My Cursor setup for OPCUA Pipe Gateway

Guido Brugnara — Leader.IT · **Cursor Pro**

Cursor MeetUp Trento · 11 June 2026 · Afliant

luma.com/7rcm3za3

The project (30 seconds)

`opcua_pipe_gateway` — OPC-UA client via STDIN/STDOUT (*"curl for the PLC"*)

- Dual implementation: **Perl** (open62541) + **Python** (python-opcua)
- Own **test servers** in both languages → interoperability matrix
- Validated on **KEPServerEX V6**, then packaged for deployment

leader.it/Portal/OpcUApipeGateway

My Cursor environment

Item	Choice
Edition	Cursor **Pro** (Agent with full tool access)
Dev machine	Desktop — main workspace
Meetup laptop	14" e-book (Intel N5000) — project copy for demo
Languages	Perl, Python, Bash, C — heterogeneous codebase

Cursor as **pair programmer**, not autopilot — I keep industrial decisions.

Custom instructions (User Rules)

Global rules that steer every session:

- **Scripts:** prefer **Perl** or **Bash**; inline comments in **English**
- **Docs:** POD format — Italian (`LEGGIMI.pod`) + English (`README.pod`)
- **Chat:** Italian with the AI; deliverables bilingual where needed
- **Code changes:** minimal scope — match existing style, no over-engineering
- **Transparency:** AI-assisted work acknowledged in source headers

*Cursor **User Rules** — persistent context, no repetition each chat.*

Workspace setup

Multi-root workspace (`Test_OPC_UA.code-workspace`):

```
folders:
  .                              ← dev & test environment
  ~/SVN/opcua_pipe_gateway       ← public distribution trunk
```

Chat context: `@file` · `@folder` · `@workspace`

Docs: `SOLUZIONE_SICURA_MULTI_ROOT.md`, `PRESERVA_HISTORY_CURSOR.md`

Workflow: Plan → Agent → Test

1. **Plan mode —** architecture, slide outlines, cost analysis (no edits)
2. **Agent mode —** code, docs, scripts; AI runs terminal & reads logs
3. **Human —** test scenarios, certificates, security policy, sign-off

```
Parallel Client + Server → 2×2 tests → AI log analysis
  → KEPServerEX validation → zero-error test suite
```

Prompt examples (real) — 1/2

Templates from real work — not a live demo. One goal · one `@file` · focused chat.

Porting

> Create a Python equivalent of `@opcua_pipe_gateway.pl` — same CLI, exit codes, and I/O format.

Cross-language QA

> Run `@test_kepserverex_v6_demo.pl` and `.py` on the same server; compare outputs and fix differences.

Industrial debug

> Analyse `@OPC_diagnostic.8.log.txt` — why does KEPServerEX reject the client certificate?

Prompt examples (real) — 2/2

Same pattern: one focused goal per chat.

Documentation

> Write POD documentation in Italian for `@opcua_pipe_gateway.pl` following existing style.

This meetup

> Plan a 10-minute Marp slide deck: Cursor setup first, OPC-UA project as case study.

How Cursor helped (concrete)

- **Porting:** Perl client → equivalent Python (same CLI contract)
- **Cross-language QA:** 4 combinations reveal bugs a single stack hides
- **Industrial debug:** OPC logs, certs, UserTokenPolicy fixes
- **Documentation:** POD, LEGGIMI, wiki, cost/license analysis
- **This deck:** Marp markdown → PDF via **Marp for VS Code**

Extensions & MCP

- **Marp for VS Code** (`marp-team.marp-vscode`) — slides → PDF
- **Perl** language support — POD, `.pl` navigation
- No project-specific **MCP** config — **Agent tools** (terminal, search) did the work
- **Browser MCP** when external OPC-UA / vendor docs are needed

Simple setup: User Rules + workspace + Agent.

Takeaways · Questions?

- **User Rules** + **multi-root workspace** + **Plan → Agent**
- **Test matrix + AI —** reusable beyond OPC-UA
- **Human-in-the-loop** for industrial / security domains

Open source: Portal · SVN `opcua_pipe_gateway/tag/`

Slides in **English** · Talk in **Italian —** thank you!