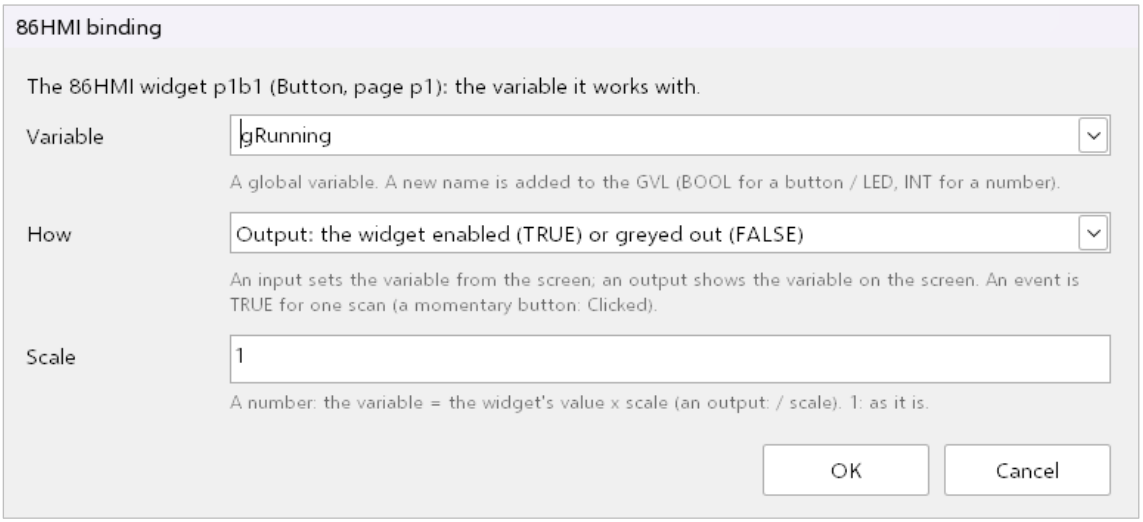


Figure 13-13 Enable binding: the widget is greyed out when the BOOL is FALSE

13.4 Text, trends, alarms and page switching

13.4.1 Text: Label, Table, TextList

The "How" of text widgets is "**Output: the value shown as text**". The "Text" field can be written in two ways:

Format	Example	Shown on the screen
Template: `%d` stands for the value	`Temperature %d C`	Temperature 25 C

One message per value, one per line	`0=Stopped`
	`1=Running`
	`2=Alarm`
		Shows "Running" when the variable is 1

- **Table**: also fill in "Column, row" to choose the cell; one binding per cell.
- **TextList**: also fill in "line". Real controller only: the 86HMI simulator has no API for TextList, so it is not updated in simulation.
- "Enable (a BOOL, optional)": blank = always update; when a BOOL is given, the text is updated only while it is TRUE.

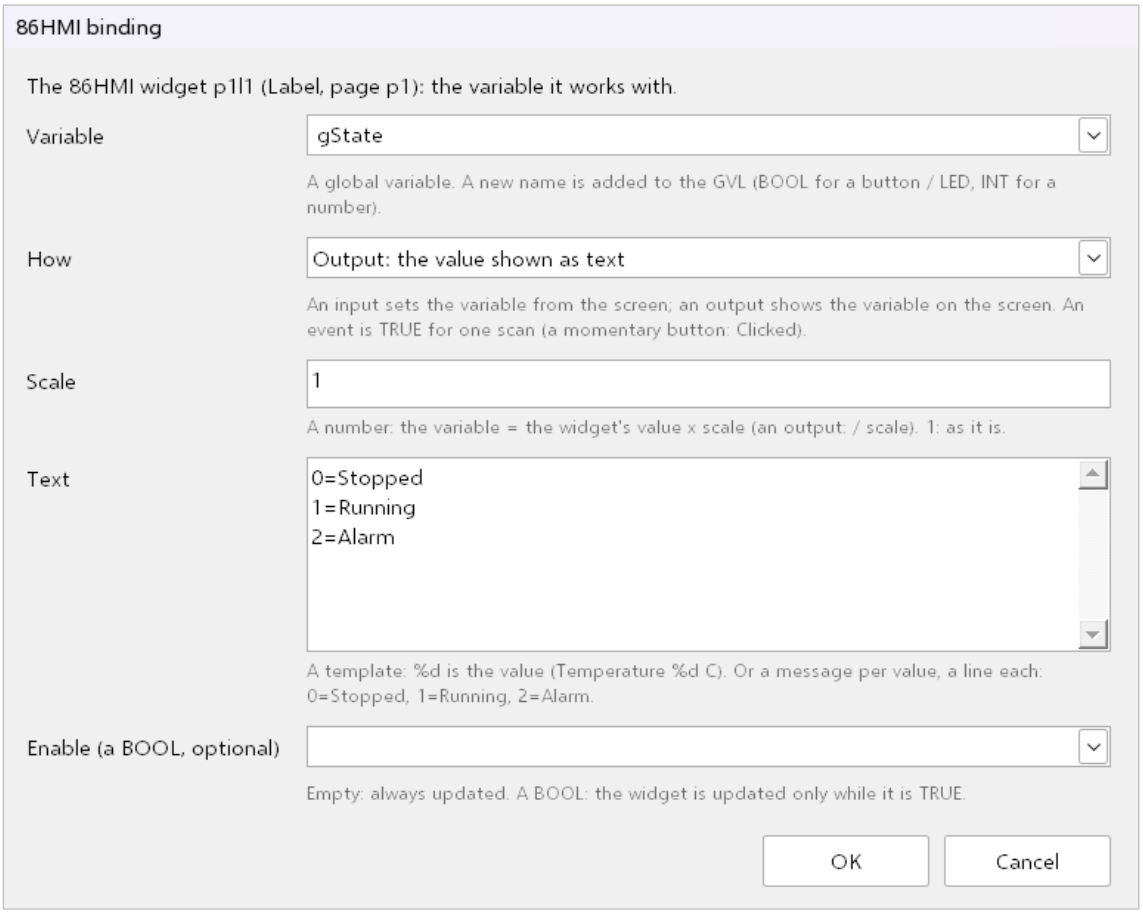


Figure 13-14 Text binding of a Label: one message per value

13.4.2 Trends: Chart, Oscilloscope

"How" is "**Output: a trend (the variable sampled into a channel)**"; in "Channel (1..4), period (ms)", for example `1, 100` means the variable is sampled into channel 1 every 100 ms. A Chart has at most 4 channels, one binding per channel (use **Add a binding...** to add a second channel). The top of the form shows the number of points and channels of this Chart.

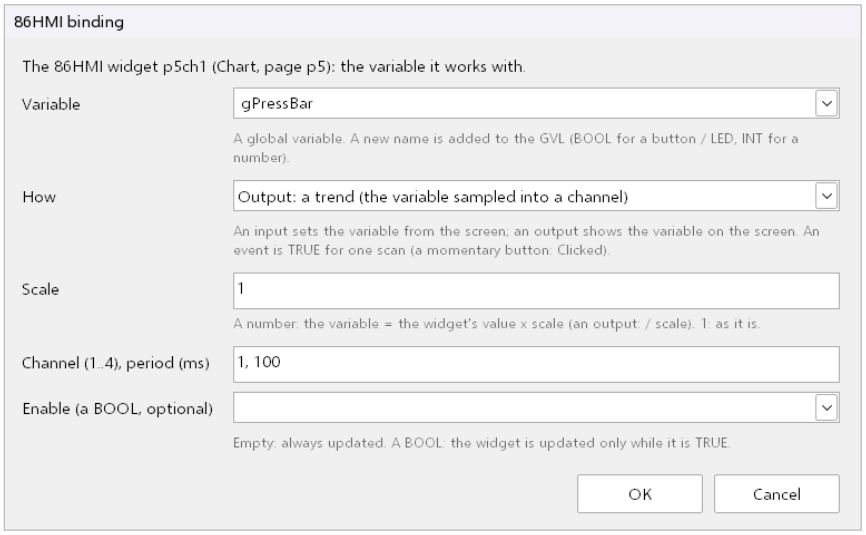


Figure 13-15 Trend binding of a Chart: channel and sampling period

13.4.3 Alarm banner

Press **Alarm banner...** on the 86HMI page (or right-click "86HMI" in the project tree): a banner is shown while a BOOL variable is TRUE and removed when it is FALSE.

Field	Description
Variable (a BOOL; empty: remove)	The variable that controls the banner; a new name is added to the GVL as a BOOL
Bank (0..15)	The banner's slot: banners of different banks show together
Text	The message on the banner, for example "Emergency stop pressed"
Enable (a BOOL, optional)	Blank = always active

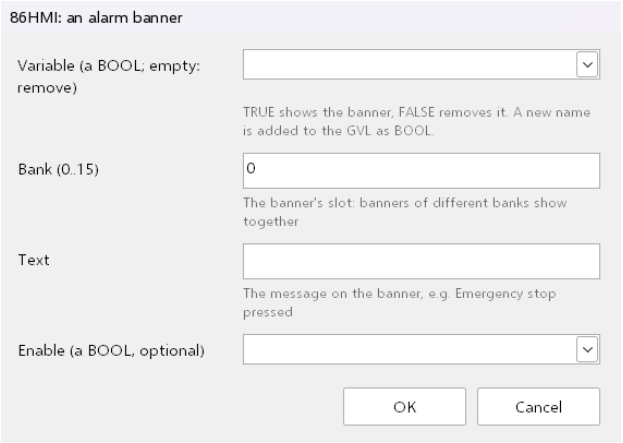


Figure 13-16 "86HMI: an alarm banner": Variable, Bank, Text

13.4.4 Page switching: the page variable

Press **Page variable...**: an integer variable decides the page currently shown (1 = p1, 2 = p2 ...), letting the PLC switch screens on its own (for example switching to the alarm page when an alarm occurs). A new name is added to the GVL as an INT.

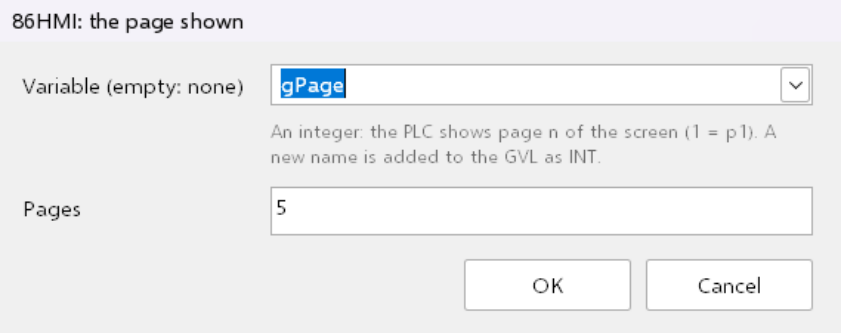


Figure 13-17 "86HMI: the page shown": page variable and number of pages

*Tip: For ordinary page switching, it is better to use the "Page Change" button script of the 86HMI Editor directly (without going through the PLC). The "Page variable" is for switching pages from the PLC; after switching, set the variable back to 0 so that it can switch again next time.

13.4.5 Binding from a variable

Right-click a ladder element (or a watch row, or a variable in ST) > "**86HMI binding of <variable>...**", choose a widget from the list (its current binding is shown after it), "Alarm banner" or "The page shown", press OK and then fill in "How". Choose "(none: remove the 86HMI bindings of this variable)" to remove all of the variable's 86HMI bindings.

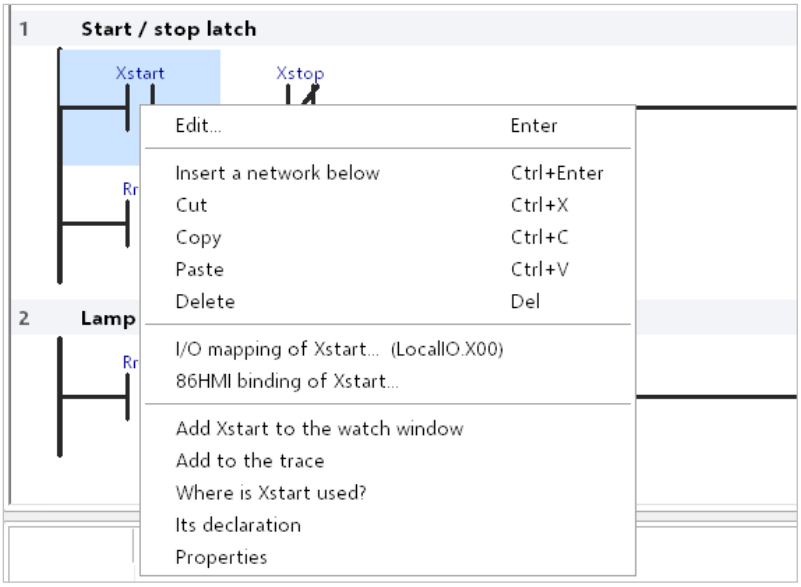


Figure 13-18 The right-click menu of a ladder element: "86HMI binding of Rstop..."

*Tip: The 86HMI screen is updated after the scan of the first task, and is not counted in the scan time or the watchdog (laying out label text and switching pages can take tens of ms).

13.5 Simulation and online

When simulating (F5), if the sketch has an 86HMI screen, the **86HMI simulator window** opens automatically: press buttons and flip switches directly on the computer and the PLC program responds immediately; LEDs, numbers, text, picture positions, alarms, pages and trends show the PLC's values. Online (Ctrl+F5), the simulator window shows the values on the controller.



Figure 13-19 The 86HMI simulator window: operate the screen directly on the computer and the PLC program responds immediately (example: booster station monitoring)

13.5.1 Illustrated: operating 86HMI in simulation

1. Make sure "**The simulator window (F5)**" is checked on the 86HMI page.

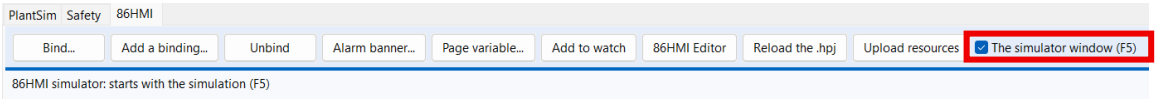


Figure 13-20-1 "The simulator window (F5)" checked

2. Press **Simulate** (F5); the simulator window opens automatically, and the top of the 86HMI page shows the connection state of the simulator window.

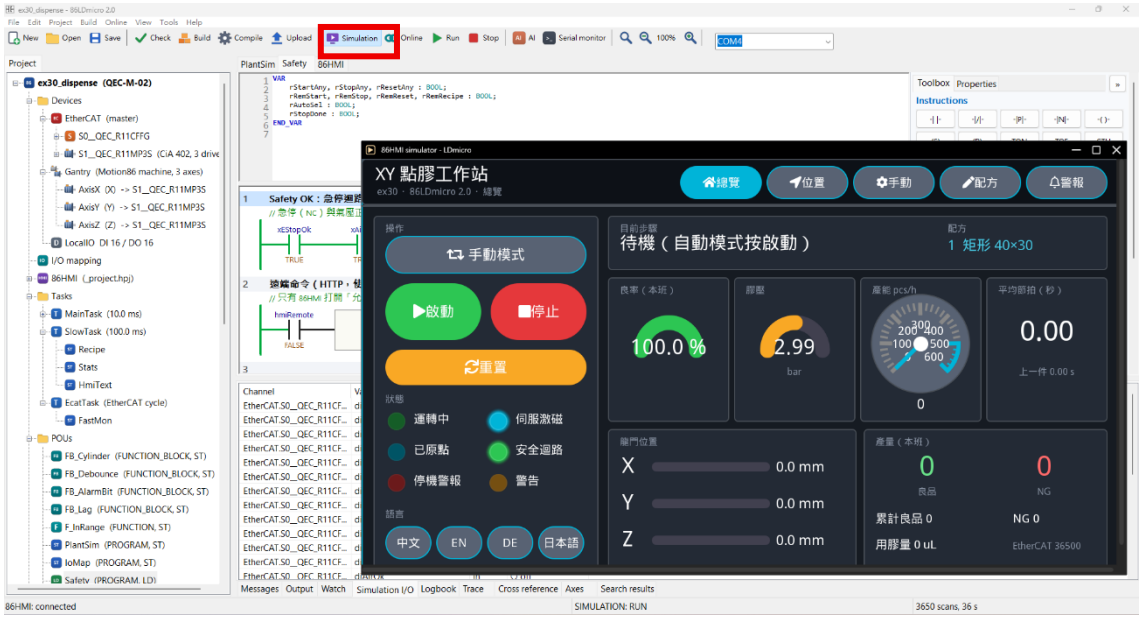


Figure 13-20-2 In simulation: the ladder diagram of 86LDmicro and the 86HMI simulator window side by side

3. Hold down a button on the screen: the bound variable becomes TRUE and the corresponding contact in the ladder diagram turns green; the "Value" column of the 86HMI page updates at the same time.
4. When a PLC output variable changes, the bound LEDs, numbers and pictures update at the same time.
5. Press F5 again to end the simulation; the simulator window closes with it.

To do	How
Close the simulator window (do not open it automatically)	Uncheck "The simulator window (F5)" on the 86HMI page
Reopen the simulator window during simulation	"Online > 86HMI simulator window (the sketch's .hpj)"
Changed a binding	Bindings take effect the next time simulation (F5) starts: end the simulation first and then press F5
See the controller's values online	Connect with Ctrl+F5 and open the simulator window

***Note: known issue** (2.0.0): in 86HMI simulation a ToggleButton switches its state correctly, but the text on the button does not switch between the "not pressed" and "pressed" texts (on the real controller the program generated by 86HMI updates the text). PLC variables are not affected.

13.6 Resource upload and languages

13.6.1 Uploading the 86HMI resources

When the 86HMI Editor runs "Code Generation", the pictures, fonts and screen data used by the screen are written to the sketch's `data\Resource`. These **resources** are stored on the controller, separately from the program (see section 10.6 for uploading in general).

Situation	What to do
First upload, or both the program and the screen changed	Press **Upload** (F8): the program and `data\Resource` are uploaded together
Only pictures, fonts or the screen's appearance changed; the PLC program did not change	Press **Upload resources**: sends only the resources, without compiling; the program on the controller stays the same (faster than F8)

1. Check the COM port on the toolbar (for example COM4).
2. On the 86HMI page press **Upload resources**, or use "**Build > Upload the 86HMI resources (data\Resource)**".

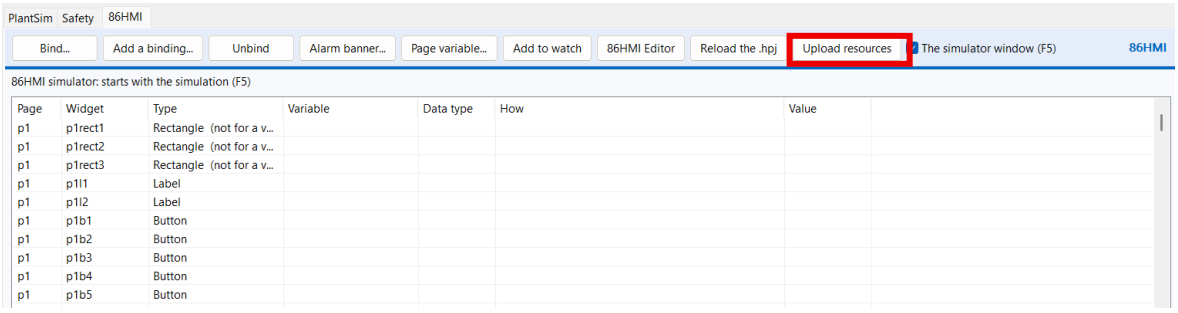


Figure 13-21-1 The Upload resources button for the 86HMI resources

3. The "Output" window shows "== Uploading the resources to COM4: …\data\Resource ..." and the progress, and finally "Resources uploaded".
4. If you were online, it goes offline automatically before the upload and reconnects automatically when it is done.

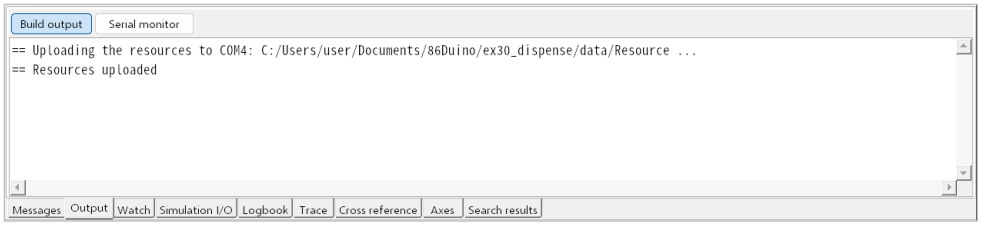


Figure 13-21-2 Uploading the 86HMI resources: the progress in the "Output" window and "Resources uploaded"

***Tip:** This function is the same as "Tools > Upload Resources to QEC M02" in the 86Duino IDE. Command line: `86ldmicro-cli upload-res <project> <COM>`.
If "Nothing to upload: … does not exist or is empty" appears, the sketch has no `data\Resource` or it is empty: first open the .hpj in the 86HMI Editor and press "Code Generation".

13.6.2 Language switch buttons

An 86HMI screen can have several languages (set in the 86HMI Editor). To let the operator switch the language with a button on the controller, **you must write a switch script for the button in the 86HMI Editor**; the bindings of 86LDmicro handle only variables and do not switch the language for you.

1. In the 86HMI Editor, select the language button (for example "Chinese") and add the script `Hmi.setLanguage(1);` to its event (Clicked) (the number is the language number, following the order of the languages in the editor: 1, 2, 3 ...).
2. Set the other language buttons in turn to `Hmi.setLanguage(2);`, `Hmi.setLanguage(3);` ...
3. Press **Code Generation**, go back to 86LDmicro and press **Upload** (F8).



Figure 13-22 86HMI Editor: the Clicked script Hmi.setLanguage(1); of a language button

> ***Note:** When the script is left empty, the simulator window may look fine, but after uploading to the controller the button does not switch the language. The ex30 example already uses the correct script.

Chapter 14 C/C++ and 86Duino integration

The PLC program of 86LDmicro finally becomes a C/C++ program for 86Duino (`plc86.cpp`), compiled by the compiler of the 86Duino IDE. So you can add your own C/C++ to the same project: serial protocols, networking, special algorithms and 86Duino / Arduino libraries can all share the same variables with LD, ST and SFC.

Where to write it	Suited to	This chapter
User code files `usercode/*.cpp86`	Code closely tied to PLC variables: communication, conversions, synchronized control in every EtherCAT cycle	14.1 to 14.4
The sketch's `.ino` and other `.cpp` / `.h` files	Code unrelated to the PLC, or existing Arduino code	14.5

14.1 User code files

Use "**Project > Add a user code file (C/C++)...**" (or right-click "User code (C/C++)" in the project tree) and enter a name (default `MyCode`) to create `usercode/<name>.cpp86`. At build time it is compiled directly into `plc86.cpp`, and it can use all the APIs and libraries of 86Duino / Arduino (write `#include <...>` as usual).

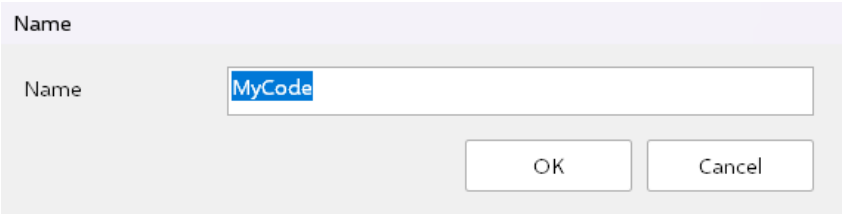


Figure 14-1 Adding a user code file: enter a name

The new file comes with a template: comments explain how to access PLC variables and list each hook function (section 14.3). The template picks the first BOOL variables in the GVL starting with `X…` and `Y…` as example names; when the project has no EtherCAT devices, `plc86_user_ethercat()` is listed as a comment:

```
// MyCode: C/C++ of the project, compiled into plc86.cpp with the PLC (Help > C/C++ user code).
//
// The PLC's variables -- not by their bare names, through these (Ctrl+Space lists them):
//   GVL(Xstart)              a global variable (GVL), read and written: GVL(Ylamp) = true;
//   GVL(table[2]), GVL(rec.x)    an array element (the ST indexes), a structure's field
//   ST(Main.count)               a PROGRAM's variable (ST(Main.t1.Q): a function block's output)
//   FUN(Scale)(x, 10)            an ST FUNCTION of the project
//   LD_READ(x), LD_WRITE(y, v)   as in LDmicro 1.1
// The hooks: the PLC calls the ones defined here (delete the ones not needed).
// The USB Serial is the upload and online port: print to Serial1..3, not to Serial.

// in setup(), after the PLC's: begin libraries (Serial1, Ethernet, CAN, Modbus ...)
void plc86_user_setup(void) {
}

// every scan of the first task, before its programs (the inputs are read)
void plc86_user_before(void) {
}

// every scan of the first task, after its programs (before the outputs are written)
void plc86_user_after(void) {
    // e.g.: GVL(Ylamp) = GVL(Xstart);
}

// loop(), every pass: non-blocking work only (no delay())
```

```
void plc86_user_loop(void) {
}

// every EtherCAT cycle (the master's cyclic callback, an interrupt): synchronous control.
// Only with an EtherCAT device (Project > Devices). Short: no delay(), no Serial, no malloc.
// void plc86_user_ethercat(void) {
//     if(!ECAT_OK)
//         return;
// }
```

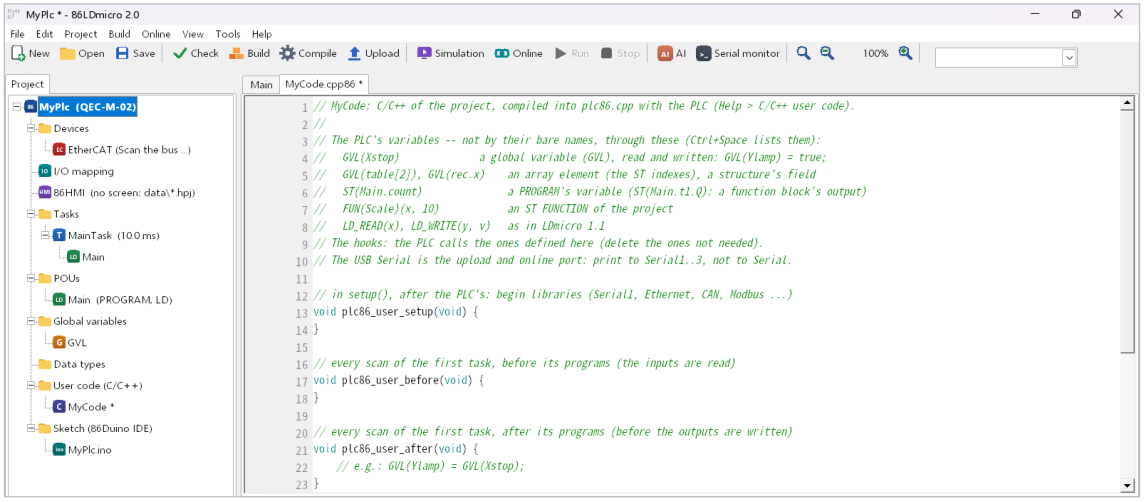


Figure 14-2 The new user code MyCode.cpp86: the template and "User code (C/C++)" in the project tree

The user code editor is the same as the ST editor, with syntax colors, Ctrl+T auto-formatting and error underlines; it also offers completion and parameter hints for all the 86Duino / Arduino APIs (about 7,600 items), and when you pick a library object or class that is not included yet, it adds the `#include` at the top of the file automatically.

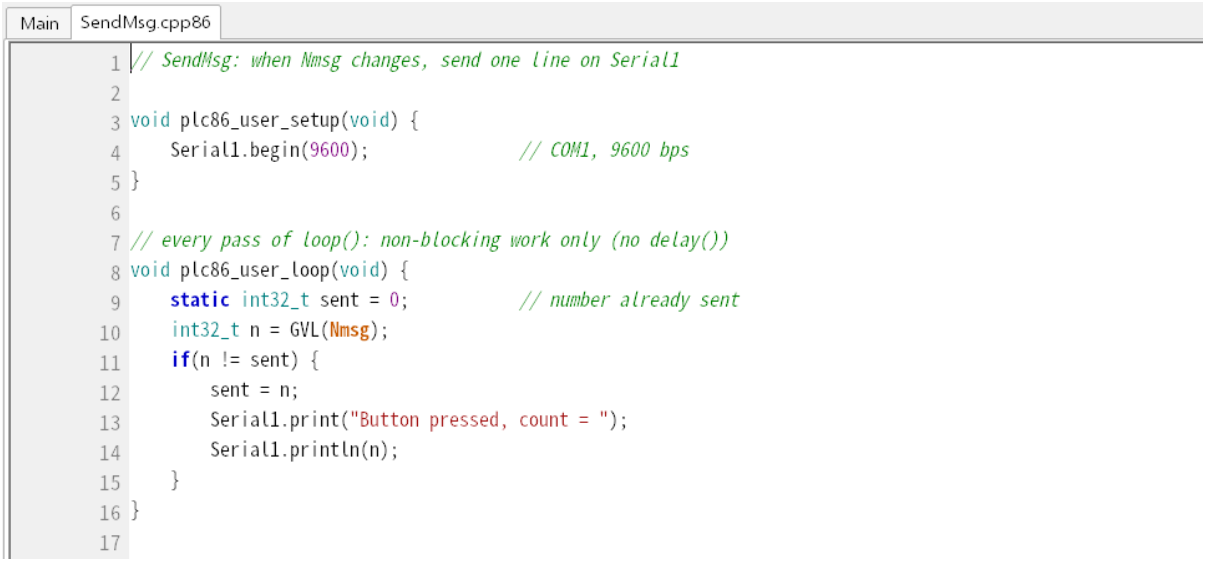


Figure 14-3 The user code editor: hook functions and GVL() accessing PLC variables

14.2 Accessing PLC variables

***Important:** You **cannot write the name of a PLC variable directly** (the actual name in C is something like `ldst_P_GVL__Xstart`); write `GVL(Xstart)`. If you write the name directly, the compile error tells you the correct form.

Form	Meaning	Example
`GVL(name)`	Global variable (read and write)	`if(GVL(Xstart)) GVL(Ylamp) = true;`
`GVL(array[i])`	Global array element (index in the ST range)	`ARRAY[1..5]`, the 3rd element: `GVL(Recipe[3]) = 100;`
`GVL(struct.field)`	Global structure field	`GVL(Axis1.pos) = 12.5f;`
`ST(Program.variable)`	Local variables of a PROGRAM, members of an FB instance, arrays, structures	`ST(Main.tLamp.Q)`, `ST(Filling.step)`
`FUN(name)(arguments)`	Calls an ST FUNCTION of the project	`float v = FUN(Scale)(raw, 0.0f, 100.0f);`
`LD_READ(x)`, `LD_WRITE(y, v)`, `LD(x)`	The LDmicro 1.1 form, same as `GVL()`	`LD_WRITE(Ylamp, LD_READ(Xstart));`
`ECAT_MASTER`, `ECAT_OK`, `ECAT_SLAVE(subdevice)`	The EtherCAT MainDevice object, MainDevice started, the driver object of a SubDevice (when the project has EtherCAT devices)	`ECAT_SLAVE(S0__QEC_R11CFFG).digitalWrite(1, HIGH);`
`AXIS(axis)`, `mc_*`, `M86_MACHINE(machine)`	Motion control (Chapter 12)	`mc_move_abs(AXIS(AxisX), 100.0f, 50.0f);`

Names, fields and indexes are checked at build time, and errors point back to the line number in the `.cpp86` (for example `usercode/MyCode.cpp86:11: error: GVL(Xstpo): there is no such global variable`). Double-click the row in "Messages" to jump to that line.

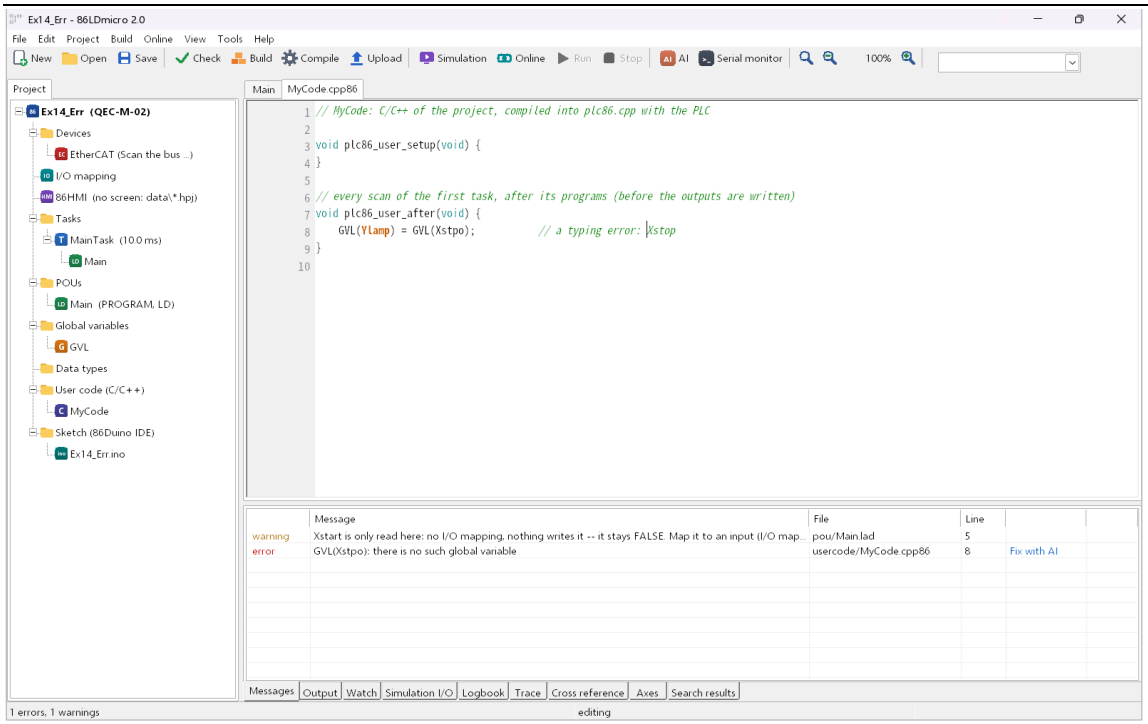


Figure 14-4 A mistyped variable name: "Messages" points to the line number in the .cpp86

When you type `GVL(`, `ST(` or `FUN(`, the available names are listed (with types and comments), and after `(` the parameters of the FUNCTION are shown; hover the mouse over a name to see its type, and in simulation or online also its value.

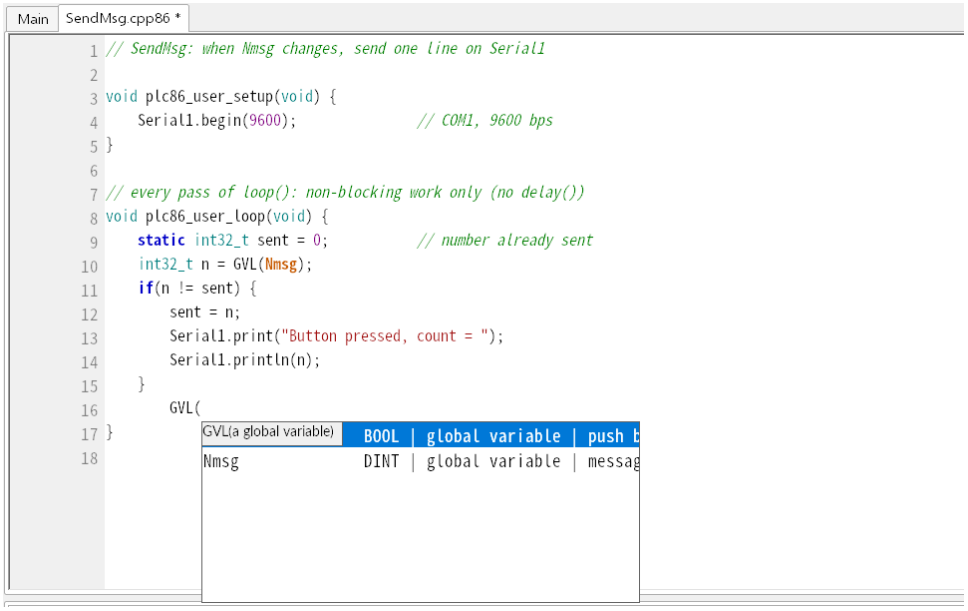


Figure 14-5 Typing GVL(lists the global variables (with types and comments)

14.2.1 Type mapping

PLC type	C type	PLC type	C type
BOOL	`bool`	REAL	`float`
SINT / USINT	`int8_t` / `uint8_t`	LREAL	`double`
INT / UINT	`int16_t` / `uint16_t`	TIME	`int32_t` (milliseconds)
DINT / UDINT	`int32_t` / `uint32_t`	Enumeration	`int32_t`
LINT / ULINT	`int64_t` / `uint64_t`	STRING[n]	`char[n+1]` (`STRING` is `char[81]`)
BYTE / WORD	`uint8_t` / `uint16_t`	DWORD / LWORD	`uint32_t` / `uint64_t`

***Tip:** In C, a STRING is a zero-terminated character array: to read it, use it directly as a const char *; to write it, use strncpy(GVL(Msg), "text", sizeof(GVL(Msg)) - 1); GVL(Msg)[sizeof(GVL(Msg)) - 1] = '\0'; so that it does not exceed the declared length and is always zero-terminated.

14.3 Hook functions

Hook functions **are called if they are defined**; the ones you do not use can be deleted. They let your C/C++ step in at fixed points of the PLC execution:

Function	When it is called	Suited to
`plc86_user_setup()`	In `setup()`, after the PLC is initialized	Initializing `Serial1.begin()`, Ethernet, CAN, Modbus, SCoop and so on
`plc86_user_before()`	Every scan of the first task: after the inputs are read, before the program runs	Passing external data to the PLC (pre-processing inputs)
`plc86_user_after()`	Every scan of the first task: after the program runs, before the outputs are written	Reading the PLC's results (post-processing outputs, reporting status)
`plc86_user_loop()`	Every pass of `loop()` (between scans)	Non-blocking communication; no `delay()`
`plc86_user_ethercat()`	Every EtherCAT cycle (the MainDevice's cycle callback, in an interrupt)	Synchronized control; keep it short
`plc86_user_ecat_config()`	After the EtherCAT SubDevices are attached, before start	Initial SDO parameters, MainDevice settings

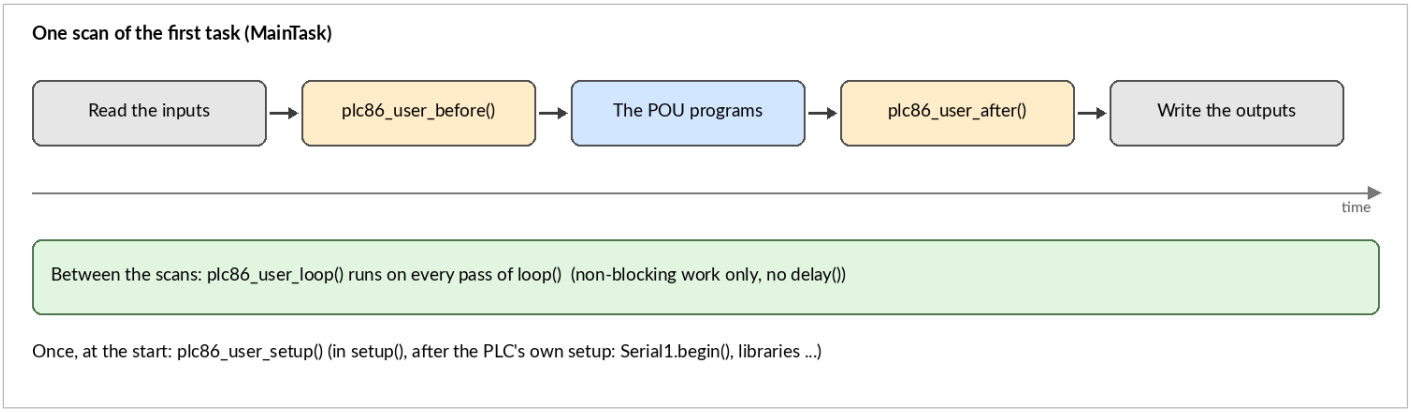


Figure 14-6 Where the hook functions sit in a scan: read inputs → before → program → after → write outputs; loop between scans

- `plc86_user_ethercat()`: available whenever the project has EtherCAT devices; you do not need to create an EtherCAT task. When there is an EtherCAT task, it is called after that task's program. It is not called before `plc86_setup()` has finished.
- `plc86_user_ecat_config()` is not in the template; add it yourself when needed (in the user code or in the `.ino`).
- The completion list can insert empty definitions of these functions directly.

14.4 Examples

14.4.1 Serial1: start on a command, report when the state changes

Scenario: an external device sends the characters `S` (start) and `T` (stop) through the controller's COM1 (RS-232 / 485); when the PLC state changes, it sends back `RUN` / `STOP`. The physical buttons still work. (For an introductory example of sending messages with Serial1, see section 6.7.)

Variable	Type	I/O mapping	Use
Xstart	BOOL	LocalIO.X00	Physical start button
Xstop	BOOL	LocalIO.X01	Physical stop button
Rstart	BOOL	(none)	Start command from Serial1 (one scan)
Rstop	BOOL	(none)	Stop command from Serial1 (one scan)
Rrun	BOOL	(none)	Running (latched)
Yrun	BOOL	LocalIO.Y00	Run indicator lamp

Ladder diagram (Main):

```
NETWORK 'start/stop'
    [ NO(Xstart) | NO(Rstart) | NO(Rrun) ] NC(Xstop) NC(Rstop) OUT(Rrun)
NETWORK 'lamp'
    NO(Rrun) OUT(Yrun)
```

User code (MyCode.cpp86):

```
static bool cmdStart = false, cmdStop = false;

void plc86_user_setup(void)
{
    Serial1.begin(9600);                    // the controller's COM1
}

void plc86_user_loop(void)                  // every pass of loop(): no delay()
{
    while(Serial1.available()) {            // read only the characters already received
        char c = Serial1.read();
        if(c == 'S') cmdStart = true;
        if(c == 'T') cmdStop = true;
    }
}

void plc86_user_before(void)                // before the program runs: hand the command to the PLC
for one scan
{
    GVL(Rstart) = cmdStart;
    GVL(Rstop) = cmdStop;
    cmdStart = cmdStop = false;
}

void plc86_user_after(void)                 // after the program runs: report when the state changes
{
    static bool last = false;
    if(GVL(Rrun) != last) {
        last = GVL(Rrun);
        Serial1.println(last ? "RUN" : "STOP");
    }
}
```

#	Action
1	Declare the variables in the table above in the GVL; map Xstart, Xstop and Yrun to the local I/O as described in section 4.6
2	In Main, paste the ladder diagram in text mode (Ctrl+T) and press "Back to the diagram"
3	"Project > Add a user code file (C/C++)...", name it MyCode, and paste the code
4	Press **Check** (Shift+F7), then **Upload** (F8)
5	Connect COM1 to a USB-to-RS-232 / 485 adapter on the computer, choose that port and 9600 in the "Serial monitor", press **Connect** and send `S`

Expected result: after sending `S`, `Rrun` and `Yrun` become TRUE and the serial monitor receives `RUN`; after sending `T` or pressing the physical stop, it receives `STOP`. `Rstart` / `Rstop` are TRUE for one scan only, so they do not get "stuck".

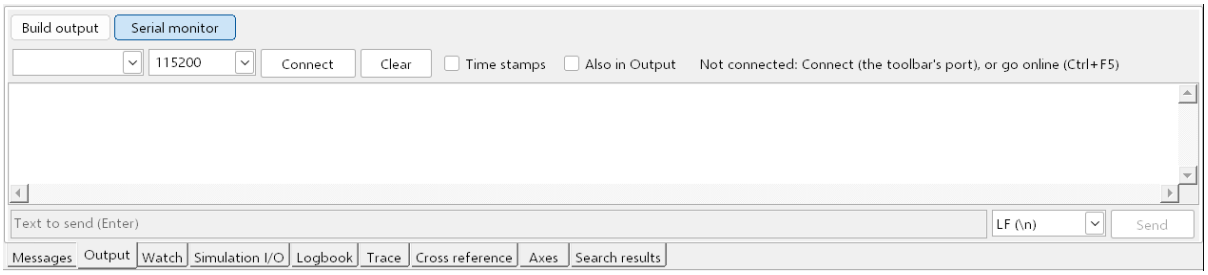


Figure 14-7 The serial monitor: port, baud rate, connect, time stamps, send

14.4.2 HTTP status page (idea)

Use the Ethernet library of 86Duino to make the simplest HTTP server: open `http://192.168.1.50/` in a browser to see the PLC variables as JSON. The following shows how to do it; change the IP address and so on to suit the network on site:

```
#include <Ethernet.h>

static byte mac[] = {0x02, 0x86, 0x00, 0x00, 0x00, 0x01};
static IPAddress ip(192, 168, 1, 50);
static EthernetServer web(80);

void plc86_user_setup(void)
{
    Ethernet.begin(mac, ip);
    web.begin();
}

void plc86_user_loop(void)                  // no waiting: return at once when there is no connection
{
    EthernetClient client = web.available();
    if(!client)
        return;
    while(client.available())               // read away the request (not parsed here)
        client.read();
    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: application/json");
    client.println("Connection: close");
    client.println();
    client.print("{\"run\":");
    client.print(GVL(Rrun) ? "true" : "false");
    client.print(",\"level\":");
    client.print(GVL(Level), 1);
    client.print(",\"count\":");
    client.print(GVL(Ncount));
    client.println("}");
    client.stop();
}
```

- The communication is written in plc86_user_loop(): it runs between scans and is not put in a PLC task; but the time it takes delays the next scan, so each pass must be very short (do not wait for a connection or data), and measure how long it takes separately as described in the next point. This example only shows the approach: when the network is congested, print() and client.stop() can also take some time.
- The time of plc86_user_loop() is not counted in the scan time of the tasks, so neither the watchdog nor the scan time in the status bar shows the delay it causes. Measure it with micros() at the start and end of plc86_user_loop(), write the maximum to a GVL variable (for example GVL(LoopMaxUs)) and watch it in the Watch window; it should be much shorter than the task's interval.
- The variables read in the response (`Rrun`, `Level`, `Ncount`) must be declared in the GVL. To **write** to the PLC from the web page, do it as in section 14.4.1: first store the value in a C variable, then hand it to the PLC in `plc86_user_before()`.

14.4.3 Synchronized output in every EtherCAT cycle

```
void plc86_user_ethercat(void)           // every EtherCAT cycle (in an interrupt)
{
    if(!ECAT_OK) return;
    static uint32_t n = 0;
    n++;
    GVL(Ncycle) = (int32_t)n;               // for LD / ST to see
    // DQ01 is written only by this code (do not also map it to a variable, and do not write it from
other code)
    bool on = GVL(Rrun) && ((n / 500) & 1);       // toggles every 500 cycles: every 0.5 s with a 1 ms
cycle (one on/off about 1 s)
    ECAT_SLAVE(S0__QEC_R11CFFG).digitalWrite(1, on ? HIGH : LOW);   // writes LOW when Rrun is FALSE
}

void plc86_user_ecat_config(void)         // after attach, before start
{
    // ECAT_SLAVE(S0__QEC_R11CFFG).sdoDownload8(index, subindex, value);   // example: SDO setting;
fill in the object and valid values from the module's object dictionary
}
```

The SubDevice name (for example `S0__QEC_R11CFFG`) is the same as the name under "Devices > EtherCAT" in the project tree (Chapter 11). DMP's QEC modules use their driver classes (for example `EthercatDevice_QECR11CFFG`), on which you can call `digitalWrite`, `digitalRead`, `analogRead` and so on; CiA 402 drives use `EthercatDevice_CiA402`, and other SubDevices use `EthercatDevice_Generic` (`pdoGetInputBuffer()` / `pdoGetOutputBuffer()`, SDO).

14.5 The sketch's .ino

The sketch's `.ino` needs only three lines (the build adds them automatically); you can put your own code in the rest:

```
#include "plc86.h"
void setup() { plc86_setup(); /* your own initialization */ }
void loop()  { plc86_loop();  /* your own code; do not delay() too long */ }
```

- The EtherCAT MainDevice, serial ports, 86HMI, Modbus and the online view are all started by `plc86_setup()` according to the project's devices, so the `.ino` does not need to, and must not, call `EVA.begin()` or `EthercatMaster::begin()` again. When 86HMI is used, the build adds `#include "myhmi.h"` and `Hmi.begin();`.
- After `#include "plc86.h"`, the `.ino` and the sketch's `.cpp` files can use `GVL(name)`, `LD_READ` and `LD_WRITE` to access GVL variables of elementary types (BOOL, integers, REAL, TIME), and can also use the EtherCAT objects and `mc_*`; arrays, structures, `ST()`, `FUN()`, `ECAT_SLAVE()` and `M86_MACHINE()` are available only in user code (.cpp86).
- With "Sketch (86Duino IDE)" in the project tree or "Project > The sketch's .ino (the 86Duino IDE's)", you can edit the `.ino` and other `.cpp` / `.h` files directly in 86LDmicro. They are saved before saving, building or compiling; after the build patches the `.ino`, its tab updates automatically.

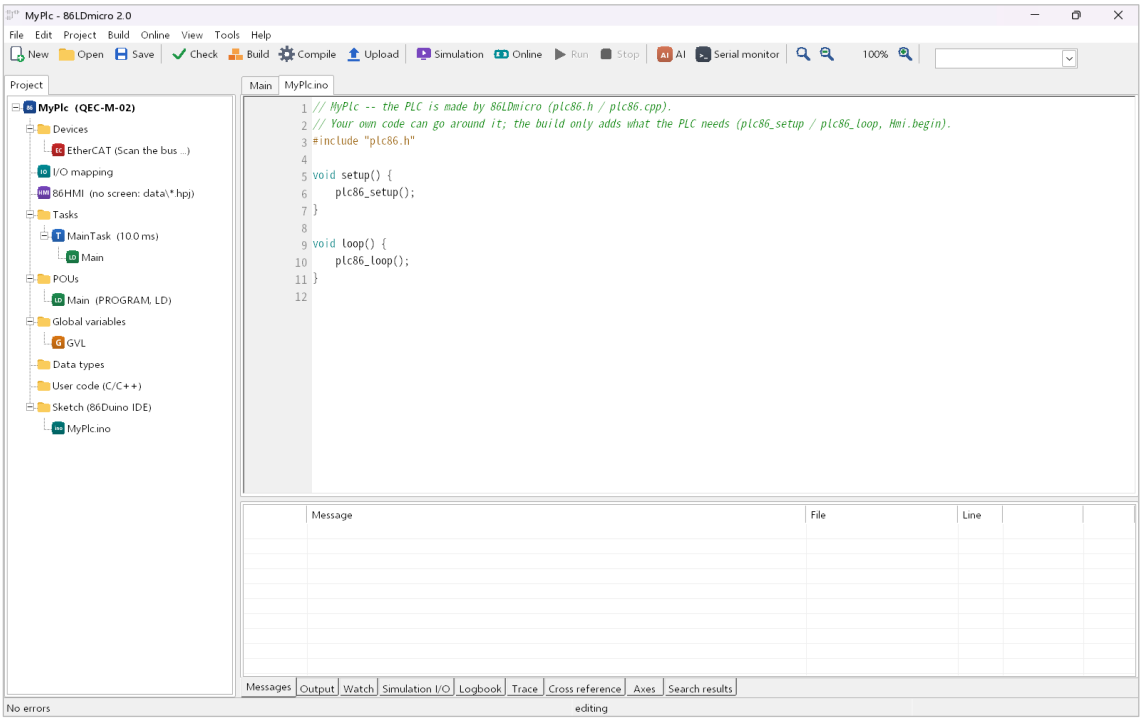


Figure 14-8 Editing the sketch's .ino in 86LDmicro

After a build, the sketch contains the following extra files (rewritten at every build, **do not edit them by hand**):

File	Contents
`plc86.h` / `plc86.cpp`	The PLC program: variables, POUs, tasks, I/O mapping, 86HMI bindings, user code
`plc86_rt.h` / `plc86_rt.cpp`	The runtime (the same for every project): online view (read / write, force, RUN / STOP), RETAIN, Logbook
`plc86_ecat.h` / `plc86_ecat.cpp`	EtherCAT MainDevice and SubDevice objects, start-up sequence (`begin` → `attach` for each SubDevice → `plc86_user_ecat_config()` → `start`)

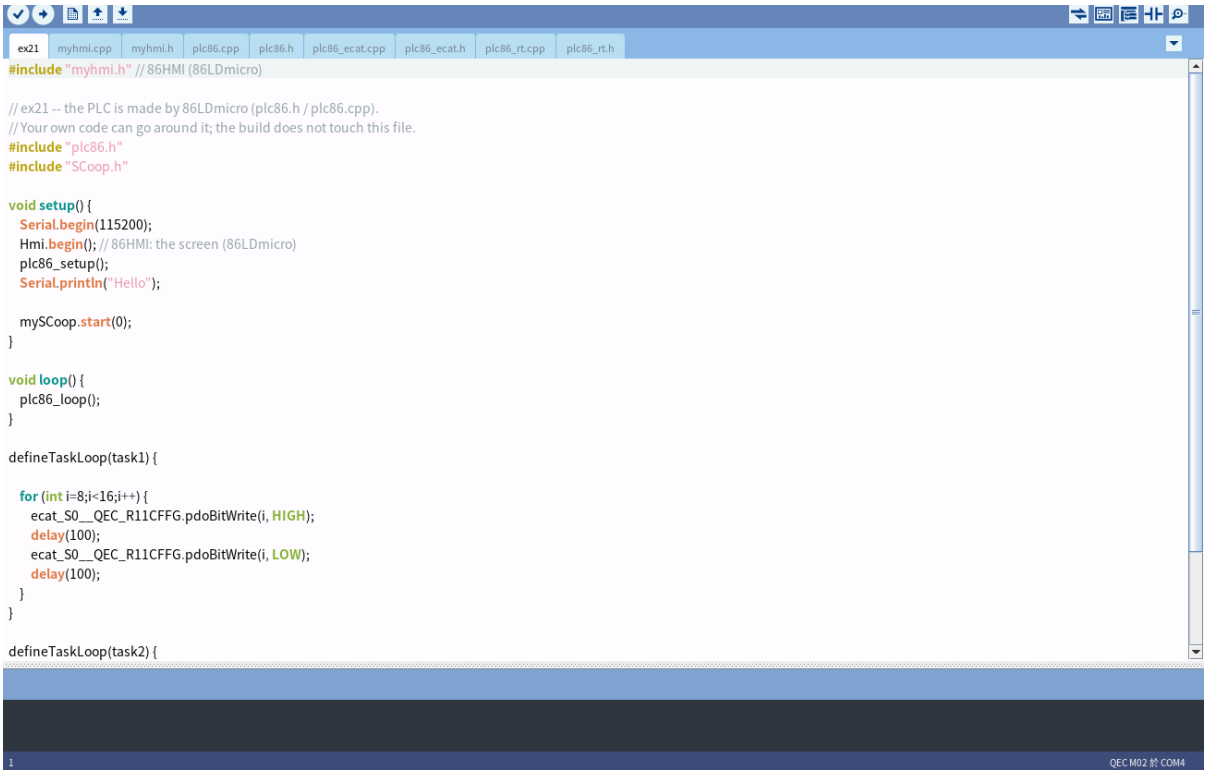


Figure 14-9 The 86Duino IDE after a build: tabs plc86.cpp, plc86.h, plc86_ecat.cpp and so on

14.6 Points to note

Note	Reason and what to do
Do not print continuously to the USB `Serial`	The USB `Serial` is used for uploading and the online view. When the computer is not reading (86LDmicro not connected, serial monitor closed), `Serial.print()` waits once the buffer is full, and **the whole program (including the PLC scan) stops with it**. Use `Serial1` to `Serial3` to communicate with external devices; for debug printing, first check that `Serial.availableForWrite()` is large enough
`plc86_user_ethercat()` runs in an interrupt	No `delay()`, `Serial.print()` or dynamic memory allocation (malloc / new); check `ECAT_OK` first; floating-point operations are allowed. The processing time must be much shorter than the EtherCAT cycle
`plc86_user_loop()` must not wait	No `delay()`, and no `while` waiting for an external response; process only the data already received each time
Do not edit the `plc86*` files by hand	They are rewritten at every build; to adjust the EtherCAT start-up sequence, use `plc86_user_ecat_config()`
Variables already mapped to physical inputs	A value written from C is overwritten when the inputs are read in the next scan; write an internal variable (for example `Rstart`) and put it in parallel with the physical input in the program
C++ standard headers	`#include <vector>` and the like are automatically placed before Arduino.h (avoiding conflicts with the min / max macros)
Initializing Serial	When the online view is enabled, the runtime has already opened `Serial`; otherwise, in `setup()` of the `.ino` or in `plc86_user_setup()`, call it yourself: `Serial.begin(115200);`

*__Note:__ If the `.ino` from 1.1 still has `#include "myeva.h"` and `EVA.begin()`, remove them when you use the EtherCAT of 2.0, and move `myeva.cpp` / `myeva.h` out of the sketch (the build comments out `EVA.begin()` automatically and warns you; see section 15.1).

Chapter 15 Import, AI, the command line and project protection

This chapter describes several auxiliary features: importing programs from LDmicro 1.1 or other IEC 61131-3 tools into 86LDmicro, using the AI assistant and MCP, batch processing with the command-line tool, and protecting a project with a password.

15.1 Importing an LDmicro 1.1 project

An LDmicro 1.1 program (`.ld`, usually `data\myLD.ld` in the sketch) can be converted into an 86LDmicro 2.0 project. The project is written to `data\86ld` of the same sketch; **the original `.ld` file is not modified**, so you can still open it with 1.1 when needed.

15.1.1 Importing from the 86Duino IDE

1. In the 86Duino IDE, open the sketch that contains `data\myLD.ld` and choose "**Tools > 86LDmicro**".
2. 86LDmicro detects the 1.1 program and asks whether to import it; click **Yes**.
3. When the import is complete, the project opens: the 1.1 ladder diagram becomes a POU, the variables go into the GVL, and the I/O pins become I/O mappings.

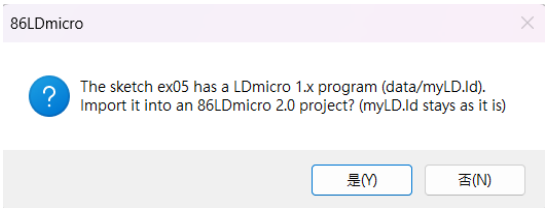


Figure 15-1 The question shown when a 1.1 project is detected

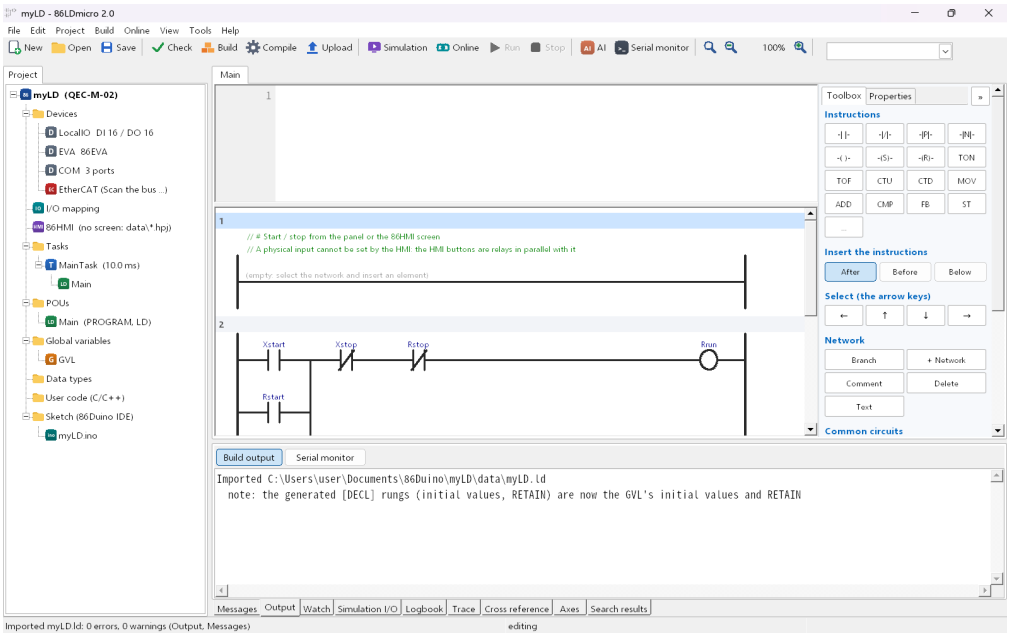


Figure 15-2 The project after the import: the 1.1 ladder diagram and ST become POUs, placed in MainTask in their original order

15.1.2 Importing from 86LDmicro

1. Choose "**File > Import a LDmicro 1.x program or a PLCopen XML (.ld, .xml)...**" and select the `.ld` file, or drag and drop the `.ld` file directly onto the 86LDmicro window (you are first asked "Import … as a new 86LDmicro project?").
2. "**The sketch folder**" in the "Import" form: if the `.ld` file is in the `data` folder of a sketch, it defaults to that sketch; otherwise to a folder next to the `.ld` file with the same name as the file. The project is written to `<sketch>\data\86ld`. Click **OK**.
3. The status bar shows "Imported …: N errors, M warnings (Output, Messages)", and the first program of the first task opens.
4. `.ino` is not overwritten if it already exists; in it, change the 1.1 `setupPlc()` / `loopPlc()` to `plc86_setup()` / `plc86_loop()` (section 14.5); the rest of the code stays unchanged.
5. Click **Check** (Shift+F7) to make sure there are no errors, then **Simulate** (F5) to verify the behavior, and finally upload.

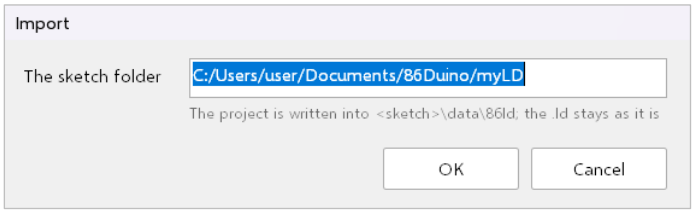


Figure 15-3 The "Import" form: the sketch folder (the project is written to <sketch>\data\86ld; the original .ld file is unchanged)

15.1.3 What becomes what

LDmicro 1.1	86LDmicro 2.0
Ladder diagram (one program)	`pou/Main.lad` (`Ladder` if an ST program is already named Main)
ST programs (POU, before / after, task=ecat)	`pou/*.st`; the order before / after the ladder diagram becomes the order in the task, and the EtherCAT cycle becomes the EtherCAT task
Library FBs, ST blocks	FUNCTION_BLOCK / FUNCTION in `pou/*.st`
Data types	`dut/*.dut`
VAR LIST, DECLARATIONS (initial values, RETAIN, inversion, filter, scaling)	`gvl/GVL.gvl` (types follow the size rules of 1.1) + inversion, filter and scaling in the I/O mapping
IO LIST (pins, Modbus)	I/O mapping (LocalIO, Pin, Modbus)
BINDINGS (EVA points, servos, HMI)	I/O mapping (EVA.*) and bindings (servos, 86HMI)
User code (global / setup / beforescan / afterscan / loop / ethercat)	`usercode/*.cpp86` hook functions `plc86_user_setup`, `_before`, `_after`, `_loop`, `_ethercat`; `LD_READ` / `LD_WRITE` work as before (section 14.2)
86EVA, COM mapping, Modbus, Ethernet, Modbus server	Devices
Online monitoring	"Online" in 86LDmicro (Ctrl+F5, section 10.4)

15.1.4 Notes

- Timer, counter and other 1.x instructions keep their 1.x semantics (counting in scan cycles) and behave the same as in 1.1; for new programs, use the IEC function blocks instead (section 7.8).
- Variables with the same name as an IEC reserved word get `_v` appended (for example `TON` → `TON_v`); names beginning with __ are reserved for internal use by 2.0.
- When you switch to the 2.0 EtherCAT (Chapter 11), remove the 1.x `#include "myeva.h"` and `EVA.begin()` from the `.ino`, and `myeva.cpp` / `myeva.h` from the sketch (the build comments out `EVA.begin()` automatically and reminds you).
- `MC_ReadActualPosition.Position` is a REAL in 2.0; change programs that connected it to a DINT to use REAL, or use `REAL_TO_DINT`.
- Compile errors no longer open the AI window automatically; when needed, click "Fix with AI" in "Messages" (section 15.3).

15.2 Importing PLCopen XML

PLCopen XML (TC6) is the standard exchange format of IEC 61131-3. `.xml` files exported by tools such as OpenPLC Editor, Beremiz and CODESYS can be imported as a new 86LDmicro project, in the same way as in section 15.1.2.

1. Export PLCopen XML in the original tool (for example, CODESYS: "Project > Export PLCopenXML"; the project file `plc.xml` of OpenPLC Editor / Beremiz is already in this format).
2. In 86LDmicro, choose "**File > Import a LDmicro 1.x program or a PLCopen XML (.ld, .xml)...**" and select the `.xml` file; or **drag and drop** the `.xml` file directly onto the 86LDmicro window and click **Yes** when asked.
3. In the "Import" form, choose the sketch folder and click **OK**.
4. Look at the numbers of errors and warnings in the status bar; "Output" lists the number of imported POUs (how many LD, ST and SFC) and the parts that were not imported.
5. Click **Check** and **Simulate** to verify, then map the I/O (section 4.6) and upload.

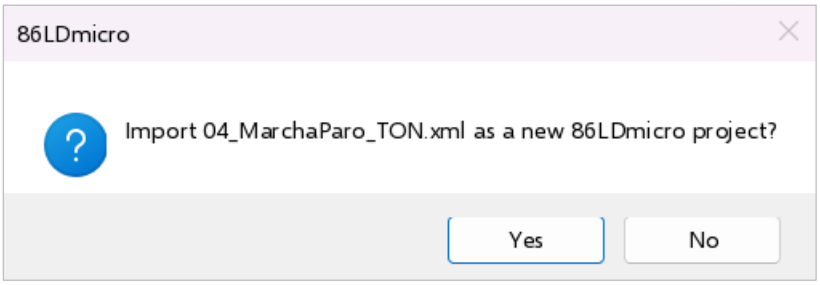


Figure 15-4 Dragging an .xml file onto the window: the question whether to import it as a new 86LDmicro project

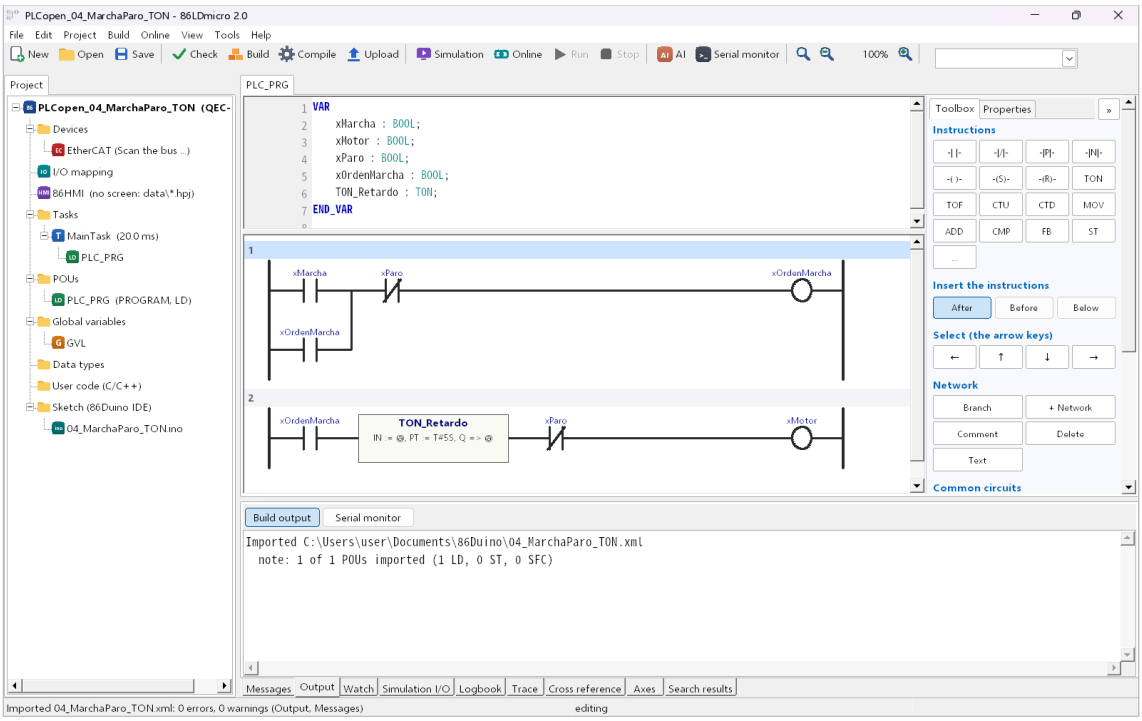


Figure 15-5 After a PLCopen XML import: the POUs, the GVL and the import summary in "Output"

PLCopen content	Import result
LD (ladder diagram)	`.lad` program (contacts, coils, function blocks, branches)
ST	`.st` program
SFC	`.sfc` program (steps, transitions, actions, branches)
Data types (STRUCT, enumerations, arrays)	`.dut`
Global variables (configuration / resource)	`.gvl`; `AT %IX…` addresses are kept only as comments: link them again in the I/O mapping
Tasks and program instances	The project's tasks and assignments
VAR_INPUT / VAR_OUTPUT of a PROGRAM	Changed to VAR (a PROGRAM in 86LDmicro has no parameters); explained in the messages
VAR_EXTERNAL	Omitted (every POU can see the global variables); explained in the messages
FBD, IL	**Not imported in this version**; listed in the messages. Rewrite them as ST or LD in the original tool, then export again

Command line: `86ldmicro-cli import <file.xml> <sketch folder> [name]`; the MCP tool `ld86_import_11` also accepts PLCopen XML.

15.3 AI assistant

The AI assistant can write or fix ST / LD / C programs, explain errors and plan the I/O. It uses **your own API key** (or a local model), and the requests go directly from this computer to the server you set; no data is sent before you click "Send" (or "Test connection").

15.3.1 The AI Assistant tab

Click **AI** on the toolbar, or choose "**Tools > AI assistant (MCP)...**", to open the "AI Assistant" tab:

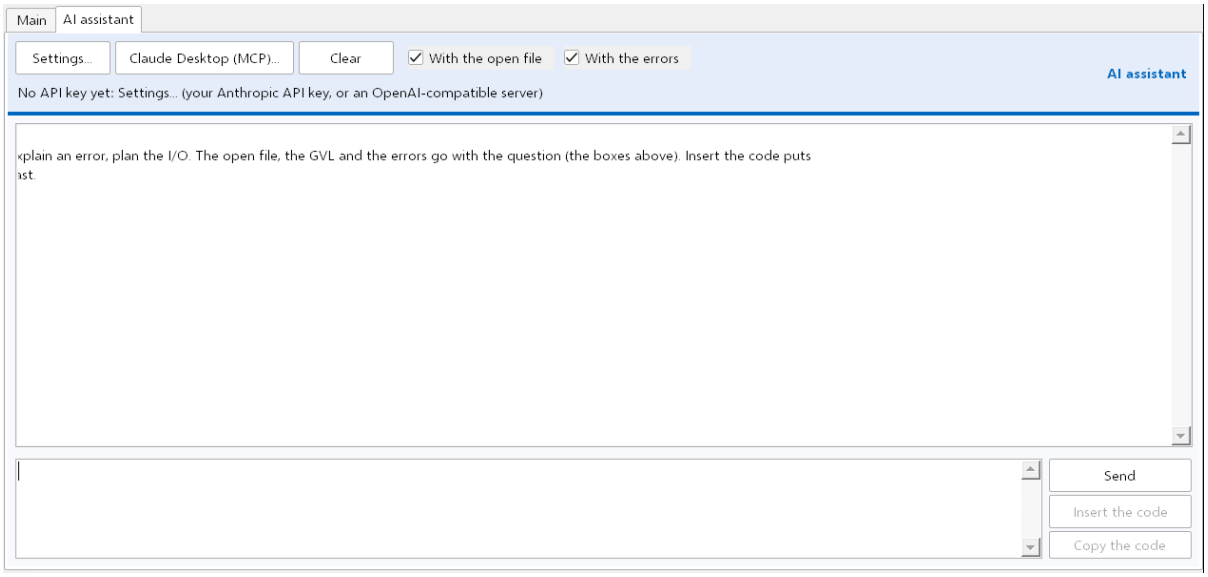


Figure 15-6 The AI Assistant tab: Settings, Claude Desktop (MCP), Clear, attaching the open file and the errors

Item	Purpose
Settings...	Choose the service, enter the API key and the model (section 15.3.2)
Claude Desktop (MCP)...	Set up MCP (section 15.4)
Clear	Clears the conversation
With the open file, With the errors (check boxes)	When checked, the question is sent together with the file currently open and the errors in "Messages" (the GVL and the tasks are always attached)
Send (Ctrl+Enter)	Sends the question typed below
Insert the code	Puts the code from the last answer into the editor: ST and C code is inserted at the cursor of the last used editor; an LD NETWORK is inserted after the selected network (new variables in a VAR block are declared automatically)
Copy the code	Copies the code in the answer to the clipboard

15.3.2 Settings: service and key

Click **Settings...** to open "AI assistant: settings":

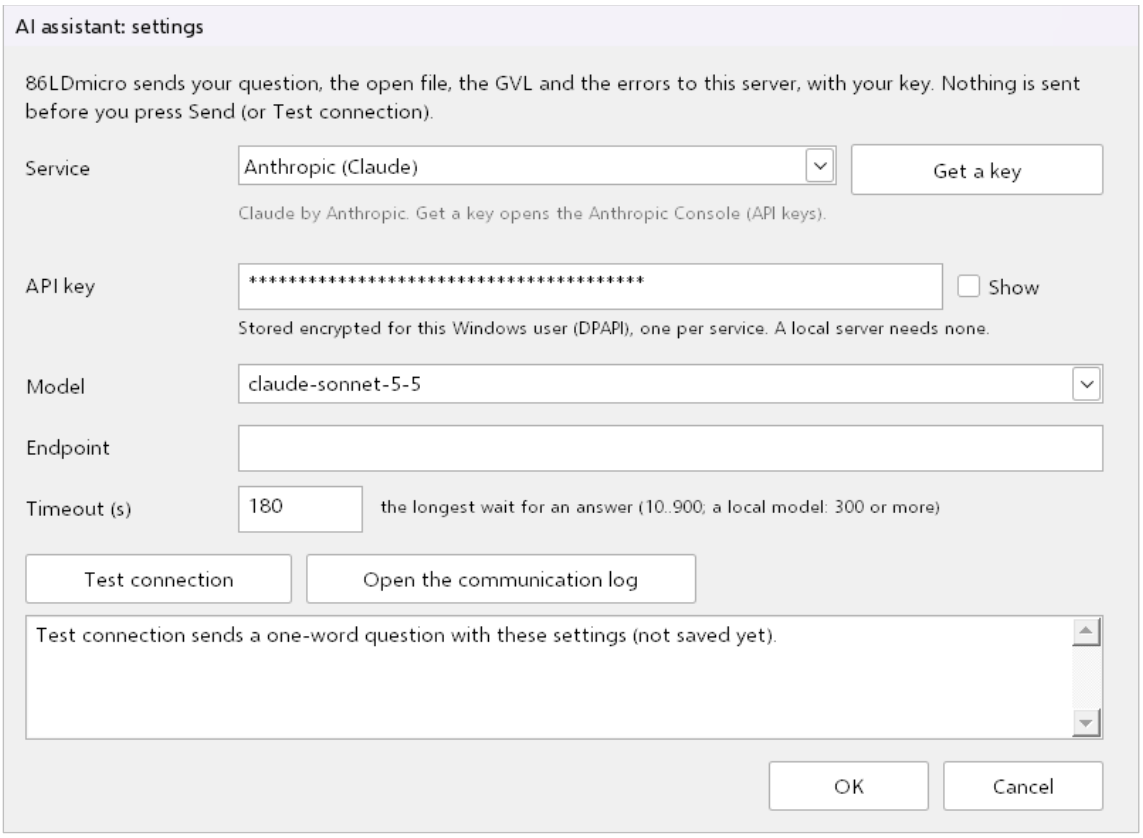


Figure 15-7 "AI assistant: settings": service, Get a key, API key, model, endpoint, timeout, Test connection, Open the communication log

Field	Description
Service	Anthropic (Claude), OpenAI, Google Gemini, Ollama (local), LM Studio (local), other OpenAI-compatible servers. When you choose one, the endpoint and the model are filled in automatically
Get a key (**Download** for local services)	Opens the service's key page in the browser (Anthropic Console, OpenAI, Google AI Studio) or the download page of the local software
API key	Stored encrypted for this Windows user (DPAPI), one key per service; not needed for local servers. Check "Show" to see it
Model	Choose one from the list, or type the name of a model that the server lists
Endpoint	Empty: the service's default; for Azure, a company gateway, vLLM and so on, enter the server's URL
Timeout (s)	The longest time to wait for an answer (10..900, default 180; 300 or more is recommended for local models)

*__Tip:__ The key, the model and the endpoint may only contain ASCII characters; if you get a related error, check whether you pasted full-width characters.

15.3.3 Test connection and the communication log

1. Click **Test connection** in the settings window: it sends a short question with the current settings (not saved yet) and runs in the background; during the test the button becomes **Stop**, which you can click to stop it.
2. On success it shows `OK: <model> answered in N ms (...)`; on failure it explains the reason in English (see the table below).
3. Click **Open the communication log** to see the URL, status, duration and the beginning of the response of each request (without the key). The log file is `%LOCALAPPDATA%\86LDmicro\ai_comm.log`.
4. When everything is correct, click **OK** to save.

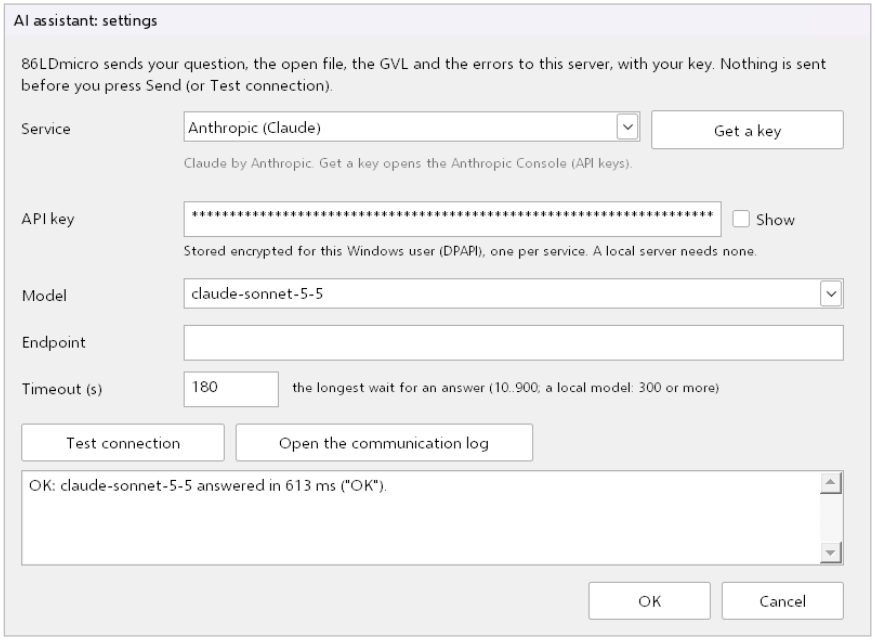


Figure 15-8 The result of Test connection: OK: … answered in N ms

Test connection message	Meaning and what to do
401 Unauthorized	The API key is wrong, expired, revoked or belongs to another service: paste the key again (Get a key)
403 Forbidden	The key is valid but cannot use this model or endpoint (plan, organization, region, project disabled)
404 Not Found	Wrong endpoint URL or model name; local server: is the model loaded / downloaded (ollama pull)?
408 Request Timeout	The server gave up waiting: try again
429 Too Many Requests	Rate limit exceeded or quota used up: wait a moment and try again, or check the account's billing
5xx Server error	A problem on the service side: try again later
Timeout	No answer within the set number of seconds: the server is busy or the local model is still loading; increase the timeout or try again
Network: …	Cannot connect: no network, DNS, wrong endpoint, server not started (Ollama / LM Studio), firewall, proxy, http / https mix-up, TLS connection failure (a proxy inspecting HTTPS, the computer's date and time)

15.3.4 Asking questions and inserting code

1. Open the POU you want to discuss (for example Main), type a question at the bottom of the AI tab, for example "Make Rrun start only 2 seconds after Xstart is pressed", and click **Send** (Ctrl+Enter).
2. The answer is shown above. When it contains code, click **Insert the code** to put it into the editor, or **Copy the code**.
3. After the insertion, the status bar shows "The AI's code is in the editor: check it (Check, Shift+F7)": be sure to click **Check** and **Simulate** to verify it.

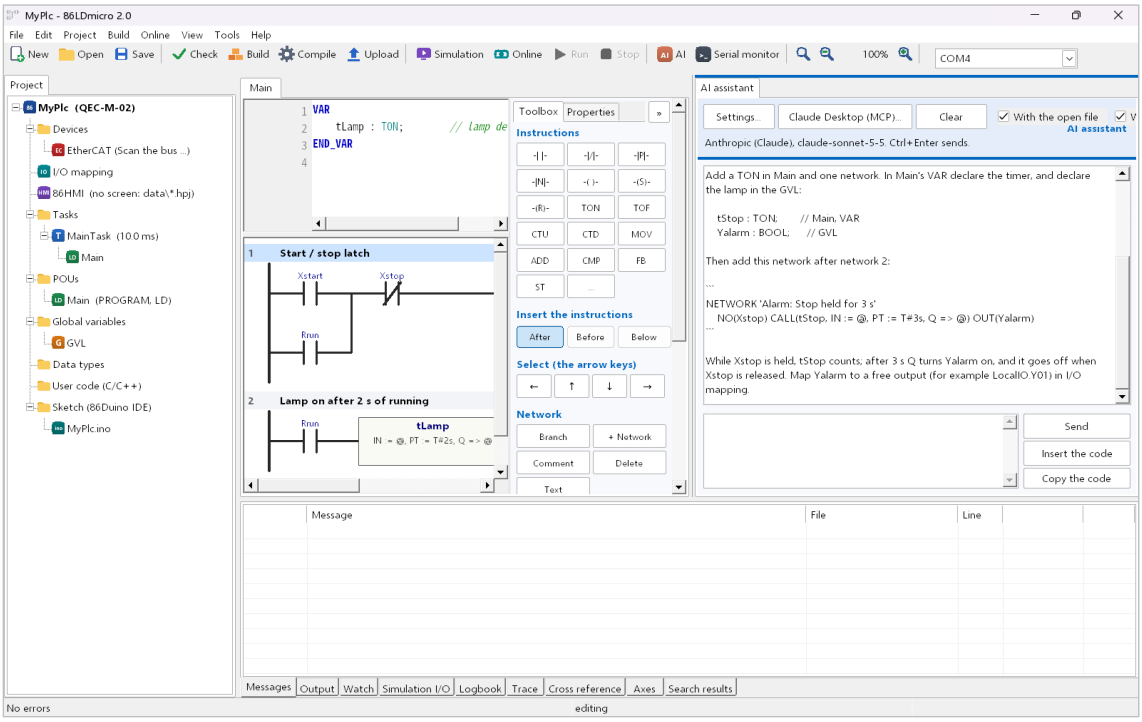


Figure 15-9 The AI's answer and "Insert the code"

15.3.5 Fixing errors with AI

Each error row in the "Messages" window has **Fix with AI** at the far right: if AI is set up, it asks about the error and the code around it directly in the AI tab; if not, it copies the question to the clipboard so that you can paste it into another AI tool.

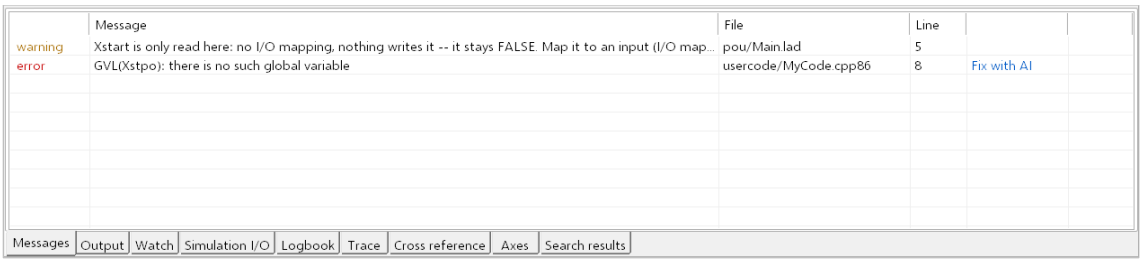


Figure 15-10 "Fix with AI" in the "Messages" window

> ***Note:** The AI's answers can be wrong. Make sure you understand the inserted code yourself, and check and simulate it before uploading it to the real machine.

15.4 MCP

The MCP (Model Context Protocol) server lets AI tools that support MCP, such as Claude Desktop, Cowork and Claude Code, read, modify, check and simulate 86LDmicro projects directly. The MCP server is installed in the `mcp` folder of 86LDmicro (`mcp\ld86_mcp.py`) and requires Python 3.

15.4.1 Setting up Claude Desktop

1. Install Python 3 (python.org; check "Add to PATH" during the installation).
2. On the AI Assistant tab, click **Claude Desktop (MCP)...** to open the "AI assistant (MCP)" window. The top of the window shows the status of the MCP server, Python and the current project.
3. Click **Set up Claude Desktop (MCP)**: it adds 86ldmicro to `claude_desktop_config.json` (the original file is backed up as `.bak`).
4. Restart Claude Desktop, then ask it directly, for example: "Open the 86LDmicro project in <folder> and explain MainTask".
5. Other MCP clients (Cowork, Claude Code…): click **Copy the MCP configuration (JSON)** and paste it into the MCP settings of that tool.

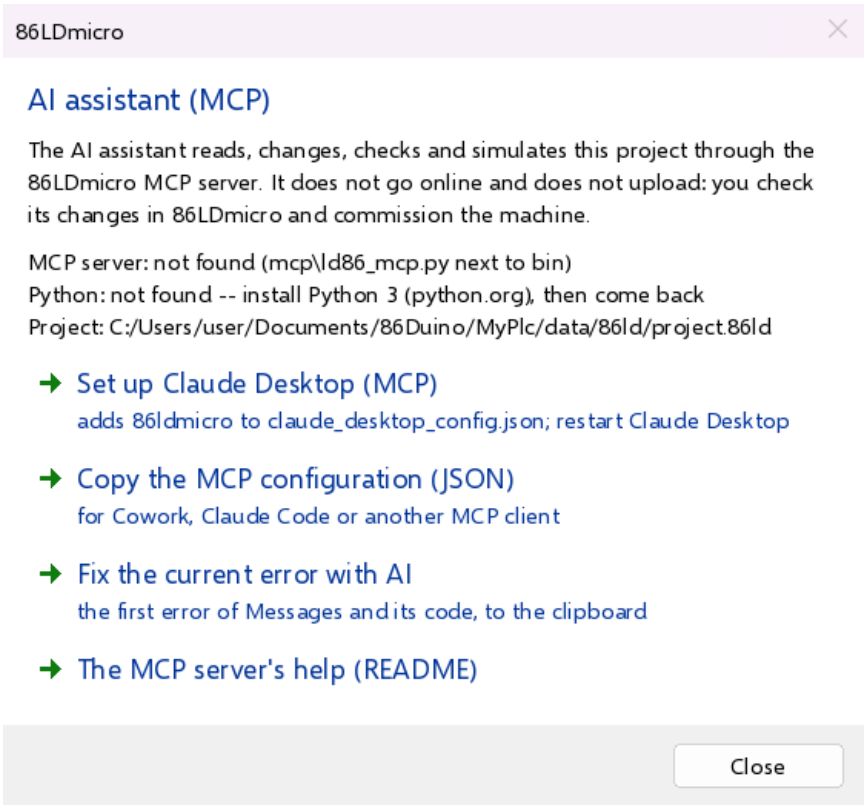


Figure 15-11 "AI assistant (MCP)": Set up Claude Desktop, Copy the MCP configuration, Fix the current error with AI, the MCP server's help

Item in the window	Purpose
Set up Claude Desktop (MCP)	Adds 86ldmicro to claude_desktop_config.json; then restart Claude Desktop
Copy the MCP configuration (JSON)	For Cowork, Claude Code or another MCP client
Fix the current error with AI	Copies the first error in "Messages" and the code to the clipboard
The MCP server's help (README)	Opens `mcp\README.md`

15.4.2 MCP tools

MCP tool	Purpose
ld86_describe	Project overview: tasks, POUs, GVL, data types, user code, devices, I/O mapping
ld86_read_file / ld86_write_file	Reads / adds or replaces POUs, the GVL, data types and user code (compiled automatically after writing, and the messages are returned)
ld86_set_project	Sets tasks, devices, I/O mapping, bindings and so on, then checks
ld86_check / ld86_simulate	Compiles and lists the errors; simulates N scans and lists the variables
ld86_build	Writes the generated files of the sketch (no compile, no upload)
ld86_new_project / ld86_import_11 / ld86_ethercat_eni	New project, import of 1.x or PLCopen XML, EtherCAT devices created from an ENI
ld86_lower / ld86_instructions	The ST that LD is converted to; description of the instruction set

15.4.3 Safety

- The MCP server **does not connect to the controller and does not upload**: verify, simulate and test-run a project changed by AI in 86LDmicro, then upload it yourself.
- When AI writes files through MCP while 86LDmicro has the same project open, choose "**View > Refresh: the project again from the disk**" (Ctrl+R) to load the changes; if there are unsaved changes, you are asked first.
- Use version control or back up `data\86ld` of the sketch first, so that you can compare and undo the changes made by AI.

15.5 Command-line tool

`tool\bin\86ldmicro-cli.exe` provides the same check, build, simulation, upload and online functions as 86LDmicro, and can be used in batch files or automated tests. Run it without arguments to list all commands:

```
> 86ldmicro-cli
86ldmicro-cli 2.0
  import <file.ld | file.xml> <sketch folder> [name]
  check  <project>
  search <project> <text> [--case] [--word]
  build  <project>
  lower  <project> <POU>
  new    <sketch folder> [name]
  compile <project> [--ide <86Duino IDE folder>]
  upload <project> <port> [--ide <dir>]
  upload-res <project> <port> [--ide <dir>]
  iotable export|import <project> <file.xlsx|.csv>
  format <file>
  ports
  instructions
  online <project> <port> status | read <path>... | write <path> <value> | ...
  eni    <project> <eni.xml> [name on the controller (C:/eni/)]
  sim    <project> <scans> [path...] [--set path=value]... [--diff-inputs]
```

Command	Purpose
`new <sketch folder> [name]`	Creates a new project (MainTask 10 ms, ladder diagram Main, GVL)
`import <myLD.ld | file.xml> <sketch folder> [name]`	Imports a LDmicro 1.x program or PLCopen XML (sections 15.1, 15.2)
`check <project>`	Check (same as Shift+F7)
`build <project>`	Build: writes plc86.cpp etc. to the sketch (same as Ctrl+B)
`compile <project> [--ide <folder>]`	Compiles without uploading (same as F7)
`upload <project> <COM> [--ide <folder>]`	Compiles and uploads (same as F8)
`upload-res <project> <COM>`	Uploads only the 86HMI resources `data\Resource` (section 13.6)
`sim <project> <scans> [variable...] [--set variable=value]...`	Simulates N scans and lists the variable values
`search <project> <text> [--case] [--word]`	Searches in the project (same as Ctrl+Shift+H)
`lower <project> <POU>`	Lists the ST that the POU is converted to
`iotable export|import <project> <file.xlsx|.csv>`	Exports / imports the I/O mapping table to / from Excel
`eni <project> <eni.xml>`	Imports an ENI and creates the EtherCAT devices (Chapter 11)
`format <file>`	Auto-format (same as Ctrl+T in the editor)
`ports`	Lists the serial ports
`instructions`	Lists the ladder instructions (JSON)
`online <project> <COM> status|read|write|force|unforce|unforce-all|run|stop|reset warm|cold|log|watch`	Online operations (section 10.4)
`online <project> <COM> sdo|coe <SubDevice> ...`	EtherCAT SDO read / write, reading the CoE object dictionary (section 11.6)

<project> can be project.86ld, the folder it is in, or the sketch folder. Example: check the MyPlc project of Chapter 5 (C:\work\MyPlc), then simulate 250 scans (MainTask 10 ms, about 2.5 seconds): set Rrun to TRUE before the first scan and list Rrun and Ylamp; from scan 201 (2 seconds) on, Ylamp becomes TRUE.

```
86ldmicro-cli check C:\work\MyPlc
86ldmicro-cli sim C:\work\MyPlc 250 Rrun Ylamp --set Rrun=TRUE
```

--set writes only once, before the first scan; it is not a force: the program or the simulated I/O can still overwrite the value afterwards. A variable mapped to an input channel (for example Xstart) is overwritten by the simulated input in every scan, so this example sets the internal variable Rrun. To create a new project, use 86ldmicro-cli new C:\work\Test Test (MainTask, an empty ladder diagram Main, GVL).

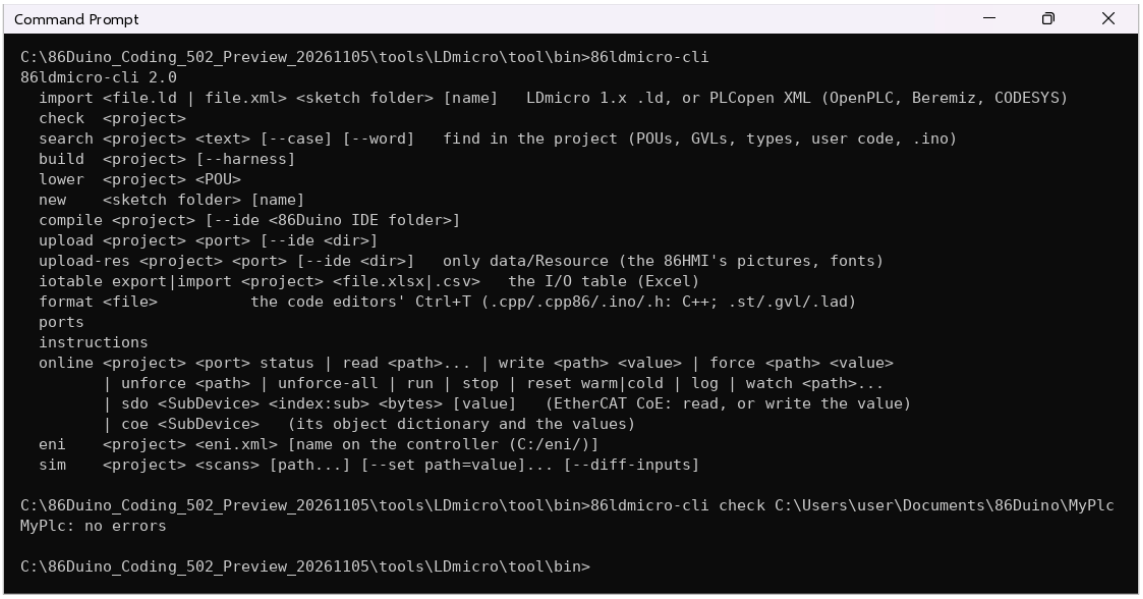


Figure 15-12 Running 86ldmicro-cli in the Command Prompt

15.6 Project protection

For projects delivered to customers, you can protect some of the programs with a password: set a password with "**Project > Protection (password)...**", then specify the protection mode of each POU and user code file.



Figure 15-13 "Protection": the password and the protection mode of each file

Protection mode	Effect
none	An ordinary file
read-only	Can be opened, simulated and built; cannot be changed without the password
hidden (encrypted)	Encrypted with AES-256 on the disk (the key is derived from the password with PBKDF2-SHA256), and only the PROGRAM line is kept on the disk; without the password the content is not shown, and the file cannot be opened, checked, simulated or built

15.6.1 Setting the protection

1. "Project > Protection (password)...".
2. Enter **Password** and **Password again** (at least 4 characters, the same both times).
3. Next to each file (for example `Main (pou/Main.lad)`), choose **none**, **read-only** or **hidden (encrypted)**, and click **OK**.
4. The status bar shows "The protection is set: Project > Protection > Lock to lock it now (it is locked when the project is opened again)".

15.6.2 Unlocking, changing and removing

A protected project is locked when it is opened. Choose "Project > Protection (password)..." again and enter the project password to unlock it (valid for this session); the same menu then offers:

Option	Description
Change the protection	The protection mode of each file, the password (new password empty: unchanged)
Lock now	Protected files require the password again
Remove the protection	All files are changed back to unencrypted and no longer need the password (you are asked to confirm again)

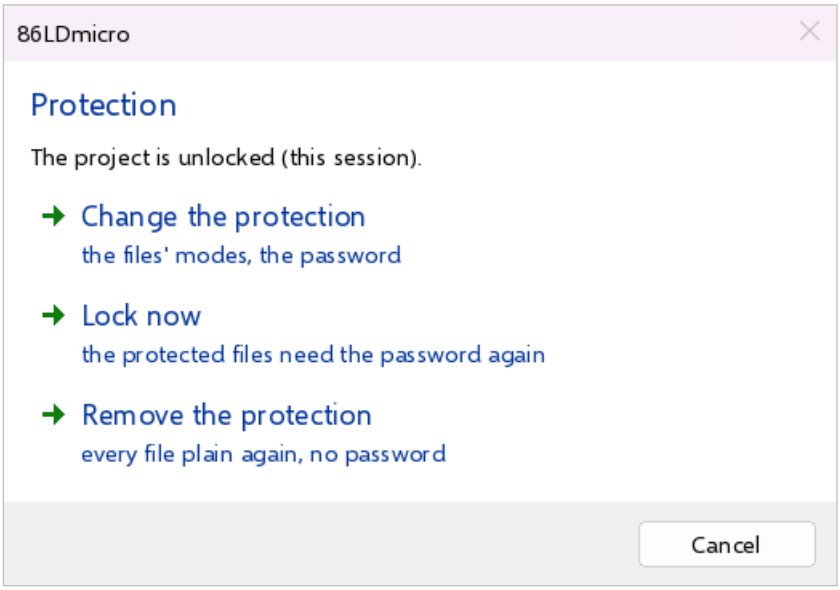


Figure 15-14 "Protection" after unlocking: Change the protection, Lock now, Remove the protection

*__Note:__ A lost password cannot be recovered; keep it safe yourself. Hidden (encrypted) files cannot be built without the password, so before delivery make sure the customer does not need to modify these programs.

Chapter 16 Frequently asked questions

This chapter collects the most common problems with installation, uploading, programs, simulation and online, 86HMI, EtherCAT and motion control. For the full list of error messages, see section 9.6.

Q1: The "Tools" menu of the 86Duino IDE has no 86LDmicro?

Make sure the 86LDmicro files are extracted to tools\LDmicro\tool of the IDE (it contains LDmicro.jar), then restart the 86Duino IDE. If Windows blocks it when it runs, the zip file was not "unblocked" before extraction: right-click the zip file > Properties > Unblock, then extract it again (section 2.3).

Q2: "The 86Duino IDE is not found" when you click "Compile" or "Upload"?

When 86LDmicro is started on its own (not opened with "Tools > 86LDmicro" in the 86Duino IDE), you must specify the location of the IDE: set the 86Duino IDE folder in "**Tools > Options...**" (section 2.6).

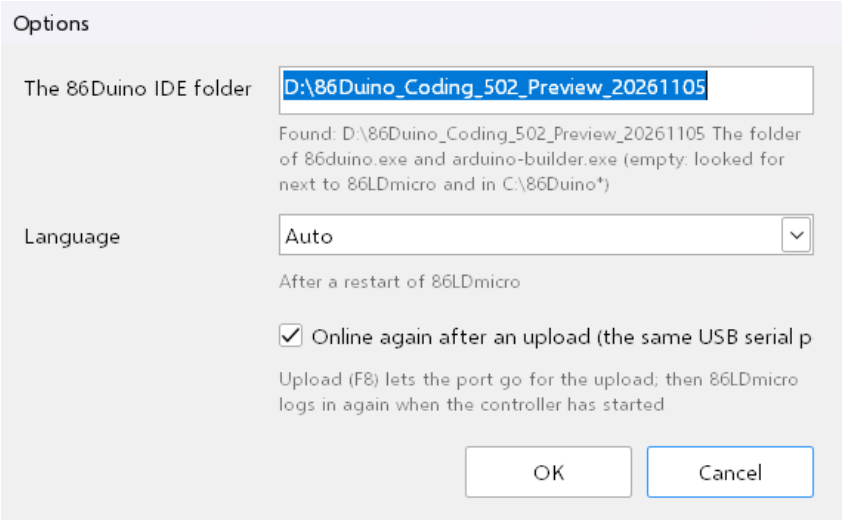


Figure 16-1 Options: the 86Duino IDE folder, Language, Online again after an upload

Q3: A project with EtherCAT or Motion86 fails to compile (sdoUpload_Begin, IKinematicsCB not found, expected class-name, etc.)?

The version of the 86Duino IDE does not match. Use the complete **86Duino Coding 502 Preview (20261105)** IDE package: it includes the new Motion86 (9 axes) and the low-level libraries it needs. Copying only the new Motion86 library into an older IDE causes compile errors.

Q4: Upload fails: could not open COMx (the serial port is in use by another program?)

The same COM port is open in another program, most often the serial monitor window of the 86Duino IDE or a terminal program. Close them, then click **Upload** (F8) again. 86LDmicro's own "Serial monitor" and the online view release the serial port automatically during the upload, so you do not need to close them manually.

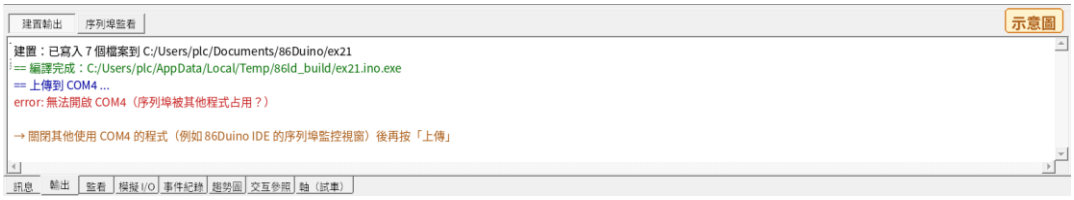


Figure 16-2 Upload failed: could not open COM4 (the serial port is in use by another program)

Message in "Output"	What to do
COMx not found	The USB cable is unplugged, the wrong port is selected or the controller is not powered: check the COM port as described in section 2.4 and select it again on the toolbar
Timeout in the middle of the upload	The USB cable or hub is unstable, or the controller is restarting: connect directly to a USB port of the computer and upload again
Compile error (.cpp file name and line number)	An error in the C/C++ user code or a library: double-click it in "Messages" to jump to the line in the .cpp86 file and fix it

Q5: The upload succeeded, but nothing happens when I press the physical button?

The most common reason is that **the variable has no I/O mapping**. The PLC program only reads and writes variables; the link between a variable and a physical point (for example `LocalIO.X00`) is set on the "I/O mapping" page (section 4.6). When you build, "Messages" shows a warning:

"Xstart is only read here: it has no I/O mapping and nothing writes it, so it is always FALSE. Map it to an input (I/O mapping, or EtherCAT > Variable mapping), or set it in a program / an 86HMI button"

1. Double-click "I/O mapping" in the project tree and make sure every input and output variable has a channel.
2. If one does not, click **Add...** and choose the variable and the channel (for example `Xstart` → `LocalIO.X00`).
3. **Upload** (F8) again.

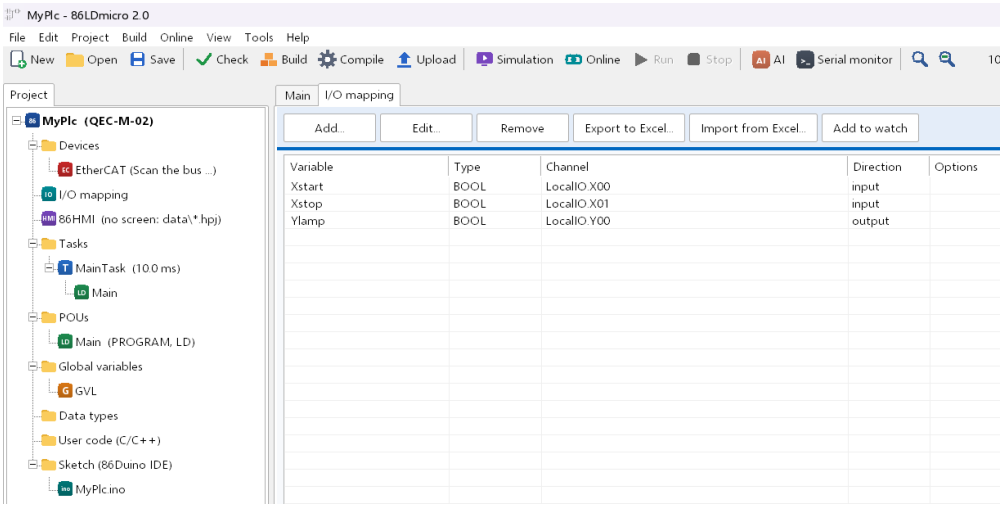


Figure 16-3 The I/O mapping page: each variable is mapped to one physical channel

Q6: An element in the ladder diagram has a red background?

The element's variable is not declared, its name is misspelled, or its type does not match (for example a REAL used as a contact). Switch to "Messages" at the bottom and double-click the error row to select the element; correct the name or declare it in the GVL / VAR, then click **Check** (Shift+F7) until there are no errors. For common messages and what to do, see section 9.6.

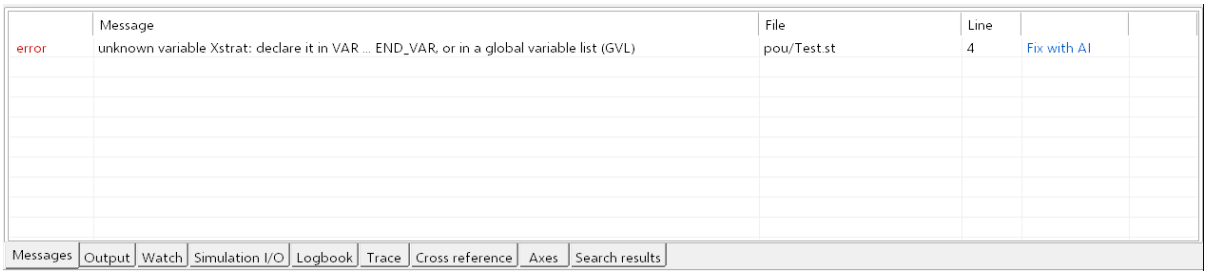


Figure 16-4 The "Messages" window: one error per row; double-click to jump to the element

Q7: Going online (Ctrl+F5) is refused, or there is never an answer?

- **Refused** (the program on the controller differs from this build): the controller is not running the program of this build of the current project; **Upload** (F8) again.
- **No answer**: make sure the online view is enabled in "Project settings" and the port is Serial (USB); the user code and the `.ino` must not use the USB `Serial`; after changing the settings, build and upload again.
- If the connection is lost for 3 seconds, the controller releases all forces automatically.

Q8: The program stops after running for a while (or stops when no computer is connected), and recovers when 86LDmicro is connected?

Usually the program keeps printing to the USB `Serial`. When the computer is not reading (86LDmicro not online, serial monitor closed), `Serial.print()` waits once the buffer is full, and the whole program (including the PLC scan) stops with it. In normal operation, do not keep printing to `Serial`; use `Serial1` to `Serial3` to communicate with external equipment; before debug printing, check whether `Serial.availableForWrite()` is large enough (section 14.6).

Q9: I forced a variable, but it has no effect?

Forced values are written at the **start** of each scan (after the inputs are read) and at the **end** (before the outputs are written). If the program writes a variable again during the scan, the program that follows reads the value the program wrote. So forcing works best on variables written only by the I/O mapping (physical inputs) and on outputs.

- Use the cross reference (Ctrl+Shift+F) to find which programs write this variable (section 9.5).
- Example: in ex30's demo mode (`gDemo = TRUE`), PlantSim rewrites `sim*` and IoMap rewrites `x*`, so forcing them is overwritten; turn the demo mode off first (`gDemo = FALSE`), then force physical inputs such as `diEntry`.

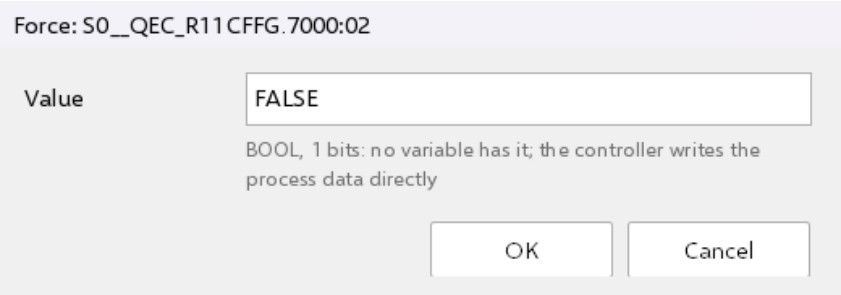


Figure 16-5 Forcing the value of a variable

Q10: During simulation I write an input variable in the Watch window, and the value changes back immediately?

A variable mapped to a physical input is determined by "**Simulation I/O**" during simulation (it is read again in every scan). Go to "Simulation I/O" at the bottom and double-click the channel to switch it ON / OFF (or enter an analog value), or use force (sections 9.2, 9.3).

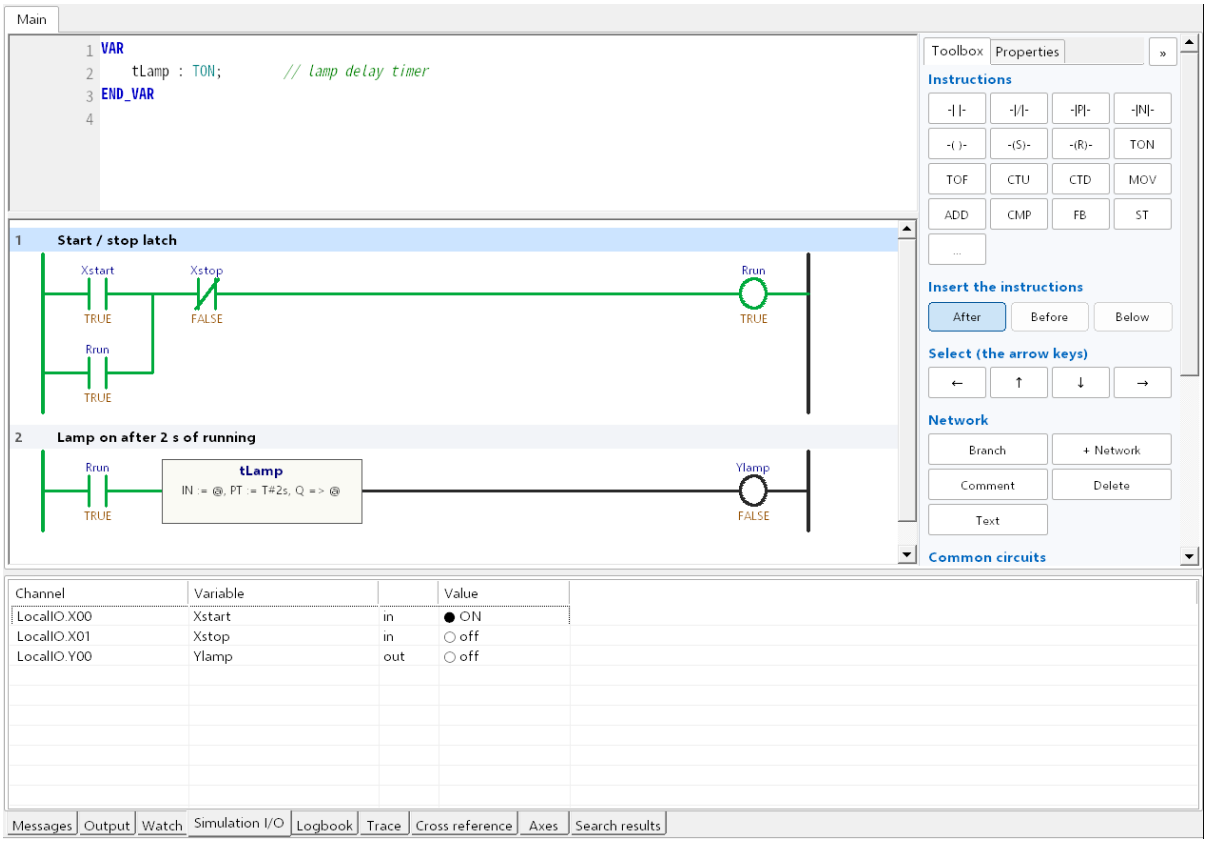


Figure 16-6 "Simulation I/O": double-click a channel to switch the simulated physical input

Q11: I press a button on the 86HMI screen, but the PLC does not respond?

- The button is bound to **a variable mapped to a physical input**: the physical input overwrites it in the next scan. Bind it to an internal variable instead (for example `Rstart`), and connect it in parallel with the physical button in the program (section 13.3).
- The binding is used as an "event", but the same variable is read in two tasks: only the first task is guaranteed to see it, and the build gives a warning. Read it in only one task, and latch it with SET if needed.
- The binding was changed during simulation: bindings take effect the next time the simulation is started (F5); end the simulation first, then press F5.
- The "Variable" column of the element in the table of the 86HMI page is empty: it is not bound yet (section 13.2).

Q12: During 86HMI simulation, the text of a ToggleButton does not switch between "not pressed" and "pressed"?

This is a **known issue** of 2.0.0: the ToggleButton's state switches correctly (and the bound variable is correct); only the text on the button in the simulation window does not change. On the real machine the program generated by 86HMI updates the text, so it is not affected.

Q13: An 86HMI language button works in simulation, but after uploading to the controller, pressing it does not switch the language?

A language button needs a switching script written in the **86HMI Editor**: add `Hmi.setLanguage(1);` to the button's Clicked event (the number is the language number), click "Code Generation" and upload again (section 13.6.2). 86LDmicro's bindings handle only variables; they do not switch the language for you.

Q14: EtherCAT does nothing (the inputs and outputs of the SubDevices do not change)?

1. Open "**Serial monitor**" (no need to go online): when everything is normal it shows `EtherCAT: running, N slaves, ...` every 5 seconds; otherwise it shows the reason for the failure.
2. `begin failed`: the EtherCAT cable is not connected to the EtherCAT port of the MainDevice, or the SubDevices are not powered.
3. `no slave`: the cable is not connected, the SubDevices are not powered, or the cable is connected to the OUT port of the SubDevice instead of IN.
4. `the ENI … cannot be used`: the ENI was not uploaded to the MainDevice or does not match the actual network: Scan again, then export and upload the ENI (Chapter 11).
5. The sketch's `.ino` must no longer call the 1.x `EVA.begin()`; remove `#include "myeva.h"` and `myeva.cpp` / `myeva.h` from the sketch.
6. When online, look at the EtherCAT messages in the "Logbook".



Figure 16-7 Logbook: the EtherCAT MainDevice start message and the start of program execution

Q15: An axis does not move, moves in the wrong direction, or the axis page shows a wrong position after homing?

Symptom	What to do
The axis does not move	Make sure it is powered on (`MC_Power.Status`), the drive has no alarm, and the PDOs of the ENI include 607Ah / 6064h; check the status with "Online" on the machine page (Chapter 12)
Wrong direction	Check "Reverse the direction" on the axis page, then build and upload again
After homing, the displayed position differs from the expected one by a fixed amount	When "Feedback shown" on the axis page uses another source (for example the encoder of the QEC-R11MP3S), homing in this version **does not realign that source**. Align it with the offset of "Feedback shown"; CSP control, in-position, home and limits still use 6064h and are not affected
The drive was moved by hand while not powered on	The position reference shifts (the Logbook shows a hint): do the homing again or use `MC_SetPosition`

Q16: An MC motion function block (MC_MoveAbsolute etc.) runs only once?

PLCopen MC function blocks start only on a **rising edge** of `Execute`. To run it again, `Execute` must first return to FALSE and then be set to TRUE. Moves of the same machine are queued in order (Buffered); a new command does not interrupt the move in progress; `MC_Stop` / `MC_Halt` stop immediately, without a deceleration ramp (sections 12.7, 12.9).

Q17: Pressing Ctrl+Space in the ST editor does not show the completion list?

In this version, completion in the ST editor **lists the candidates automatically when you type the first few characters of a variable or keyword** (showing the type and the comment); Ctrl+Space does not open the list in the ST editor. The editors for C/C++ user code and sketch files do support Ctrl+Space.

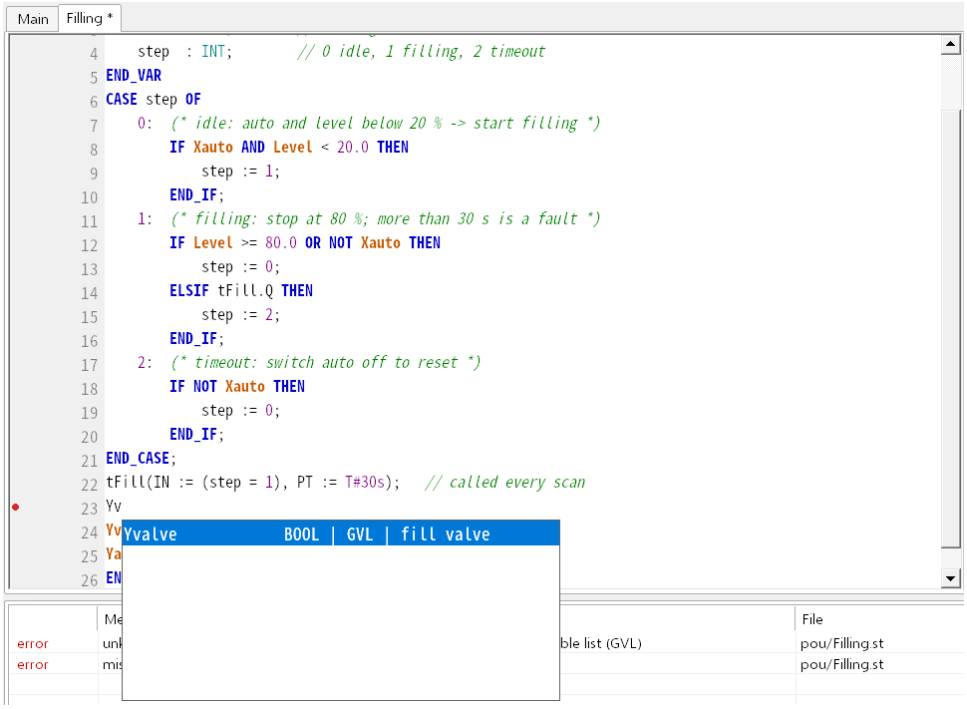


Figure 16-8 Completion in ST: typing Yv lists Yvalve (BOOL, GVL, comment)

Q18: Some programs are missing after importing PLCopen XML? Or files were changed outside, and 86LDmicro did not update?

- **FBD and IL** programs in PLCopen XML are not imported in this version; they are listed in "Output" and "Messages". Rewrite them as ST or LD in the original tool, then export again (section 15.2).
- After the project's files have been changed with an external editor, AI / MCP or version control, choose "**View > Refresh: the project again from the disk**" (Ctrl+R). If there are unsaved changes, you are asked first; stop the simulation or the online view first.

When you report a problem, please include:

- The 86LDmicro version (`tool\VERSION.txt`, for example `2.0.0 2026-11-11`) and the hardware combination (controller, modules, drive models).
- The steps you took and screenshots; the build messages in the "Output" window and the contents of the "Logbook".
- If necessary, a compressed file of the project folder (`data\86ld` in the sketch). For contact information, see "Contact us" at the end of this manual.

Appendix A Comparison with LDmicro 1.1

Users familiar with LDmicro 1.1 (user manual v1.6.3) can use the table below to find how things are done in 86LDmicro 2.0. For importing 1.1 projects, see section 15.1.

Item	LDmicro 1.1	86LDmicro 2.0
Starting	Start LDmicro from the "Tools" menu of 86Duino Coding	"Tools > 86LDmicro" in the 86Duino IDE, or the 86LDmicro button on the IDE toolbar (section 2.5)
Program files	One `.ld` file (`data\myLD.ld`)	Project folder `data\86ld`: `project.86ld`, `pou`, `gvl`, `dut`, `usercode` (section 4.1)
Programming languages	Ladder diagram (with ST blocks)	LD, ST, SFC (IEC 61131-3) and C/C++ user code
Choosing the MCU / board	Settings → General (PLC Configuration)	The controller in "Project > Settings" (default QEC-M-02)
Scan cycle	PLC Cycle Time in Settings → General	The interval of a "Task" (default MainTask 10 ms); there can be several tasks (section 4.3)
COM port and baud rate	COM Mapping, Baudrate in Settings → General	COM devices in "Devices" (section 4.4)
Modbus	Settings → Modbus, variable prefixes M / I / H	modbus and modbus-server in "Devices", and the I/O mapping
EtherCAT	Settings → Virtual Arduino + 86EVA Configurator (`myeva.h` / `myeva.cpp`, `EVA.begin()`)	EtherCAT Configurator → export the ENI → 86LDmicro reads the ENI back and the SubDevices become devices (Chapter 11); `.ino` no longer calls `EVA.begin()`
Variables and types	The prefix determines the kind (X input, Y output, R internal, T timer, C counter); double-click in the IO List to change the size	Declare IEC types (BOOL, INT, REAL, TIME…) in the GVL / VAR; the prefix is only a naming convention (Appendix C)
I/O pins	Double-click the Pin Name column of the IO List to choose a pin	"I/O mapping" page: variable ↔ channel (for example `LocalIO.X00`), including inversion, filter and scaling (section 4.6)
Timers, counters	TON, TOF, CTU… instructions (counting in cycles)	IEC function blocks TON, TOF, TP, CTU… (`CALL`, timed in real time); the 1.x instructions are still available (Chapter 7)
Generating C code	Code Generation button	**Build** (Ctrl+B)
Generated files	`myplc.cpp` / `myplc.h`, `myeva.h` / `myeva.cpp`, `ladder.h`	`plc86.cpp` / `plc86.h`, `plc86_rt.*`, `plc86_ecat.*` (section 14.5)
Entry points in the `.ino`	`EVA.begin()` → `setupPlc()` / `loopPlc()`	`plc86_setup()` / `plc86_loop()` (added automatically by the build)
Compiling and uploading	Go back to 86Duino Coding and click "Upload"	86LDmicro **Compile** (F7) and **Upload** (F8) directly (section 10.1)
Simulation	F7 simulation mode, F8 real-time simulation, Space single cycle	**Simulate** (F5), speed x1 / x10; Simulation I/O, 86HMI simulation window (Chapter 9)
Online monitoring	Online monitoring (ldmon)	**Online** (Ctrl+F5): power flow, watch, write, force, RUN / STOP (section 10.4)
86HMI	86HMI binding dialog box	86HMI page and binding form (the same concept); one widget can have several bindings (Chapter 13)
User C code	User Code (global / setup / beforescan / afterscan / loop / ethercat)	Hook functions in user code files `.cpp86`; access variables with `GVL()` (Chapter 14)
AI	An AI window opens on compile errors	AI Assistant tab, "Fix with AI" in "Messages", MCP (sections 15.3, 15.4)

Item	LDmicro 1.1	86LDmicro 2.0
Help	F1 or File → Manual... (built-in manual.txt)	The "Help" menu opens the online documentation site; `tool\docs` is the offline version

Appendix B Keyboard shortcuts

The following are all keyboard shortcuts of 86LDmicro 2.0.0 (accelerator keys of the main window and keys of each editor).

B.1 Main window (any tab)

Key	Function	Menu
Ctrl+N	New project	File
Ctrl+O	Open project	File
Ctrl+S	Save	File
Ctrl+Z / Ctrl+Y	Undo / Redo	Edit
Ctrl+X / Ctrl+C / Ctrl+V	Cut / Copy / Paste	Edit
Del	Delete	Edit
Ctrl+A	Select all	Edit
Ctrl+F	Find in this tab	Edit
F3 / Shift+F3	Find next / Find previous	Edit
Ctrl+Shift+F	Find a variable in the project (Cross reference)	Edit
Ctrl+Shift+H	Find in the project (Search results)	Edit
Shift+F7	Check	Build
Ctrl+B	Build (write the sketch)	Build
F7	Compile (86Duino)	Build
F8	Upload to the controller	Build
F5	Simulate (start / end)	Online
Ctrl+F5	Login to the controller (online)	Online
F9 / Shift+F9	RUN / STOP	Online
Ctrl++ (or numeric keypad +)	Zoom in	View
Ctrl+- (or numeric keypad -)	Zoom out	View
Ctrl+0 (or numeric keypad 0)	100%	View
Ctrl+mouse wheel	Zoom in / Zoom out	—
Ctrl+\	Side by side: move the tab in front to the right / end it	View
Ctrl+R	Refresh: the project again from the disk	View
Ctrl+W, middle mouse button	Close the tab	—
Ctrl+Tab / Ctrl+Shift+Tab	Next / previous tab	—

B.2 Find bar and project tree

Key	Function
Enter or F3	Find bar: next
Shift+Enter or Shift+F3	Find bar: previous
Esc	Close the find bar
Ctrl++ / Ctrl+- (when the project tree has the focus)	Expand all / Collapse all
Enter (Search results)	Jump to the selected place (same as double-click)

B.3 Ladder diagram editor

Key	Function
← / → / ↑ / ↓	Move the selection
Ctrl+← / Ctrl+→	Move the selected element left / right
Ctrl+↑ / Ctrl+↓ (when a network is selected)	Move the network up / down
Home / End	Select the first / last network
Enter, double-click	Edit the parameters of the selected element; when a network is selected, edit the network title and comment
Del, Backspace	Delete the selected element or network
Ctrl+Enter / Ctrl+Shift+Enter	Insert a network after / before the selected network
Ctrl+C / Ctrl+X / Ctrl+V	Copy / Cut / Paste
Ctrl+Z / Ctrl+Y (or Ctrl+Shift+Z)	Undo / Redo
Ctrl+T	Text mode (.lad text)
Shift+instruction key	Insert before the selected element
B	Add a normally open contact in parallel below the selected element

Keys for inserting instructions (inserted after the selected element, or according to "Insert the instructions" in the side panel):

Key	Instruction	Key	Instruction	Key	Instruction
C	Normally open contact NO	S	Set coil SET	D	CTD
N	Normally closed contact NC	R	Reset coil RST	M	MOV
P	Positive transition contact PT	O	TON	A	ADD
L	Coil OUT	F	TOF	E	Condition contact CMP
K	Function block CALL	U	CTU	T	ST statements

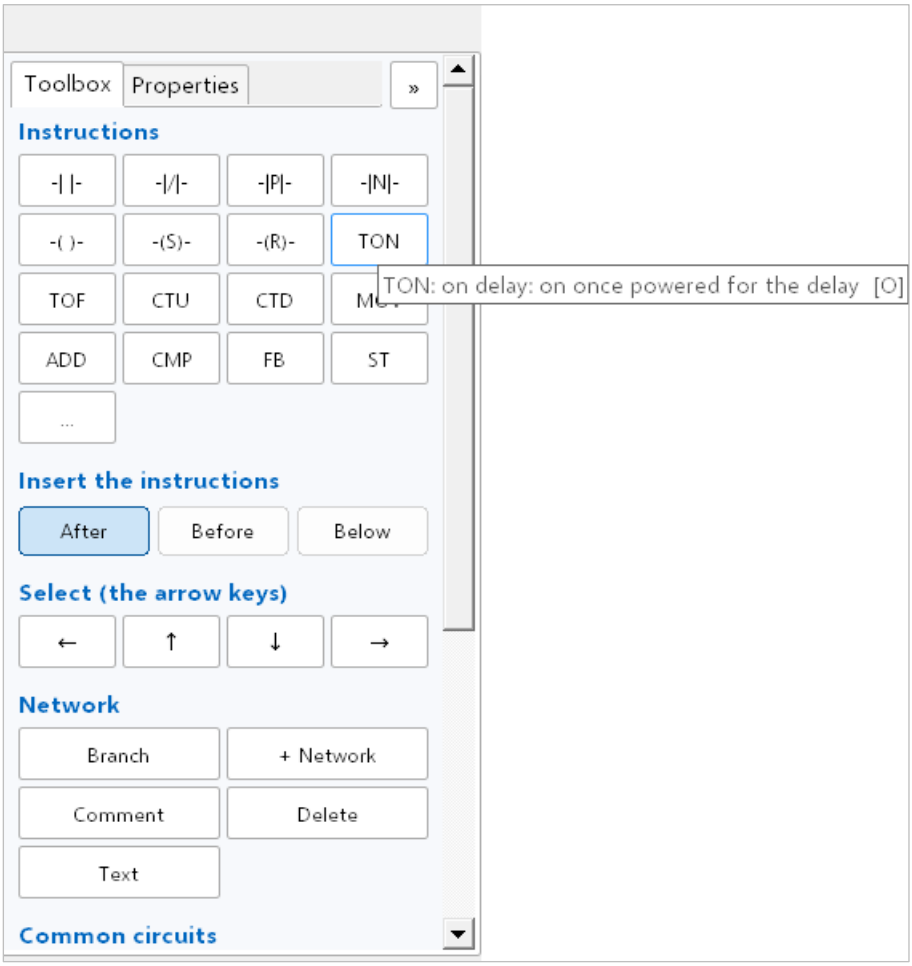


Figure B-1 "Toolbox" in the side panel: hovering the mouse over an instruction button shows its description and key

B.4 SFC editor

Key	Function
Insert	+ Step: add a transition and a step after the selection
Enter, double-click	Edit the selected step or transition
Del	Delete (a step together with its transition)
↑ / ↓	Previous / next item in the same sequence
← / →	Adjacent branch
Ctrl+Z / Ctrl+Y (or Ctrl+Shift+Z)	Undo / Redo
Ctrl+T	Text mode (.sfc text)

B.5 Code editors (ST, GVL, data types, C/C++, sketch)

Key	Function
Ctrl+T	Auto-format (adjusts only the indentation at the start of lines and extra blank lines; Ctrl+Z undoes it)
Ctrl+Space	Open the completion list (C/C++ editors). ST editor: the list appears automatically when you type the first few characters; in this version Ctrl+Space does not open the list
↑ / ↓, PgUp / PgDn	Move in the completion list
Enter or Tab	Completion list: insert the selected item
Esc	Close the completion list or the parameter hint
Tab / Shift+Tab	Indent 4 spaces / unindent
Enter	New line keeping the indentation (one more level after {, THEN, DO and so on)
Ctrl+Z / Ctrl+Y	Undo / Redo

B.6 AI assistant

Key	Function
Ctrl+Enter	Send the question
Esc	Close "AI assistant: settings"

Appendix C Variable naming and types

C.1 Naming conventions

In 86LDmicro 2.0 the type of a variable is determined by its declaration, and names are not restricted. For readability and maintainability, we recommend keeping the prefix conventions of LDmicro 1.1 (all examples in this manual are named this way):

Prefix	Use	Declaration example	Description
X	Input	`Xstart : BOOL;`	Mapped to a physical input (I/O mapping)
Y	Output	`Ylamp : BOOL;`	Mapped to a physical output
R	Internal flag	`Rrun : BOOL;`	Latches, HMI buttons, step flags
T	Timer instance	`Tdelay : TON;`	One instance per timer
C	Counter instance	`Cparts : CTU;`	One instance per counter
M	Numeric value	`Mposition : INT;`	Positions, set values, measured values

C.2 Naming rules

- Only letters, digits and the underscore _ may be used, and a name cannot begin with a digit; names are not case-sensitive (`Xstart` and `XSTART` are the same variable).
- A name cannot be the same as an IEC reserved word (`TON`, `IF`, `END_VAR`…); when importing from 1.1, such names get `_v` appended. Names beginning with __ are reserved for internal use by 86LDmicro.
- Variables declared in the **GVL** are visible to all POUs, 86HMI and C/C++; variables declared in the `VAR` of a POU can be used only in that POU (other POUs read them as `ProgramName.Variable`).
- The same name can be declared only once in the GVL.

C.3 Data types and ranges

Type	Size	Range	C type
BOOL	1 bit	FALSE / TRUE	`bool`
SINT	8 bits	-128 to 127	`int8_t`
USINT	8 bits	0 to 255	`uint8_t`
INT	16 bits	-32,768 to 32,767	`int16_t`
UINT	16 bits	0 to 65,535	`uint16_t`
DINT	32 bits	-2,147,483,648 to 2,147,483,647	`int32_t`
UDINT	32 bits	0 to 4,294,967,295	`uint32_t`
LINT	64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	`int64_t`
ULINT	64 bits	0 to 18,446,744,073,709,551,615	`uint64_t`
BYTE / WORD / DWORD / LWORD	8 / 16 / 32 / 64 bits	Bit string (unsigned)	`uint8_t` to `uint64_t`
REAL	32 bits	IEEE 754 single precision, about ±3.4E38, about 7 significant digits	`float`
LREAL	64 bits	IEEE 754 double precision, about ±1.8E308, about 15 significant digits	`double`
TIME	32 bits	Milliseconds, ±2,147,483,647 ms (about ±24.8 days); the largest constant is T#24d20h31m23s647ms	`int32_t`
STRING / STRING[n]	n+1 bytes	String, 80 characters by default, at most 254 characters	`char[n+1]`
Enumeration (TYPE … (A, B, C))	32 bits	The declared values	`int32_t`

Arrays (ARRAY[1..5] OF INT), structures (STRUCT) and enumerations are defined in "Data types" (section 4.5). The IEC runtime of 86LDmicro 2.0.0 does not yet support WSTRING, DATE, TIME_OF_DAY (TOD), DATE_AND_TIME (DT), POINTER or REF_TO.

***Note:** When a value is outside the range of its type, Check reports an error such as "the value … does not fit INT"; when a count or a sum can exceed 32,767, use DINT.

C.4 Constants and how to write TIME

Kind	Notation	Value
TIME	`T#500ms`, `T#2s`, `T#1.5s`, `T#1m30s`, `T#1h30m`, `T#1d`	The units d, h, m, s and ms can be combined; numbers can have a decimal point and underscores
TIME (full notation)	`TIME#2m15s`	Same as `T#2m15s`
Hexadecimal, binary, octal	`16#FFFF`, `2#1010_1010`, `8#777`	65535, 170, 511
Typed	`INT#-5`, `REAL#1.5`	A constant of that type
Real number	`1.5`, `1.5E3`	1.5, 1500.0
String	`'hello'`	Single quotes
Boolean	`TRUE`, `FALSE`	

To convert between TIME and integers, use `TIME_TO_DINT(t)` (milliseconds) and `DINT_TO_TIME(ms)`; compare a time with a time, for example `IF Fill.T >= T#5s THEN`. Declaration examples (all of the notations above have been verified by check and simulation):

```
VAR
    tFill   : TIME := T#1m30s;     (* 90000 ms *)
    tPulse  : TIME := T#1.5s;      (* T#1s500ms *)
    Mask    : WORD := 16#FFFF;
    Pattern : BYTE := 2#1010_1010;
    Total   : LINT;
    Ratio   : LREAL := 1.5E3;
    Msg     : STRING[20] := 'hello';
END_VAR
```

Contact us

If you run into any technical problem while following this manual, or have purchasing or customization needs for QEC series products, please contact us through the following channels:

Item	Contact
Sales / purchasing	info@qec.tw (QEC product line)
Technical support / R&D	info@icop.com.tw (ICOP Technology Inc.)
Online documentation	https://www.qec.tw/86duino/86ldmicro/

The "Help" menu of 86LDmicro opens the online documentation site directly; for the latest user manual and documentation, refer to the website.