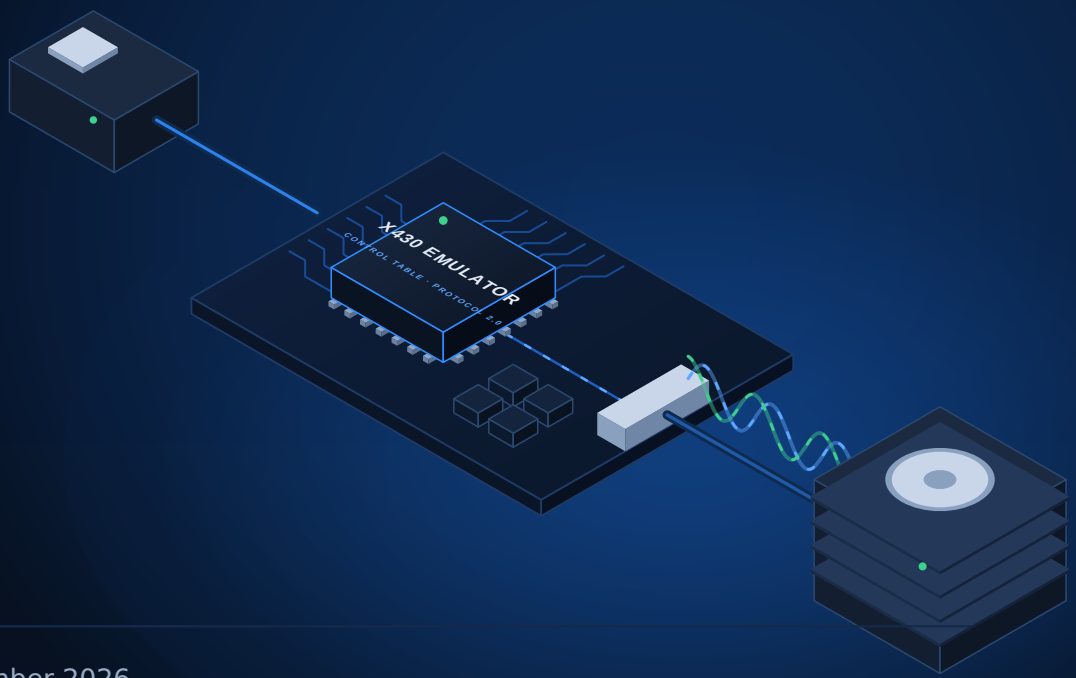
SystemFromScratch

OPEN-SOURCE EMBEDDED LIBRARY

X430 Emulator

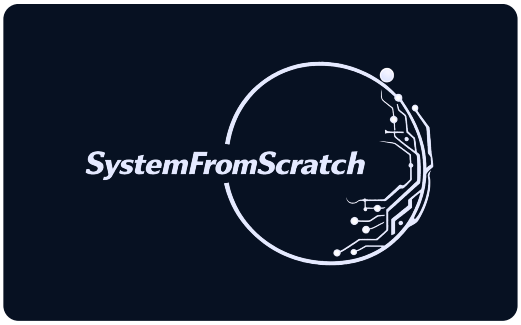
DYNAMIXEL X430 Control Table Emulator in portable C

USER MANUAL



Version 1.2.0 · November 2026
Anton Vynohradov
systemfromscratch.com

Imprint



X430 Emulator User Manual
Library version 1.2.0
Edition: November 2026

Author
Anton Vynohradov
SystemFromScratch® — systemfromscratch.com
GitHub: github.com/AntonVynohradov
LinkedIn: linkedin.com/in/vynohradov
ORCID: orcid.org/0009-0005-6965-0911
E-mail: contact@systemfromscratch.com

Online
Documentation: https://antonvynohradov.github.io/DYNAMIXEL-X430-Control-Table-Emulator/
Source code: https://github.com/AntonVynohradov/DYNAMIXEL-X430-Control-Table-Emulator

Copyright © 2026 Anton Vynohradov.

The X430 Emulator library and this manual are licensed under the Apache License, Version 2.0 (the “License”); you may not use these files except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 and in Appendix B. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

SystemFromScratch® and the SystemFromScratch® logo are registered trademarks. The Apache License 2.0 grants no rights to use them; they may not be used to endorse or promote products derived from X430 Emulator without prior written permission.

DYNAMIXEL and ROBOTIS are trademarks of ROBOTIS Co., Ltd. STM32 is a registered trademark of STMicroelectronics International N.V. Product names are used only to identify the emulated devices and the test hardware; DYNAMIXEL-X430-Control-Table-Emulator is not affiliated with or endorsed by ROBOTIS or STMicroelectronics. All other trademarks are the property of their respective owners.

This manual is generated from the same sources as the online documentation. The information in it was checked against the library version stated above; later versions may differ.

Contents

1 Introduction 6
- 1.1 What the library does 6
- 1.2 What the library does not do 6
- 1.3 Who it is for 7
- 1.4 Key properties 7
- 1.5 Terminology 7
- 1.6 Conventions in this documentation 8

2 Getting Started 9
- 2.1 Get the library 9
- 2.2 Add it to your project 9
- 2.3 Prerequisites for development 9
- 2.4 Build with presets 10
- 2.5 Unit tests and coverage 10
- 2.6 Static analysis 10
- 2.7 Versioning and releases 11
- 2.8 Measured figures 11
- 2.9 Documentation 11

3 How It Works 12
- 3.1 Modules 12
- 3.2 Execution contexts 13
- 3.3 States 14
- 3.4 Memory model 14
- 3.5 Error handling 15
- 3.6 Threading and reentrancy 15
- 3.7 Design decisions 16

4 Using the Library 17
- 4.1 Setting up a bus and its devices 17
- 4.2 Reacting to the host: hooks 18
- 4.3 Keeping the live registers current 19
- 4.4 Faults 19
- 4.5 Using the simulator 19
- 4.6 Return values 20
- 4.7 Common mistakes 20

5 Extending the Library 21
- 5.1 Where to extend 21
- 5.2 Several kinds of devices on one line 21
- 5.3 Adding a model 22
- 5.4 Contributing to the library 22

6 Configuration 23
- 6.1 Build-time options 23
- 6.2 Run-time configuration 25
- 6.3 Memory footprint 26
- 6.4 Stack usage 27

6.5 CMake options ... 28

7 X430 Emulation ... 30
7.1 The Control Table model ... 30
7.2 Addressing and status packets ... 31
7.3 Writes ... 31
7.4 Indirect Address and Data ... 32
7.5 Lifecycle ... 32
7.6 Group instructions ... 33
7.7 Persistent blocks ... 34
7.8 The simulator ... 34
7.9 Limits of the emulation ... 36

8 Integration Guide ... 37
8.1 Timing ... 37
8.2 Bare metal ... 37
8.3 RTOS ... 38
8.4 Events ... 39
8.5 Persistent storage ... 39
8.6 Adding the library to a CMake build ... 40
8.7 Host tools ... 40
8.8 Bring-up checklist ... 44

9 API Reference ... 46
9.1 Version — x430.h ... 46
9.2 Definitions — x430_defs.h ... 46
9.3 Types — x430_types.h ... 47
9.4 Variables — x430.h ... 48
9.5 Functions — x430.h ... 48
9.6 Return codes summary ... 59

10 MISRA C:2012 compliance ... 60
10.1 Checking ... 60
10.2 Current status ... 60
10.3 Manual review notes ... 60
10.4 Deviation register ... 61

Author & Company ... 62
Author ... 62
SystemFromScratch® ... 63
Contact ... 63
Support and feedback ... 64
Trademarks and acknowledgements ... 64

Appendix A Changelog ... 65
A.1 1.2.0 - 2026-11-22 ... 65
A.2 1.1.0 - 2026-11-15 ... 65
A.3 1.0.0 - 2026-11-14 ... 66

Appendix B License ... 68
B.1 What the license allows ... 68
B.2 What it requires ... 68
B.3 What it does not give ... 68

B.4 Trademarks 69
B.5 Full license text 69

CHAPTER 1

Introduction

DYNAMIXEL-X430-Control-Table-Emulator (the *X430 emulator*) is a C11 library that makes a microcontroller answer on a DYNAMIXEL bus like a ROBOTIS X430 actuator. It decodes Protocol 2.0 instruction packets (with DXL2-C (github.com/AntonVynohradov/DXL2-C)), keeps the Control Table of the selected X430 model, enforces the rules a real actuator enforces and answers with status packets. The motor behind it can be anything — a stepper, a BLDC, a simulation: the library reaches it only through hooks, so the host tools of the DYNAMIXEL ecosystem work unchanged. DynamixelSDK (end to end) and DYNAMIXEL Wizard 2.0 (on hardware) are verified; ROS 2 `dynamixel_hardware_interface` uses the same instructions (points to check (p. 40)).

1.1 What the library does

- **The X430 Control Table.** One of 17 models per build, generated from YAML data that the test suite compares with the ROBOTIS e-Manual tables: addresses, sizes, access, initial values and ranges, limit registers, the EEPROM lock while the torque is on.
- **Every instruction a host uses.** PING, READ, WRITE, REG_WRITE, ACTION, FACTORY_RESET, REBOOT, CLEAR, BACKUP, SYNC_READ/WRITE, BULK_READ/WRITE, FAST_SYNC_READ, FAST_BULK_READ, with Status Return Level, Return Delay Time, Secondary ID and broadcast.
- **The rules around the motor.** Indirect Address/Data, Bus Watchdog, hardware error shutdown and the alert bit, Startup Configuration, Drive Mode "torque on by goal update", Goal Position := Present Position on torque on, persistence of the EEPROM area and of the Control Table backup with a header and a CRC.
- **A simulator** (optional) that moves the shaft with the X430 profiles, for tests and for setting up ROS 2 or the Wizard before the hardware exists.

1.2 What the library does not do

Concern	Why it is not in the library
UART, DMA and baud rate drivers	Hardware-specific; the application moves the received bytes in and writes the status packets out in a callback.
Motion and current control	The motor is the application's; the write hook receives each accepted goal, limit and mode.
Flash or EEPROM drivers	The library reads and writes opaque blocks in parts through `nvm_read` / `nvm_write`; where they live is up to the application.
Threads, queues, interrupts	The library defines the contexts and takes an optional lock; the application decides where they run. See the integration guide.
Protocol 1.0, other DYNAMIXEL series	Only Protocol 2.0 and the X430 family are emulated. Another family needs its model data.

1.3 Who it is for

- **Firmware of custom actuators** (stepper or BLDC joints, grippers) that should appear as DYNAMIXEL servos to existing host software.
- **Test benches and CI** that need X430 devices without hardware (the simulator and the POSIX host example serve them on a pseudo-terminal).
- **Projects that must be reviewed or certified:** MISRA C:2012, 100 % coverage, no heap.

1.4 Key properties

Property	Value
Language	C11, no warnings with `-Wall -Wextra -Wpedantic -Wconversion`
Dependencies	`<stddef.h>`, `<stdint.h>`, `<stdbool.h>`; DXL2-C 2.0.0 (fetched by CMake)
Memory	No heap, no global or static mutable data (`.data` and `.bss` are empty)
RAM	384 B per device, 1,664 B per bus, 32 B per simulator (Cortex-M4, default options; details)
Flash	10,119 B with the model table, plus 842 B of DXL2-C (Cortex-M4, `-Os`, all features)
Stack	424 B worst case, `x430_bus_receive()`, with any packet size (Cortex-M4 `-Os`, details)
Coding standard	MISRA C:2012, 0 Cppcheck 2.13 addon violations, 2 deviations
Tests	100 % line and branch coverage of `src/` in both unit test presets (137 and 104 GoogleTest cases); model data against the e-Manual; 52 checks end to end with the official DynamixelSDK
Thread safety	One bus context per line, an optional lock for the live register updates; see Threading
License	Apache License 2.0

The library was verified on hardware in a stepper joint firmware (an STM32G431 board, Zephyr) with DYNAMIXEL Wizard 2.0; the screenshots in the integration guide (p. 40) come from it.

1.5 Terminology

Host

The program that sends instruction packets: DynamixelSDK, DYNAMIXEL Wizard, ROS 2.

Bus

One serial line (TTL or RS-485) and the devices this firmware hosts on it (`struct x430_bus`).

Device

One emulated actuator with one DYNAMIXEL ID (`struct x430_device`). A dual-axis 2XL430/2XC430 is two devices.

Control Table

The register image of a device: EEPROM area (addresses 0–63, persistent, locked while the torque is on) and RAM area (64–661).

Hook

An application function the device calls: write, read, event, nvm_read, nvm_write.

Live register

A read-only RAM register the application keeps current: Present Position, Present Velocity, Moving and the others.

1.6 Conventions in this documentation

- Registers are written with their e-Manual name and address: *Goal Position (116)*.
- Protocol errors are the `DXL2_ERR_*` codes of DXL2-C (Result Fail, Instruction, CRC, Data Range, Data Length, Data Limit, Access); library status codes are `X430_OK` and `X430_ERR_*`.
- The example model is the XM430-W350, the default of the build.

CHAPTER 2

Getting Started

This chapter shows how to get the X430 emulator, add it to your project, and build and test it.

2.1 Get the library

```
git clone https://github.com/AntonVynohradov/DYNAMIXEL-X430-Control-Table-Emulator.git
```

Each release (github.com/AntonVynohradov/DYNAMIXEL-X430-Control-Table-Emulator/releases) provides the source code, a prebuilt static library for Linux x86-64 (default options: XM430-W350, every feature) with its headers, the MISRA report and the PDF user manual. For an MCU, and for any other model, build the library from source: the Control Table is generated for the model chosen at build time.

2.2 Add it to your project

CMake.

```
set(X430_MODEL xm430_w350)        # X430_KCONFIG ON for Kconfig fragments, see
Configuration
include(path/to/DYNAMIXEL-X430-Control-Table-Emulator/x430.cmake)
target_link_libraries(app PRIVATE x430)
```

`x430.cmake` generates the Control Table of the model into the build directory, fetches DXL2-C 2.0.0 with FetchContent (or takes a local checkout given in `FETCHCONTENT_SOURCE_DIR_DXL2`) and defines the static library `x430`, which links `dxl2`.

Any other build system. Run `python3 tools/x430_gen_model.py --models-dir models --model xm430_w350 --out-dir gen`, compile `src/*.c`, `gen/x430_model_table.c` and the two sources of DXL2-C, and add the repository root, `inc/`, `gen/` and the DXL2-C folders to the include path. Pass the same `X430_*` definitions to the library and the application, and define `DXL2_MAX_PACKET_SIZE=768U` for the library, DXL2-C and the application alike (`x430_config.h` has no default and stops the build without it).

As a submodule with tests. A project that sets `UNIT_TESTS=ON` also builds the unit tests of this library (`x430_gtest`, CTest label `x430`); `COMPILE_DEFINITIONS` and `COMPILE_OPTIONS` of the project reach the library as well.

2.3 Prerequisites for development

- C11 compiler, CMake 3.20+, Python 3 (the model generator needs only the standard library; PyYAML is used when installed)
- Optional: GoogleTest and gcovr, Cppcheck 2.13, clang-format 18, Kconfiglib, Python 3 with `requirements-dev.txt` (MkDocs, WeasyPrint, ziamath), Doxygen, Pango, `arm-none-eabi-gcc` for the measurements

2.4 Build with presets

Preset	Purpose
`X430[Debug]`	Static library with debug information (XM430-W350, without Kconfig)
`X430[Release]`	Optimized static library
`X430[UnitTests]`	GoogleTest suite with coverage, every feature; target `coverage`
`X430[UnitTestsMinimal]`	The same with every optional feature left out
`X430[Kconfig]`	Configuration with Kconfiglib and `configs/defconfig`; target `menuconfig`
`X430[Examples]`	`x430_host_sim`, the POSIX host simulator

```
cmake --preset "X430[Debug]"
cmake --build "build/X430[Debug]"
```

Another model: `cmake --preset "X430[Debug]" -DX430_MODEL=xl430_w250`.

2.5 Unit tests and coverage

```
cmake --preset "X430[UnitTests]"
cmake --build "build/X430[UnitTests]" --target coverage
```

The `coverage` target runs the tests and fails below `X430_COVERAGE_MIN` (100 %) line or branch coverage of `src/`. Both unit test presets reach 100 %. The tests are in `tests/unit/` (one file per area: API contract, protocol, group instructions, lifecycle, device rules, simulator); `tests/compile/x430_api_check.c` compiles the public API alone, as an application sees it. CTest also runs `tests/models`, which compares the model data with the e-Manual tables.

End to end with the official DynamixelSDK (Linux):

```
cmake --preset "X430[Examples]" && cmake --build "build/X430[Examples]"
tests/sdk/run_sdk_check.sh "build/X430[Examples]"
```

The script clones the ROBOTIS DynamixelSDK, serves three simulated devices on a pseudo-terminal and runs 52 checks: PING, READ/WRITE, Sync, Bulk and Fast reads, Indirect Address, REBOOT, FACTORY_RESET, CLEAR.

2.6 Static analysis

Target	Tool
`cppcheck-analyze`	Cppcheck, all checks
`misra-analyze`	Cppcheck MISRA C:2012 addon; fails on any violation
`codeql-analyze`	CodeQL `cpp-queries`

Check MISRA with the Cppcheck version of CI (2.13, Ubuntu 24.04): newer versions and Windows hosts report other findings.

Formatting

```
clang-format -i x430.h inc/*.h src/*.c src/*.h tests/unit/*.cpp tests/unit/*.h \
    tests/compile/*.c examples/host_sim/*.c
```

CI checks every C/C++ file with `clang-format` 18.

2.7 Versioning and releases

The X430 emulator follows Semantic Versioning (semver.org): the public API (`x430.h` and `inc/`) is stable within a major version. The version comes only from the latest `vX.Y.Z` Git tag: `cmake/version.cmake` sets it in CMake builds and the release workflow stamps it into the packaged `x430.h`. A header taken from the repository and built without CMake reports `0.0.0` (unknown). Every change is recorded under `[Unreleased]` in `CHANGELOG.md`.

A release tag runs `release.yml`: changelog check, version stamp, unit tests with the coverage gate in every unit test preset, MISRA gate, release build, and a GitHub Release with the static library, the headers, the MISRA report and the PDF manual.

2.8 Measured figures

`tools/x430_measure.sh <build-dir> [dxl2-dir]` prints every figure of the Configuration (p. 26) chapter: the instance sizes on Cortex-M4 and on the host, the flash of each module and the worst-case stack of every public function.

2.9 Documentation

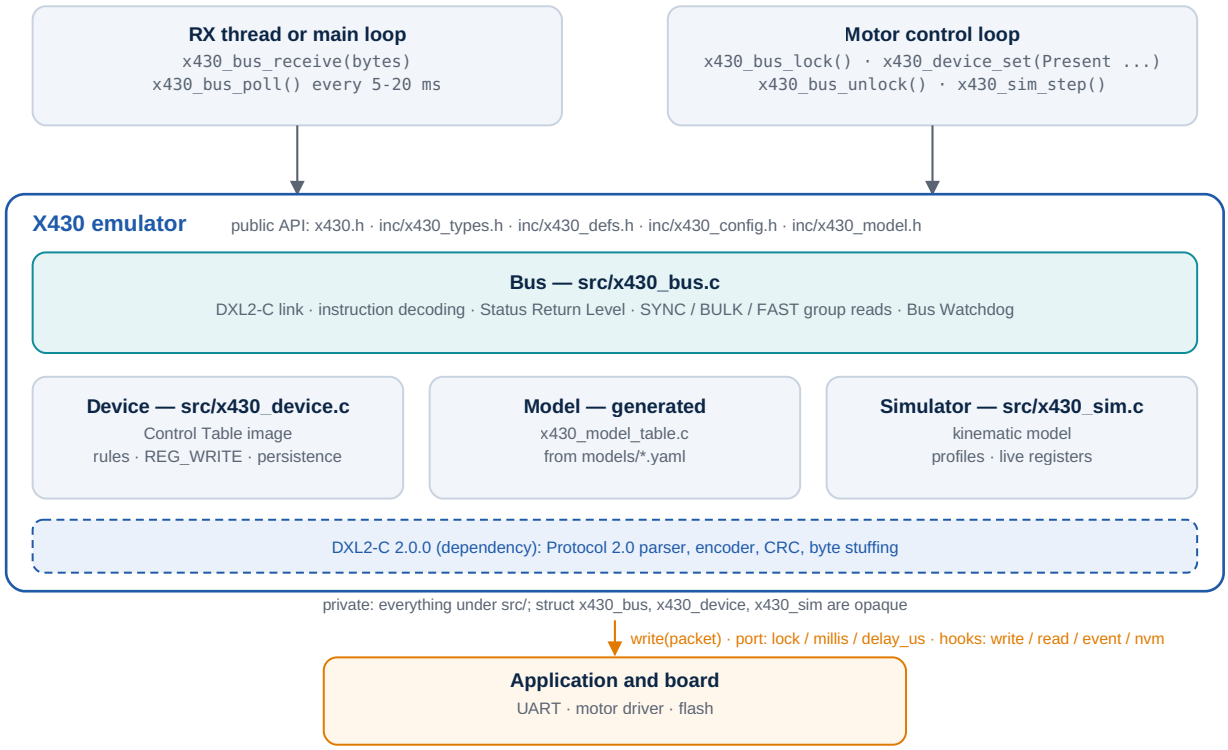
Task	Command	VS Code task
Live preview	`mkdocs serve`	*Docs: Serve on local-host*
Build the website	`mkdocs build`	*Docs: Build site*
Build the PDF manual	`python tools/docs/build_manual.py`	*Docs: Build PDF manual*
Regenerate diagrams	`python tools/docs/diagrams.py`	*Docs: Generate diagrams*
Compile the C snippets	`python tools/docs/check_snippets.py` (after configuring `X430[Debug]`)	*Docs: Check C snippets*

CHAPTER 3

How It Works

This chapter explains the parts of the X430 emulator, the contexts it runs in, the states of a device, who owns which memory, how errors are reported and what may run in parallel. What a host sees, instruction by instruction, is in X430 Emulation (p. 30).

3.1 Modules

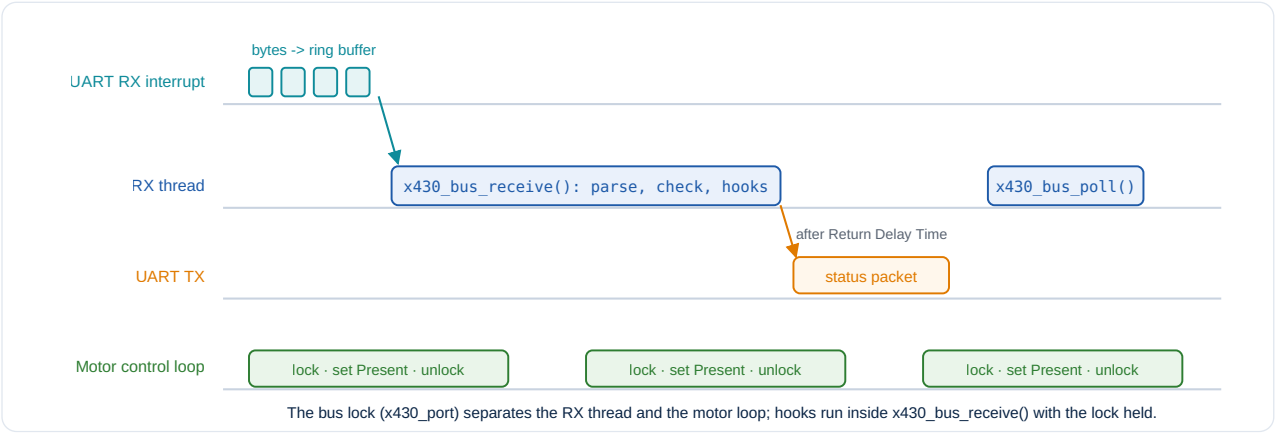


Module	Files	Responsibility
Public API	`x430.h`, `inc/x430_types.h`, `inc/x430_defs.h`, `inc/x430_config.h`, `inc/x430_model.h`	Functions, configuration types, hooks, status codes, Control Table addresses, build-time options, the model descriptor type
Bus	`src/x430_bus.c`	DXL2-C link, addressing (ID, Secondary ID, broadcast), Status Return Level, Return Delay Time, single and group instructions, Fast read packets, Bus Watchdog timing, the bus lock
Device	`src/x430_device.c`	Control Table image, write checks, write hook and storage, Indirect Address, REG_WRITE/ACTION, REBOOT, FACTORY_RESET, CLEAR, BACKUP, hardware errors, persistent blocks

Module	Files	Responsibility
Model	`<build>/x430/generated/x430_model_table.c`, `x430_model_info.h`	The register list and address map of the selected model, written by `tools/x430_gen_model.py` from `models/`
Simulator	`src/x430_sim.c`	Kinematic model of the actuator behind a device (optional)
DXL2-C	dependency	Protocol 2.0 parser, encoder, CRC-16, byte stuffing

Only the public API is visible to the application. Everything under `src/` is private: the instances `struct x430_bus`, `struct x430_device` and `struct x430_sim` are opaque blocks of memory whose layout may change in any release.

3.2 Execution contexts



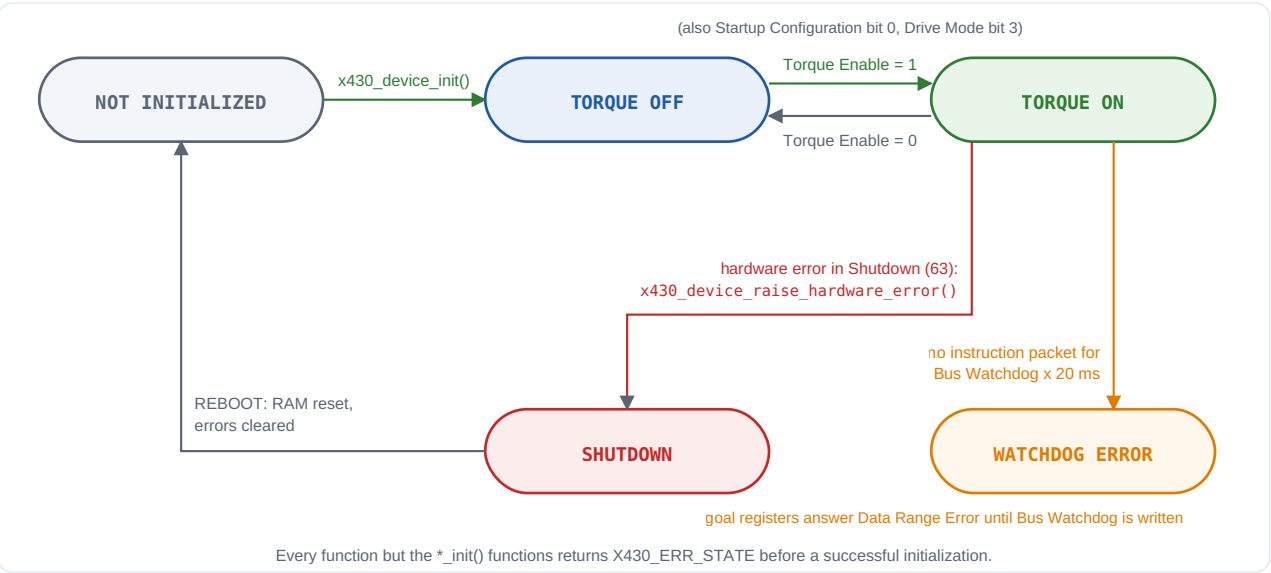
The bus lock (x430_port) separates the RX thread and the motor loop; hooks run inside x430_bus_receive() with the lock held.

	Bus context	Motor context
Functions	`x430_bus_receive()`, `x430_bus_handle_packet()`, `x430_bus_poll()`	`x430_bus_lock()`, `x430_device_set()` / `x430_device_get()`, `x430_bus_unlock()`, `x430_sim_step()`
Context	One RX thread or the main loop, never an interrupt	The motor control loop (task or main loop)
Rate	As bytes arrive; `x430_bus_poll()` every 5–20 ms	The rate of the control loop (for example 1 kHz)
Work	Parse, check, call the hooks, store, answer	Copy the motor state into the live registers
Time	Bounded by the packet: loops over its parameters, the touched registers, the devices of the bus	Bounded; `x430_sim_step()` is proportional to its `elapsed_ms`

The two contexts share the Control Tables. The bus functions take the bus lock (`x430_port::lock`) for the whole packet; the motor context takes the same lock around its updates with `x430_bus_lock()`. On bare metal both run in the main loop and the port has no lock.

Hooks run inside the bus context with the lock held. A hook may call the lock-free device functions (`x430_device_get()`, `x430_device_set()`, `x430_device_id()`, `x430_device_raise_hardware_error()`) but no bus function; to hand work to another thread, post to an application queue from the hook.

3.3 States



State	Torque Enable (64)	Entered by	Left by
Not initialized	–	a zeroed or never initialized instance	`x430_device_init()`
Torque off	0	initialization, REBOOT, a write of 0, a shutdown	a write of 1, Startup Configuration bit 0, a goal write with Drive Mode bit 3
Torque on	1	the transitions above; Goal Position takes Present Position	a write of 0, a hardware error in Shutdown (63)
Shutdown	0, locked	`x430_device_raise_hardware_error()` with a bit enabled in Shutdown	REBOOT (Hardware Error Status cleared)
Watchdog error	1, Bus Watchdog reads −1	no instruction packet for Bus Watchdog × 20 ms	a write of Bus Watchdog

While a hardware error bit is set, every status packet carries the alert bit (0x80).

3.4 Memory model

Object	Owner	Size on Cortex-M4	Lifetime
`struct x430_bus`	Application (static, stack or pool)	1,664 B (768-byte packets, the default), 640 B (256-byte packets)	From `x430_bus_init()`
`struct x430_device`	Application	384 B (with every feature and without)	From `x430_device_init()`
`struct x430_sim`	Application	32 B	From `x430_sim_init()`

Object	Owner	Size on Cortex-M4	Lifetime
Configurations	Application; copied by the *_init() functions	–	Only during the call
Hooks (`struct x430_hooks`), port (`struct x430_port`)	Application	–	As long as the device or bus (the library keeps a pointer)
Model table	Library (`const`, flash)	921 B	Static

A device holds the compact image of its Control Table (316 bytes: the 662 addresses without their long reserved runs, theory (p. 34)) and the REG_WRITE data (`X430_REG_WRITE_SIZE`); it keeps no copy of a persistent block, the hooks read and write blocks in parts. A bus holds the DXL2-C link (receive buffer and decoded packet) and one packet buffer in which it builds and byte-stuffs every status packet, Fast Sync/Bulk Read answers included. There is no heap, no global or static mutable variable and no hidden buffer: the sizes are fixed at compile time by `X430_BUS_WORDS`, `X430_DEVICE_WORDS` and `X430_SIM_WORDS` (plain arithmetic of the configuration macros, which static analysis can evaluate) and checked against the private state by `_Static_assert`.

3.5 Error handling

Situation	Result
`NULL` instance or argument	`X430_ERR_PARAM`; nothing is changed
A device ID above 252, a bus without writer	`x430_device_init()` / `x430_bus_init()` return `X430_ERR_PARAM`; the instance is not changed
Any call before a successful initialization	`X430_ERR_STATE`; no hook is called
Attaching a device twice	`X430_ERR_STATE`
Bus full, ID already used on the bus	`X430_ERR_CONFIG`
The byte writer returns non-zero	`x430_bus_receive()` / `x430_bus_handle_packet()` return `X430_ERR_HAL` (once)
A feature that is not compiled in	`X430_ERR_UNSUPPORTED` (`x430_device_backup()`, `x430_sim_*()`)
An instruction the device refuses	`X430_OK`; the protocol error (`DXL2_ERR_*`) goes into the status packet and into the `error` output
Malformed packet (CRC, length)	Dropped by DXL2-C; no answer, as a real device

Instruction semantics never fail at the API level: an invalid address, length or value is a protocol error for the host, not an error of the application.

3.6 Threading and reentrancy

- **Per bus:** one bus context; the bus functions are not reentrant for the same bus.

- **Live register updates** from another context go between `x430_bus_lock()` and `x430_bus_unlock()`; the lock must be recursive if a hook can lead to code that takes it again.
- **Different buses** share nothing and may run in parallel.
- **Hooks** run in the context of the bus function that calls them.

3.7 Design decisions

Decision	Reason
One model per build, generated tables	The Control Table is data (YAML checked against the e-Manual), not code; a build carries only the table it needs.
Library = table and protocol, motor = application	The same library serves steppers, BLDC drives and the simulator; motion control has no place in a protocol emulator.
Two passes for a write: check every register, then hook and store	A multi-register write is all or nothing for the checks, as on the real actuator.
Hooks receive real addresses	Indirect Data is resolved by the library; the application needs one write path.
Status codes for API misuse, `DXL2_ERR_*` for the host	Errors of the application and errors of the host never mix.
Persistent blocks with header and CRC	A block from another model, format or a damaged flash page is ignored; the device starts with factory values.
Opaque instances	The application cannot depend on internal fields (MISRA deviation DEV-01).
`X430_ERR_STATE` before initialization	A zeroed or garbage instance never calls a NULL hook.

CHAPTER 4

Using the Library

This chapter shows how to set up a bus and its devices, how the application reacts to the host through the hooks, how it keeps the live registers current and how errors come back. The UART, RTOS and flash side is in the Integration Guide (p. 37); the build options are in Configuration (p. 23).

4.1 Setting up a bus and its devices

One `struct x430_bus` serves one serial line; one `struct x430_device` is one DYNAMIXEL ID. Initialize the bus and every device, then attach the devices:

```
static struct x430_bus bus;
static struct x430_device joint;

static int uart_write(void* context, const uint8_t* data, size_t length)
{
    (void)context;
    return board_uart_write(data, length);
}

int app_start(void)
{
    const struct x430_bus_config bus_config = {.write = uart_write, .port = &port};
    const struct x430_device_config joint_config = {
        .id = 11U,                  /* factory ID */
        .firmware_version = 0U,     /* 0: X430_FIRMWARE_VERSION (52) */
        .model_information = 0x00010000U,
        .hooks = &hooks,
        .user_data = NULL,
    };
    int result = x430_bus_init(&bus, &bus_config);

    if (result == X430_OK)
    {
        result = x430_device_init(&joint, &joint_config);
    }
    if (result == X430_OK)
    {
        result = x430_bus_attach(&bus, &joint);
    }
    return result;
}
```

`x430_device_config::id` is the **factory ID**: it is used while no EEPROM block is stored and again after FACTORY_RESET 0xFF; an ID stored by the host wins at startup. `x430_device_init()` loads the factory values, the stored EEPROM area (`nvm_read`), the RAM area and applies the Startup Configuration — which may already call the write hook to switch the torque on. Initialize a device before attaching it, never while a bus serves it. A dual-axis 2XL430/2XC430 is two devices (factory IDs 1 and 2).

4.2 Reacting to the host: hooks

Hook	Called when	The application
`write`	For every register a WRITE, SYNC_WRITE, BULK_WRITE or ACTION touches, after all of them passed the checks, before the value is stored	Applies goals, limits and modes to the motor; returns `DXL2_ERR_NONE` or a `DXL2_ERR_*` code to refuse
`read`	Before data is read, once per run of contiguous real addresses	May refresh the live registers it covers
`event`	REBOOT, FACTORY_RESET, CLEAR, Bus Watchdog, Baud Rate, shutdown, backup restored	Resets or stops the motor, switches the UART
`nvm_read`, `nvm_write`	Initialization, REBOOT, EEPROM writes, FACTORY_RESET, BACKUP	Reads or writes part of an opaque block at an offset

Every member may be NULL. A write hook for a stepper joint:

```c
static uint8_t joint_write(struct x430_device* device, uint16_t address, uint8_t size,
                           int32_t value, void* user_data)
{
    uint8_t result = DXL2_ERR_NONE;
    int32_t mode = 0;

    (void)size;
    (void)user_data;
    (void)x430_device_get(device, X430_ADDR_OPERATING_MODE, &mode); /* already validated
*/
    switch (address)
    {
        case X430_ADDR_TORQUE_ENABLE:
            if (!motor_enable(value != 0))
            {
                result = DXL2_ERR_RESULT_FAIL;
            }
            break;
        case X430_ADDR_GOAL_POSITION:
            motor_move_to(value);
            break;
        case X430_ADDR_GOAL_VELOCITY:
            if (mode == (int32_t)X430_MODE_VELOCITY)
            {
                motor_run(value);
            }
            break;
        default:
            break; /* stored only */
    }
    return result;
}
```

The value arrives sign-extended and already checked (range, limit registers, EEPROM lock, Bus Watchdog). On Torque Enable 0 → 1 the library copies Present Position into Goal Position after the hook, so keep Present Position current.

4.3 Keeping the live registers current

The device does not know the motor. Update Present Position (132), Present Velocity (128), Present Current/Load (126), Present PWM (124), Moving (122), Moving Status (123), Velocity/Position Trajectory (136/140), Present Input Voltage (144) and Present Temperature (146) with `x430_device_set()`, either periodically:

```
void motor_loop_1ms(void)
{
    (void)x430_bus_lock(&bus);
    (void)x430_device_set(&joint, X430_ADDR_PRESENT_POSITION, motor_position());
    (void)x430_device_set(&joint, X430_ADDR_PRESENT_VELOCITY, motor_velocity());
    (void)x430_device_set(&joint, X430_ADDR_MOVING, motor_moving() ? 1 : 0);
    (void)x430_bus_unlock(&bus);
}
```

or on demand in the `read` hook, where the lock is already held. `x430_device_set()` checks nothing but that a register starts at the address.

4.4 Faults

Report a fault of the power stage with `x430_device_raise_hardware_error()`:

```
void on_overtemperature(void)
{
    (void)x430_bus_lock(&bus);
    (void)x430_device_raise_hardware_error(&joint, X430_HW_ERROR_OVERHEATING);
    (void)x430_bus_unlock(&bus);
}
```

The bit stays in Hardware Error Status (70) until REBOOT and every status packet carries the alert bit. If the bit is enabled in Shutdown (63) and the torque is on, the library switches Torque Enable off, calls the event hook with `X430_EVENT_TORQUE_SHUTDOWN` and refuses Torque Enable = 1 with Result Fail until REBOOT.

4.5 Using the simulator

Instead of a motor, a device can run the built-in model:

```
int sim_start(void)
{
    const struct x430_device_config config = {
        .id = 1U, .hooks = &x430_sim_hooks, .user_data = &joint_sim};
    int result = x430_sim_init(&joint_sim, &joint, NULL); /* before the device */

    if (result == X430_OK)
    {
        result = x430_device_init(&joint, &config);
    }
    return result;
}

void sim_tick_1ms(void)
{
    (void)x430_sim_step(&joint_sim, 1U); /* takes the bus lock when attached */
}
```

4.6 Return values

Code	Value	Returned when
`X430_OK`	0	Success (also when the host gets a protocol error)
`X430_ERR_PARAM`	−1	`NULL` pointer, a value outside its range, no register at the address, no device with the ID
`X430_ERR_STATE`	−2	An instance without a successful initialization; a device that is already attached
`X430_ERR_CONFIG`	−3	Bus full, or the device ID is already used on the bus
`X430_ERR_HAL`	−4	The byte writer returned non-zero for a status packet
`X430_ERR_UNSUPPORTED`	−5	BACKUP or the simulator is not compiled in

4.7 Common mistakes

Symptom	Likely cause
Every call returns `X430_ERR_STATE`	The *_init() function failed or was not called
The host gets no answer	Wrong baud rate or ID; Status Return Level 0/1; the received bytes never reach `x430_bus_receive()`
DYNAMIXEL Wizard shows TIME OUT on the Control Table tab	`DXL2_MAX_PACKET_SIZE` was lowered below 768: the Wizard reads the whole table (662 bytes) at once (Configuration)
ROS 2 refuses the device	The Model Number is not in its model files (XW430-T200); use `CONFIG_X430_MODEL_NUMBER_OVERRIDE`
The arm jumps when the torque is switched on	Present Position was not current when the host wrote Torque Enable
EEPROM writes answer Access Error	The torque is on (as on a real X430), or the packet used the Secondary ID
The host sees the old values after a power cycle	No `nvm_write` hook, `nvm_write` does not keep the parts at their offsets, or `nvm_read` fails
Deadlock in a hook	The hook called a bus function, or the lock is not recursive

CHAPTER 5

Extending the Library

Most extensions are built on top of the API: a motor behind the hooks, other devices on the same line, a product-specific range of a register. This chapter shows where each belongs and the rules for changes to the library itself.

5.1 Where to extend

You want to…	Do it in	Library change needed?
Run on another MCU, UART or RTOS	The byte writer, the port and the RX code	No
Drive another kind of motor	The hooks of the device	No
Change a range or an initial value of the model (for example a supply above 16 V)	An application overrides file (Configuration)	No
Report your firmware version to the host	`x430_device_config::model_information` (Model Information, 2)	No
Accept more Baud Rate codes	`X430_BAUD_RATE_CODE_MAX` (in `COMPILE_DEFINITIONS` or Kconfig) and a check in the write hook	No
Host other, non-X430 devices on the same line	Your own DXL2-C link and `x430_bus_handle_packet()` (below)	No
Answer for registers beyond the X430 table	Your own router before the bus (below)	No
Add an X430 model	`models/x430/<model>.yaml`, `Kconfig.x430`, the e-Manual snapshot	Model data only
Emulate another DYNAMIXEL family	`models/<family>_family.yaml` and its models	Model data; the addresses of `x430_defs.h` must match
A new instruction	`src/x430_bus.c`	Yes

5.2 Several kinds of devices on one line

An application that already runs its own DXL2-C link — for example to host a controller device with its own Control Table next to the joints — routes the packets itself and hands the rest to the bus:

```
#define CONTROLLER_ID 200U

void controller_handle(const struct dxl2_packet* packet);

static void on_packet(const struct dxl2_packet* packet, void* context)
```

```
{
    (void)context;
    if (packet->id == CONTROLLER_ID)
    {
        controller_handle(packet);
    }
    else
    {
        (void)x430_bus_handle_packet(&bus, packet); /* answers through the bus writer */
    }
}
```

The same router can answer READs of an address window beyond the 662-byte X430 table (a driver register window, for example) before the bus sees them; the bus would answer them with an Access Error.

5.3 Adding a model

1. Write `models/x430/<model>.yaml` with the differences from `models/x430_family.yaml` (copy a similar model): Model Number, variant (`current` or `load`), operating modes, removed registers, initial values and ranges.
2. Add the choice entry and the `X430_MODEL_NAME` default to `Kconfig.x430` (`tests/models` fails if it is missing).
3. Add the e-Manual rows to `tests/models/emanual_x430.json`.
4. Run the unit tests with `-DX430_MODEL=<model>`.

`python3 tools/x430_gen_model.py --models-dir models --dump <model>` prints the resolved table.

5.4 Contributing to the library

1. **Code.** C11, no new dependencies, no heap, no global state; the rules of `misra-analyze`; internal types stay in `src/`, `x430.h` declares the public API only.
2. **Tests.** 100 % line and branch coverage of `src/` in every unit test preset (`X430[UnitTests]`, `X430[UnitTestsMinimal]`); the test binary also passes in one process.
3. **Instance size.** If a private state grows, update the arithmetic of `X430_BUS_BYTES`, `X430_DEVICE_BYTES` or `X430_SIM_BYTES` in `inc/x430_types.h`; the build fails until it fits, and a unit test fails if it wastes a word.
4. **Analysis.** `misra-analyze` 0 violations with Cppcheck 2.13, or a recorded deviation in MISRA C:2012 (p. 60).
5. **Format.** `clang-format` 18 reports no changes.
6. **Documentation.** The affected pages, the measured figures (`tools/x430_measure.sh`), the diagrams and `CHANGELOG.md` `[Unreleased]`.
7. **Compatibility.** The public API is stable within a major version.

CHAPTER 6

Configuration

The X430 emulator has two kinds of configuration. **Build-time options** select the emulated model, the sizes and the optional instructions; they come from Kconfig or from plain compiler definitions and end up as macros of `inc/x430_config.h`. **Run-time configuration** is the content of `struct x430_bus_config`, `struct x430_device_config` and `struct x430_sim_config`, passed to the *_init() functions. This chapter lists both, then the memory footprint and the measured stack usage.

6.1 Build-time options

Kconfig symbol	Macro	Default	Range	Meaning
`X430_MODEL_*` (choice)	`X430_MODEL_NAME`, `X430_MODEL_IS_<KEY>`	XM430-W350	17 models	Emulated model (one per build); CMake `X430_MODEL` without Kconfig
`X430_MODEL_NUMBER_OVERRIDE`	`X430_MODEL_NUMBER`	0 = documented number	0..65535	Model Number reported by PING and address 0
`X430_MODEL_OVERRIDES`	–	empty	file	YAML file of application overrides of the model (below)
`X430_FIRMWARE_VERSION`	`X430_FIRMWARE_VERSION`	52	1..255	Firmware Version (6) when the device configuration gives 0
`X430_MAX_DEVICES`	`X430_MAX_DEVICES`	4	1..253	Devices (IDs) per bus
`X430_REG_WRITE_SIZE`	`X430_REG_WRITE_SIZE`	32	4..662	Largest REG_WRITE data per device, bytes
`X430_BAUD_RATE_CODE_MAX`	`X430_BAUD_RATE_CODE_MAX`	7	7..255	Highest Baud Rate (8) code a write accepts
`X430_TURN_TIMEOUT_MS`	`X430_TURN_TIMEOUT_MS`	100	1..60000	Shared bus: longest wait for the status of the ID before a hosted device in a SYNC_READ / BULK_READ (group instructions)
`X430_DXL2_MAX_PACKET_SIZE`	`DXL2_MAX_PACKET_SIZE` (DXL2-C)	768	64..4096	Largest packet; limits READ length and Fast read answers
`X430_BULK`	`X430_FEATURE_BULK`	y	y/n	BULK_READ, BULK_WRITE
`X430_FAST_READ`	`X430_FEATURE_FAST_READ`	y	y/n	FAST_SYNC_READ, FAST_BULK_READ

Kconfig symbol	Macro	Default	Range	Meaning
X430_BACKUP	X430_FEATURE_BACKUP	y	y/n	BACKUP, Backup Ready, Startup RAM Restore
X430_SIMULATOR	X430_FEATURE_SIMULATOR	y	y/n	x430_sim

The ranges are those of Kconfig.x430; without Kconfig, x430_config.h checks X430_MAX_DEVICES and X430_REG_WRITE_SIZE with #error and DXL2-C checks its packet size. The application must be compiled with the same values as the library; with CMake they are public compile definitions of the x430 target.

Without Kconfig (the default) the options are compiler definitions, given in the CMake variable COMPILE_DEFINITIONS; compiler options of the library sources (optimization, CPU flags) go into COMPILE_OPTIONS. Both names have no prefix on purpose: a project sets them once for all the libraries it includes.

With Kconfig

With X430_KCONFIG=ON (off by default; preset X430[Kconfig]), Kconfiglib (github.com/ulfalizer/Kconfiglib) merges the fragments of X430_CONF_FILES (default configs/defconfig) into <build>/x430/.config and writes x430_autoconf.h. As long as the fragments do not change, the existing .config is kept, so values chosen in menuconfig survive.

```
cmake --preset "X430[Kconfig]"
cmake --build "build/X430[Kconfig]" --target menuconfig      # terminal UI
cmake --build "build/X430[Kconfig]" --target savedefconfig   # minimal fragment
cmake -S . -B build -DX430_CONF_FILES="$PWD/configs/defconfig;$PWD/my_robot.conf"
```

```
# my_robot.conf
CONFIG_X430_MODEL_XW430_T333=y
CONFIG_X430_MAX_DEVICES=6
CONFIG_X430_BACKUP=n
```

A project with its own Kconfig tree switches X430_KCONFIG on, points X430_KCONFIG_ROOT at its top-level Kconfig and sources Kconfig.x430 from it. An RTOS with its own Kconfig (Zephyr) keeps X430_KCONFIG off: the application Kconfig sources Kconfig.x430 and the glue passes the CONFIG_X430_* values to the CMake variables.

Without Kconfig

Without Kconfig only Python 3 is needed. Every optional feature is on unless defined to 0; options are compiler definitions, given in COMPILE_DEFINITIONS:

```
cmake -S . -B build -DX430_MODEL=xl430_w250 \
    -DCOMPILE_DEFINITIONS="X430_MAX_DEVICES=8U;X430_FEATURE_BACKUP=0" -DCOMPILE_OPTIONS="-
Os"
```

Packet size

`DXL2_MAX_PACKET_SIZE` limits the length of a READ answer to `DXL2_MAX_PACKET_SIZE − 11` bytes and the size of a Fast read answer to `DXL2_MAX_PACKET_SIZE` bytes (`10 + devices × (bytes + 4)`). The default is 768: a READ of up to 757 bytes, so a host can read the whole Control Table (addresses 0–661, 662 bytes) at once, as DYNAMIXEL Wizard 2.0 does (with 256 it gets a Data Length Error and shows TIME OUT). A product whose hosts read less may lower it: with 256 the bus needs 640 B instead of 1,664 B (the stack does not depend on the packet size: the bus builds its answers in its own buffer).

The default has two places, one per build system: `CONFIG_X430_DXL2_MAX_PACKET_SIZE` in `Kconfig.x430` and the cache variable `X430_DXL2_MAX_PACKET_SIZE` in `x430.cmake`, which gives it to DXL2-C as a public compiler definition, so the library, its tests and the application that links `x430` all see the same value. `x430_config.h` has no default of its own and stops a build that sets neither. Without CMake, define `DXL2_MAX_PACKET_SIZE` (for example `-DDXL2_MAX_PACKET_SIZE=768U`) for the library, DXL2-C and the application alike; the default of DXL2-C alone is 256.

Application overrides of the model

A product built on the emulator may need another range or initial value than the ROBOTIS model, for example a supply above the 16 V of an XM430. Instead of editing `models/`, give a YAML file with the same `overrides:` mapping as the model files:

```
overrides:
  Max Voltage Limit: {min: 95, max: 300}
  Min Voltage Limit: {min: 95, max: 300}
```

Set its path in `CONFIG_X430_MODEL_OVERRIDES` or in the CMake cache variable `X430_MODEL_OVERRIDES`. `tools/x430_gen_model.py --overrides <file>` applies it after the model file; an unknown register name or an initial value outside the resulting range stops the build.

6.2 Run-time configuration

The *_init() functions copy the structures, so they may be temporaries or `const` in flash. The hooks and the port they point at are not copied: keep them alive.

`struct x430_bus_config`

Field	Required	Meaning
`write`	Yes	`int write(void* context, const uint8_t* data, size_t length)`: writes one complete status packet; returns 0, or non-zero on error (reported as `X430_ERR_HAL`)
`write_context`	–	Passed to `write`
`port`	–	`struct x430_port`: platform services, NULL for none

`struct x430_port`

Member	Called from	Used for	NULL means
`context`	–	Passed to every member	–
`lock`, `unlock`	Bus functions, `x430_bus_lock()`, `x430_bus_unlock()`, `x430_sim_step()`	Mutual exclusion of the Control Tables	Single context (bare metal)
`millis`	Bus functions, `x430_device_read()`	Realtime Tick (120), Bus Watchdog	Tick 0, no watchdog timeout
`delay_us`	Before each status packet	Return Delay Time (9) × 2 µs	Status sent at once

`struct x430_device_config`

Field	Unit	Range	Example	Meaning
`id`	–	0..252	11	Factory ID (0 = model default 1); a stored ID wins
`firmware_version`	–	0..255	0	Firmware Version (6); 0 = `X430_FIRMWARE_VERSION`
`model_information`	–	32 bit	`0x00010000`	Model Information (2), free for the application
`hooks`	–	pointer	`&hooks`	Application hooks, NULL = pure register device
`user_data`	–	pointer	`&motor`	Passed to every hook

`struct x430_sim_config`

Field	Unit	Default (0)	Meaning
`input_voltage`	0.1 V	120 (12.0 V)	Present Input Voltage (144)
`temperature`	°C	30	Present Temperature (146)

6.3 Memory footprint

Measured with `tools/x430_measure.sh` and `arm-none-eabi-gcc` 13.2.1 for Cortex-M4 (`-mcpu=cortex-m4 -mthumb`), model XM430-W350, `X430_MAX_DEVICES` 4, `X430_REG_WRITE_SIZE` 32, `DXL2_MAX_PACKET_SIZE` 768:

Object	All features	No optional feature
`struct x430_device` (RAM per device)	384 B	384 B
`struct x430_bus` (RAM per bus)	1,664 B	1,664 B
`struct x430_bus`, `DXL2_MAX_PACKET_SIZE` 256	640 B	640 B
`struct x430_sim`	32 B	–
`.data` + `.bss`	0 B	0 B

A device is its compact Control Table image (316 B), the REG_WRITE data (`X430_REG_WRITE_SIZE`, 32 B) and its state; it keeps no copy of a persistent block. A bus is the DXL2-C link (receive buffer, in which DXL2-C 2.0.0 also decodes the packet: 804 B with 768-byte packets), one packet buffer for its answers (`DXL2_MAX_PACKET_SIZE`), the device list (`X430_MAX_DEVICES` pointers) and the pending SYNC_READ / BULK_READ of a shared bus (`X430_MAX_DEVICES` entries of 12 B). The optional features change code, not RAM.

Flash (`.text`, including constants) per module:

Module	`-Os`	`-O2`	`-Os`, no optional feature
`x430_device.c`	4,158 B	5,916 B	3,814 B
`x430_bus.c`	3,018 B	4,132 B	2,242 B
`x430_sim.c`	2,022 B	2,364 B	32 B
`x430_model_table.c` (XM430-W350)	921 B	921 B	921 B
Library	**10,119 B**	**13,333 B**	**7,009 B**
DXL2-C (`dxl2_protocol.c`, `dxl2_link.c`)	842 B	980 B	842 B

The model table holds 55 registers of 16 bytes, the two Indirect Address blocks (the library generates their 56 registers) and the three segments of the image; a register is found by binary search. DXL2-C is listed as compiled: the bus uses only its parser and `dxl2_crc16()`, so a link with `--gc-sections` drops `dxl2_link_send()` and `dxl2_link_send_status()`.

On the x86-64 host the device needs 392 B and the bus 2,488 B (64-bit pointers).

6.4 Stack usage

Worst-case stack of each public function including the library and DXL2-C functions it calls, measured with `-fstack-usage -fcallgraph-info=su` and `tools/docs/stack_usage.py`. The packet handler and the byte writer that DXL2-C calls through function pointers are included; the application's hooks, port functions and byte writer are not — add their stack to the function that calls them.

Function	Cortex-M4 `-Os`	Cortex-M4 `-O0`	`-Os`, packet 256
`x430_bus_init()`	48	104	48
`x430_bus_attach()`	40	104	40
`x430_bus_receive()`	424	656	424
`x430_bus_handle_packet()`	392	600	392
`x430_bus_poll()`	32	104	32
`x430_bus_lock()`, `x430_bus_unlock()`	8	64	8
`x430_bus_find()`	40	104	40
`x430_device_init()`	136	208	136
`x430_device_get()`	92	144	92
`x430_device_set()`	92	144	92

Function	Cortex-M4 `-Os`	Cortex-M4 `-O0`	`-Os`, packet 256
`x430_device_id()`	16	96	16
`x430_device_raise_hardware_error()`	20	72	20
`x430_device_flush_events()`	20	72	20
`x430_device_read()`	112	136	112
`x430_device_write()`	256	384	256
`x430_device_reg_write()`	264	384	264
`x430_device_action()`	248	384	248
`x430_device_reboot()`	128	200	128
`x430_device_factory_reset()`	144	216	144
`x430_device_clear()`	108	176	108
`x430_device_backup()`	120	192	120
`x430_sim_init()`	16	80	16
`x430_sim_step()`	284	680	284

The deepest chain of `x430_bus_receive()` is a REG_WRITE: dispatch, the write of the device (the written bytes merged with Indirect Data resolved). No packet-sized buffer is on the stack: the bus builds and byte-stuffs each status packet in place in its own buffer, and the device reads and writes persistent blocks in parts (a 16-byte chunk on the stack while it checks a block). Give the bus context this figure plus its hooks.

To re-measure (in a configured build directory):

```
tools/x430_measure.sh "build/X430[Release]"
```

6.5 CMake options

Option	Default	Meaning
`COMPILE_DEFINITIONS`	empty	Compile definitions of the library (public), e.g. `X430_MAX_DEVICES=8U`
`COMPILE_OPTIONS`	empty	Compile options of the library sources, e.g. `-Os`
`UNIT_TESTS`	`OFF`	Unit tests: in this repository the GoogleTest suite and the `coverage` target; as a submodule the executable `x430_gtest`, registered with the CTest of the project (label `x430`)
`X430_KCONFIG`	`OFF`	Configure with Kconfiglib
`X430_CONF_FILES`	`configs/defconfig`	Kconfig fragments
`X430_KCONFIG_ROOT`	`Kconfig`	Top-level Kconfig file

Option	Default	Meaning
`X430_MODEL`	`xm430_w350`	Model without Kconfig
`X430_MODEL_OVERRIDES`	empty	Application overrides of the model without Kconfig
`X430_DXL2_MAX_PACKET_SIZE`	768	`DXL2_MAX_PACKET_SIZE` without Kconfig
`X430_DXL2_GIT_TAG`	`v2.0.0`	DXL2-C release fetched with FetchContent
`FETCHCONTENT_SOURCE_DIR_DXL2`	–	Local DXL2-C checkout (offline builds)
`X430_EXAMPLES`	`OFF`	POSIX host example
`X430_COVERAGE_MIN`	`100`	Coverage gate (lines and branches, %)
`X430_MISRA_FAIL_ON_VIOLATION`	`ON`	`misra-analyze` fails on any violation
`X430_MISRA_RULE_TEXTS`	empty	Licensed MISRA rule texts for readable reports

CHAPTER 7

X430 Emulation

This chapter describes what a host sees: the Control Table model, how each instruction is executed, the rules a real X430 enforces and how the library reproduces them, the format of the persistent blocks, and the kinematic model of the simulator with its equations. Every rule is taken from the code (`src/`) and checked by the unit tests in `tests/unit/`; the measured results come from `tests/unit/gtest_x430_sim.cpp` and from the DynamixelSDK check in `tests/sdk/`.

7.1 The Control Table model

`models/x430_family.yaml` holds the Control Table shared by the X430 family (662 addresses, two Indirect Address blocks of 28 entries) with two variants:

Variant	Models	Current registers	Operating modes
`current`	XM430, XH430, XD430, XW430	Current Limit (38), Goal Current (102), Present Current (126)	0, 1, 3, 4, 5, 16
`load`	XL430, XC430, 2XL430, 2XC430	Present Load (126) only	1, 3, 4, 16

`models/x430/<model>.yaml` lists what differs: Model Number, variant, removed registers (LED on the XW430) and initial values or ranges. `tools/x430_gen_model.py` merges both, validates the result (sizes, overlaps, EEPROM below address 64, a check for every writable register) and writes `x430_model_table.c` with the register list and an address map (one byte per address: register index + 1, 0 = reserved).

Every register has a write check:

Check	Valid values	Registers (examples)
range	min..max	ID (0..252), Return Delay Time (0..254), limits
mask	only the bits of a mask	Drive Mode (bits 0, 2, 3), Shutdown
opmode	an operating mode of the model	Operating Mode (11)
symmetric	−limit..+limit, the limit is another register	Goal PWM (±36), Goal Current (±38), Goal Velocity (±44)
goal position	Min..Max Position Limit (52, 48) in mode 3; ±1,048,575 in the other modes	Goal Position (116)
indirect	RAM address 64..661	Indirect Address 1–56

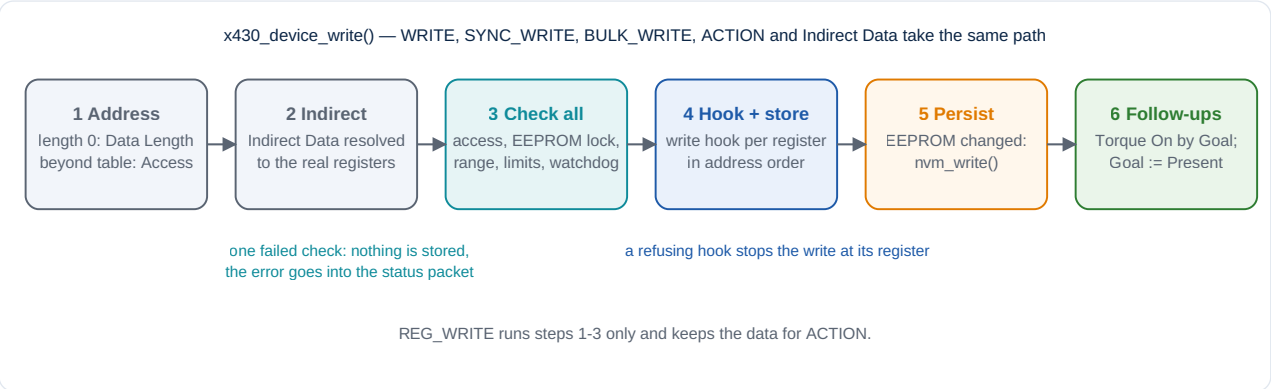
`tests/models` compares every model with a snapshot of the e-Manual control tables (size, access, area, initial value and range of every register). Differences are allowed only where the emulator deviates on purpose:

Item	e-Manual	Emulator	Why
Protocol Type (13)	1–2	2 only	Only Protocol 2.0 is emulated
Protocol Type initial on XW	1	2	Same; the XW description names 2 as the default
Secondary ID (12) range	0–252	0–255	253–255 disable the function (as described)
Bus Watchdog (98) range	1–127	0–127	0 disables and clears the error (as described)
XW430-T200 Model Number	1,270	1,280	The table repeats the number of the XW430-T333; ROBOTIS tools use 1,280
ID write that collides with another emulated device	accepted	Data Range Error	Two local devices with one ID would answer together
Partial write of a multi-byte register	–	bytes are merged	Not specified; the whole new value is checked

7.2 Addressing and status packets

- A packet to an ID is executed by the device with that **ID**; only it answers.
- Devices whose **Secondary ID** (12) equals the packet ID execute it silently; they refuse EEPROM writes and FACTORY_RESET. Values 253–255 disable the Secondary ID.
- **Broadcast** (0xFE): PING is answered by every device in ascending ID order; WRITE, REG_WRITE, ACTION, REBOOT, CLEAR, BACKUP and FACTORY_RESET (not option 0xFF) are executed silently; READ is ignored.
- Status packets of other devices on the line are ignored.
- **Status Return Level** (68): 0 = PING only, 1 = PING and READ (also the group reads), 2 = all. The level written by an instruction already applies to its own answer.
- **Return Delay Time** (9) × 2 µs is waited through `x430_port::delay_us` before each answer.
- The **alert bit** (0x80) is set in the error byte of every status while Hardware Error Status (70) is not 0.
- REBOOT and FACTORY_RESET answer first, then act. After an ID change the answer still comes from the old ID.

7.3 Writes



1. A length of 0 is a Data Length Error; a range beyond the table an Access Error.

2. Bytes written through Indirect Data are resolved to the addresses they point at.
3. Every register the write touches is checked with its new value (the bytes of the write merged into the current bytes). Errors: Access (read-only, EEPROM while the torque is on, EEPROM through the Secondary ID), Data Range (value, limit registers, a goal while a Bus Watchdog error is active), Result Fail (Torque Enable after a shutdown). If one check fails, nothing is stored.
4. In address order the write hook is called for each register; it may refuse, which stops the write at that register. Accepted registers are stored.
5. If an EEPROM register changed, the EEPROM area is stored through `nvm_write`; a failure is reported as Result Fail.
6. Drive Mode bit 3: a goal written with the torque off switches the torque on (through the write hook). Torque Enable 0 → 1 copies Present Position into Goal Position.

Reserved addresses read as 0 and ignore writes. REG_WRITE runs steps 1–3, keeps up to `X430_REG_WRITE_SIZE` bytes and sets Registered Instruction (69); ACTION runs the whole write. A Baud Rate change raises `X430_EVENT_BAUD_RATE` after the status packet, so the answer still leaves at the old rate.

7.4 Indirect Address and Data

Indirect Address 1–28 (168–222) map Indirect Data 1–28 (224–251), and 29–56 (578–632) map 634–661. An Indirect Data byte behaves like the byte it points at (size, range and access are those of the target register); a byte that points at Indirect Data itself is plain storage. ROS 2 `dynamixel_hardware_interface` uses this to read Present Position and Present Velocity with one Fast Sync Read; the DynamixelSDK check does the same.

7.5 Lifecycle

Situation	EEPROM area (0–63)	RAM area (64–661)
`x430_device_init()`	Factory values, then the stored block	Factory values, then the Startup Configuration
REBOOT	Kept	Factory values, then the Startup Configuration; errors cleared
FACTORY_RESET 0xFF / 0x01 / 0x02	Factory values (keep the ID / keep ID and Baud Rate)	As REBOOT
BACKUP 0x01	–	Whole table stored, Backup Ready (147) = 1
BACKUP 0x02 (torque off)	Restored except ID, Baud Rate, Protocol Type	Restored except Torque Enable, Bus Watchdog

Startup Configuration (60): bit 1 restores the RAM area from a valid backup, bit 0 switches the torque on (through the write hook). FACTORY_RESET 0xFF gives the device its factory ID from `x430_device_config::id` — or the model default 1 when that ID is used by another device of the bus.

Bus Watchdog

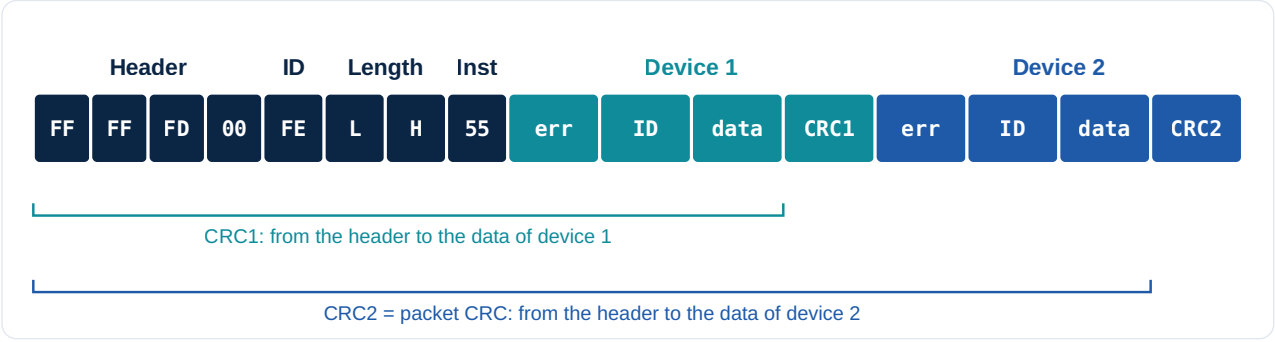
With Torque Enable = 1 and Bus Watchdog (98) = 1..127, the device expects an instruction packet at least every $T_{\mathrm{wd}} = 20\,\mathrm{ms} \times$ Bus Watchdog. The time since the last instruction packet is compared in `x430_bus_poll()` and on the next packet; when it is exceeded, Bus Watchdog reads 0xFF (−1), the event `X430_EVENT_BUS_WATCHDOG` fires and the goal registers answer Data Range Error until Bus Watchdog is written again. The resolution is the period of `x430_bus_poll()`.

CLEAR

Option 0x01 with the magic bytes `44 58 4C 22` clears the multi-turn information of Present Position. It is refused with Result Fail while Moving (122) is 1. The event hook does the work (the simulator moves its shaft); without an event hook the library wraps Present Position into 0..4095.

7.6 Group instructions

- **SYNC_READ / BULK_READ:** one status packet per hosted device, in the order of the request, taking turns like X series actuators on one wire (below).
- **FAST_SYNC_READ / FAST_BULK_READ:** one status packet with ID 0xFE. Per device it carries `[error, ID, data, CRC]`; each CRC covers the packet from the header up to that point, and the last one is the packet CRC. The packet is not byte-stuffed, as DynamixelSDK expects. If the request lists an ID that this bus does not host, the bus stays silent (below).



- **SYNC_WRITE / BULK_WRITE:** applied to the hosted IDs without status; a malformed BULK_WRITE is ignored as a whole.

Several buses on one wire

A product may put one emulator bus on each MCU and connect the MCUs to one half-duplex wire, together with real actuators: each bus then hosts only some of the IDs of a request. Every bus hears the status packets of the others (the bus decodes them; it never answers one).

- **SYNC_READ / BULK_READ take turns.** The device listed first answers at once. Every later hosted device answers only after the status packet of the ID listed just before it: at once after a device of the same bus, else when the status of that ID has been received from the wire. So the answers never overlap, as on a bus of X series actuators.
- **A missing device ends the read.** If the status of the ID before does not arrive within `X430_TURN_TIMEOUT_MS` (default 100 ms) of the last progress of the read, or the host sends its next instruction first, the remaining hosted devices of the read stay silent. DynamixelSDK stops at the first ID that does not answer anyway, and a late answer could collide with the

next instruction. A hosted device that does not answer reads (Status Return Level 0) ends the read the same way. The timeout uses the millisecond clock of the port; keep it longer than one status packet at the lowest baud rate in use plus the Return Delay Time.

- **A Fast read of several buses is not answered.** The answer to FAST_SYNC_READ and FAST_BULK_READ is one packet whose segments come from all devices, chained by a running CRC; a bus cannot build the part of another one. When the request lists an ID that this bus does not host, it stays silent, and the ROBOTIS ROS 2 hardware interface falls back to SYNC_READ after 10 attempts. With all IDs on one bus nothing changes: keep all IDs of one ROS 2 hardware component on the same emulator bus to use the Fast read.

7.7 Persistent blocks

The library hands the application opaque blocks; their layout is fixed so that a block of another model or a damaged flash page is detected:

Offset	Size	Content
0	4	Magic `X430`
4	2	Model Number (little endian)
6	1	Format version (1)
7	1	Slot: 0 EEPROM area, 1 backup
8	2	Payload size: 64 (EEPROM area) or 316 (the compact image of the device)
10	n	Payload: the first n bytes of the compact image (the table bytes 0..63 for the EEPROM area)
10 + n	2	CRC-16 of DYNAMIXEL Protocol 2.0 (polynomial 0x8005) over bytes 0..9+n

A block with any field that does not match is ignored: the device keeps its factory values. The EEPROM block is 76 bytes (`X430_NVM_EEPROM_SIZE`), the backup 328 bytes (`X430_NVM_BACKUP_SIZE`).

The **compact image** is the Control Table a device keeps in RAM without its long reserved runs: for every X430 model the addresses 0–147, 168–251 (Indirect Address 1–28 and Indirect Data 1–28) and 578–661 (Indirect Address 29–56 and Indirect Data 29–56), 316 of 662 bytes. The generator writes these segments into the model table; the reserved bytes left out read 0 like every other reserved address.

7.8 The simulator

The simulator replaces the motor with a kinematic model that follows the X430 profiles. It works in integer units: the shaft position s in 1/1000 pulse (4,096 pulses per turn) and the shaft velocity v in 1/1000 pulse per second, advanced in steps of $\Delta t = 1$ ms. The register units convert as

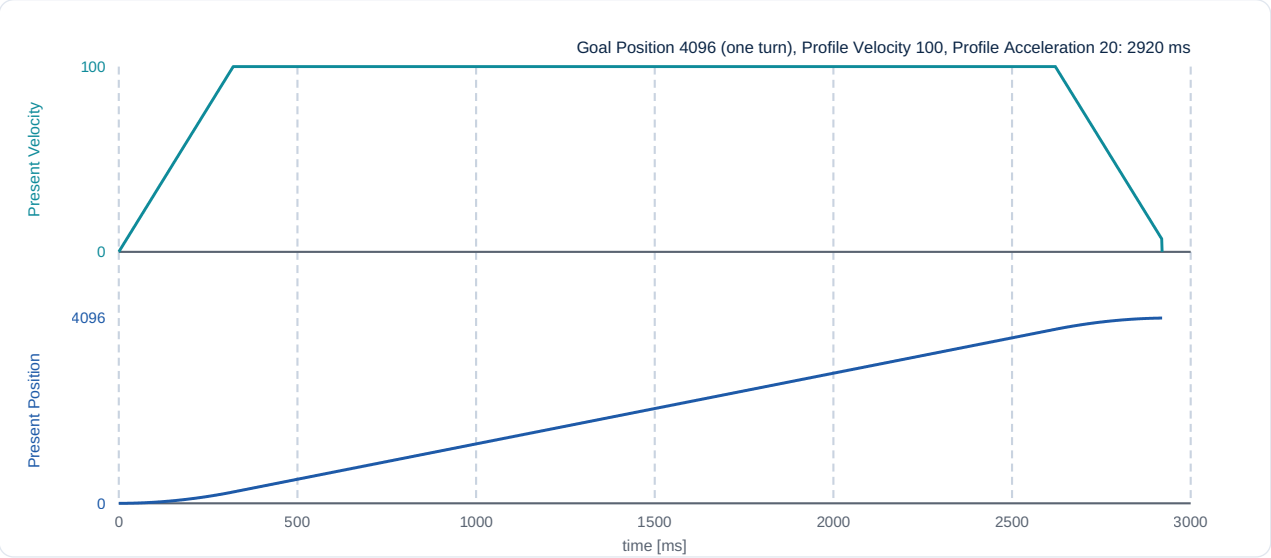
$$u_v = 0.229 \ \frac{\text{rev}}{\text{min}} = 15{,}633 \ \frac{\text{mpulse}}{\text{s}}$$
$$u_a = 214.577 \ \frac{\text{rev}}{\text{min}^2} = 244{,}140 \ \frac{\text{mpulse}}{\text{s}^2}$$

Position modes (3, 4, 5)

With g the Goal Position on the shaft ((Goal Position − Homing Offset)× direction $\times 1000$), V the profile speed (Profile Velocity $\times u_v$, or Velocity Limit $\times u_v$ when Profile Velocity is 0) and A = Profile Acceleration $\times u_a$, each step k computes

$$
\begin{aligned}
e_k &= g - s_k, \qquad d_k = |e_k| \\
r_k &= \min\left(1000\, d_k,\ \sqrt{2A \min\left(d_k,\ 10^9\right)},\ V\right) \\
v_{k+1} &= v_k + \operatorname{clamp}\left(\operatorname{sgn}(e_k)\, r_k - v_k,\ -\tfrac{A}{1000},\ \tfrac{A}{1000}\right) \\
s_{k+1} &= s_k + \left\lfloor v_{k+1}/1000 \right\rceil_0
\end{aligned}
$$

where $1000\, d_k$ is the speed that arrives within one step, the square root the fastest speed that can still brake to standstill at the goal, and $\lfloor \cdot \rceil_0$ a division truncated toward zero. With $A = 0$ the velocity jumps to its target. When the remaining distance is not larger than the step (or below half a pulse) the shaft is placed on the goal and $v = 0$. The square root is an integer square root (at most 32 iterations). Drive Mode bit 2 (time-based profile) is treated like the velocity-based profile.



The curve is computed by `tools/docs/diagrams.py` with the same integer arithmetic.

Velocity, PWM and current modes (1, 16, 0)

The target velocity is Goal Velocity $\times u_v$ (mode 1), Goal PWM / PWM Limit × Velocity Limit $\times u_v$ (mode 16) or Goal Current / Current Limit × Velocity Limit $\times u_v$ (mode 0), limited to ±Velocity Limit $\times u_v$ and ramped with A as above; the shaft advances by $\left\lfloor v/1000 \right\rceil_0$ per step. With the torque off the velocity is 0 and the shaft holds its position.

Live registers

After the steps of `x430_sim_step()` the simulator writes

Register	Value
Present Position (132)	$\text{round}(s/1000) \times$ direction + Homing Offset
Present Velocity (128)	$\text{round}(v \times \text{direction}/u_v)$

Register	Value
Present Current / Load (126)	Velocity change of the last step in u_v, limited to Current Limit (or ±1000 for Present Load) — a rough stand-in for the effort
Present PWM (124)	Present Velocity / Velocity Limit × 885
Moving (122), Moving Status (123)	1 and Profile Ongoing while the shaft turns or a Goal Position move is not done; In-Position, profile type (step, rectangular, triangular, trapezoidal)
Velocity / Position Trajectory (136 / 140)	The target velocity; Goal Position in the position modes
Present Input Voltage (144), Present Temperature (146)	From `struct x430_sim_config`

Direction is −1 with Drive Mode bit 0 (reverse), +1 otherwise.

Measured

Scenario (test)	Simu-lator	Closed form	Differ-ence
Move of 4,096 pulses, Profile Velocity 100, Profile Acceleration 20 (`TrapezoidalMoveTimeMatchesTheClosedForm`), $T = d/v + v/a$	2,920 ms	2,940.3 ms	−0.69 %
Velocity ramp 0 → 100, Profile Acceleration 5 (`VelocityRampTimeMatchesTheClosedForm`), $t = v/a$	1,275 ms	1,280.7 ms	−0.45 %

The peak Present Velocity of the move is exactly the Profile Velocity (100). The move ends slightly early because the last steps place the shaft on the goal as soon as it is closer than one step. The tests accept ±1 %.

7.9 Limits of the emulation

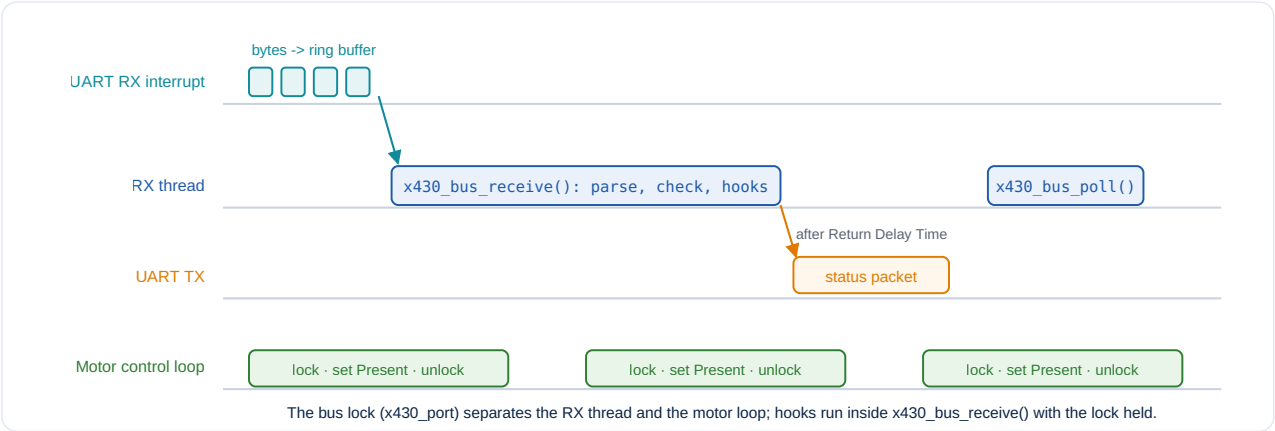
- Motion, current control and PID gains belong to the motor: the library stores the gains and limits, the application applies them.
- Present Input Voltage and Present Temperature are not compared with their limits by the library: the application raises the hardware error.
- Only Protocol 2.0; Protocol Type accepts 2 only.
- One model per build; all devices of a build emulate the same model.

CHAPTER 8

Integration Guide

The X430 emulator does not touch hardware. An integration has four parts:

1. **Input:** the bytes received on the DYNAMIXEL line (TTL half duplex or RS-485), passed to `x430_bus_receive()` in any chunking.
2. **Output:** the byte writer of `struct x430_bus_config`, which sends one complete status packet (and switches the direction of a half-duplex line).
3. **The motor:** the hooks of each device and the live register updates.
4. **Contexts and storage:** the port (lock, millisecond clock, delay) and the persistent blocks.



The bus lock (x430_port) separates the RX thread and the motor loop; hooks run inside x430_bus_receive() with the lock held.

8.1 Timing

Item	Requirement
`x430_bus_receive()`	As soon as bytes arrive; a status packet is sent from inside the call
Return Delay Time (9)	Waited through `x430_port::delay_us` (default 250 × 2 µs = 500 µs)
`x430_bus_poll()`	Every 5–20 ms (Bus Watchdog unit 20 ms)
Live register updates	At the rate of the control loop, between `x430_bus_lock()` and `x430_bus_unlock()`
Interrupts	Never call a library function from an interrupt handler: the UART interrupt only stores bytes
Stack of the bus context	424 B on Cortex-M4 `-Os` (with any packet size), plus the hooks (Configuration)

8.2 Bare metal

```
static int uart_write(void* context, const uint8_t* data, size_t length)
{
    (void)context;
    return board_uart_write(data, length);
}

static uint32_t millis(void* context)
```

```
{
    (void)context;
    return board_millis();
}

static void delay_us(void* context, uint32_t us)
{
    (void)context;
    board_delay_us(us);
}

static const struct x430_port port = {
    .context = NULL, .lock = NULL, .unlock = NULL, .millis = millis, .delay_us = delay_us}
;

void serve_forever(void)
{
    for (;;)
    {
        uint8_t bytes[64];
        const size_t count = board_uart_read(bytes, sizeof(bytes)); /* from the RX ring */

        (void)x430_bus_receive(&bus, bytes, count);
        if (board_tick_1ms())
        {
            (void)x430_device_set(&joint, X430_ADDR_PRESENT_POSITION, motor_position());
            (void)x430_bus_poll(&bus);
        }
    }
}
```

`bus` and `joint` are set up as in Using the Library (p. 17) with `.port = &port`. Without a lock everything runs in the main loop.

8.3 RTOS

```
UART ISR --bytes--> ring buffer --> RX thread: x430_bus_receive()   (lock held inside)
                                        `- hooks --commands--> queue --> motor thread
motor thread: x430_bus_lock(); x430_device_set(Present ...); x430_bus_unlock();
timer or RX thread: x430_bus_poll() every 5-20 ms
```

The lock must be recursive if a hook can lead to code that takes it again. Zephyr mutexes are recursive:

```
static void lock(void* context)
{
    mutex_lock(context); /* Zephyr: k_mutex_lock(context, K_FOREVER) */
}

static void unlock(void* context)
{
    mutex_unlock(context); /* Zephyr: k_mutex_unlock(context) */
}

static const struct x430_port rtos_port = {
    .context = &bus_mutex, .lock = lock, .unlock = unlock, .millis = port_millis,
    .delay_us = port_delay_us};
```

FreeRTOS: `xSemaphoreTakeRecursive` / `xSemaphoreGiveRecursive`, `xTaskGetTickCount` converted to milliseconds. POSIX: a `pthread_mutex_t` of type `PTHREAD_MUTEX_RECURSIVE` (see `examples/host_sim/x430_host_sim.c`).

8.4 Events

The event `X430_EVENT_BAUD_RATE` arrives after the status packet of the write that changed Baud Rate (8), so the answer leaves at the old rate:

```
static uint8_t on_event(struct x430_device* device, enum x430_event event, void*
user_data)
{
    int32_t code = 0;

    (void)user_data;
    if ((event == X430_EVENT_BAUD_RATE) || (event == X430_EVENT_FACTORY_RESET))
    {
        (void)x430_device_get(device, X430_ADDR_BAUD_RATE, &code);
        board_uart_set_baud((uint8_t)code); /* code 1 = 57600 bps, 3 = 1 Mbps */
    }
    else if ((event == X430_EVENT_BUS_WATCHDOG) || (event == X430_EVENT_TORQUE_SHUTDOWN))
    {
        motor_stop();
    }
    else
    {
        /* reboot, clear, backup restored */
    }
    return DXL2_ERR_NONE;
}
```

8.5 Persistent storage

The EEPROM area (76-byte block) and the backup (328-byte block) are opaque byte blocks with a header and a CRC (format (p. 34)). The device keeps no copy of them: it reads and writes them in parts at an offset, so the hooks map directly onto a memory-mapped flash page or an I²C EEPROM:

```
static int nvm_read(struct x430_device* device, enum x430_nvm_slot slot, uint16_t offset,
                    uint8_t* data, uint16_t length, void* user_data)
{
    (void)device;
    (void)user_data;
    return flash_read((uint32_t)slot, offset, data, length); /* 0, or negative without
data */
}

static int nvm_write(struct x430_device* device, enum x430_nvm_slot slot, uint16_t offset,
                     const uint8_t* data, uint16_t length, void* user_data)
{
    (void)device;
    (void)user_data;
    if (offset == 0U)
    {
        flash_erase((uint32_t)slot); /* a new block starts at offset 0 */
    }
    return flash_write((uint32_t)slot, offset, data, length); /* 0, or negative: Result
Fail */
}
```

A block is read as header, payload in chunks of 16 bytes and CRC (checked before anything is taken from it), then once per register the device takes. It is written as three parts with increasing offsets — header, payload, CRC — so the last part ends at `X430_NVM_EEPROM_SIZE` or `X430_NVM_BACKUP_SIZE`; a block cut short by a reset fails its CRC and the device starts with factory values. To keep the previous block until the new one is complete, write into a second page and switch pages when the last part arrives.

A key-value store that keeps whole records only (Zephyr NVS or settings, for example) needs one buffer of `X430_NVM_BACKUP_SIZE` bytes (`X430_NVM_EEPROM_SIZE` without BACKUP) in the application: collect the parts there and store the record when the last part arrives; for `nvm_read`, load the record into it at offset 0 and copy from it. All devices can share that buffer when they are initialized and served from one context: a device finishes one block (it always starts at offset 0) before it or another device touches the next.

`nvm_write` is called from the bus context after the change. On an MCU with one flash bank an erase stalls the CPU: refuse the write (return a negative value, the host gets Result Fail) while the power stage runs, or copy the parts and write them from another thread.

8.6 Adding the library to a CMake build

```
set(X430_MODEL xm430_w350)
set(COMPILE_DEFINITIONS X430_MAX_DEVICES=2U)  # options of x430_config.h
set(COMPILE_OPTIONS -Os)                      # flags of the library sources
include(path/to/DYNAMIXEL-X430-Control-Table-Emulator/x430.cmake)
target_link_libraries(app PRIVATE x430)
target_link_libraries(x430 PUBLIC zephyr_interface)   # Zephyr applications only
```

With `UNIT_TESTS=ON` in the project, the unit tests of this library are built too (`x430_gtest`) and run with the project's CTest (`ctest -L x430`); the `x430` library itself never contains them.

In a Zephyr module, take DXL2-C from a west project instead of FetchContent with `FETCHCONTENT_SOURCE_DIR_DXL2`, pin it to the tag `x430.cmake` expects (`X430_DXL2_GIT_TAG`), and put DXL2-C's `cmake/` folder first on `CMAKE_MODULE_PATH` (Zephyr has a `version.cmake` of its own).

8.7 Host tools

ROS 2 (`dynamixel_hardware_interface`)

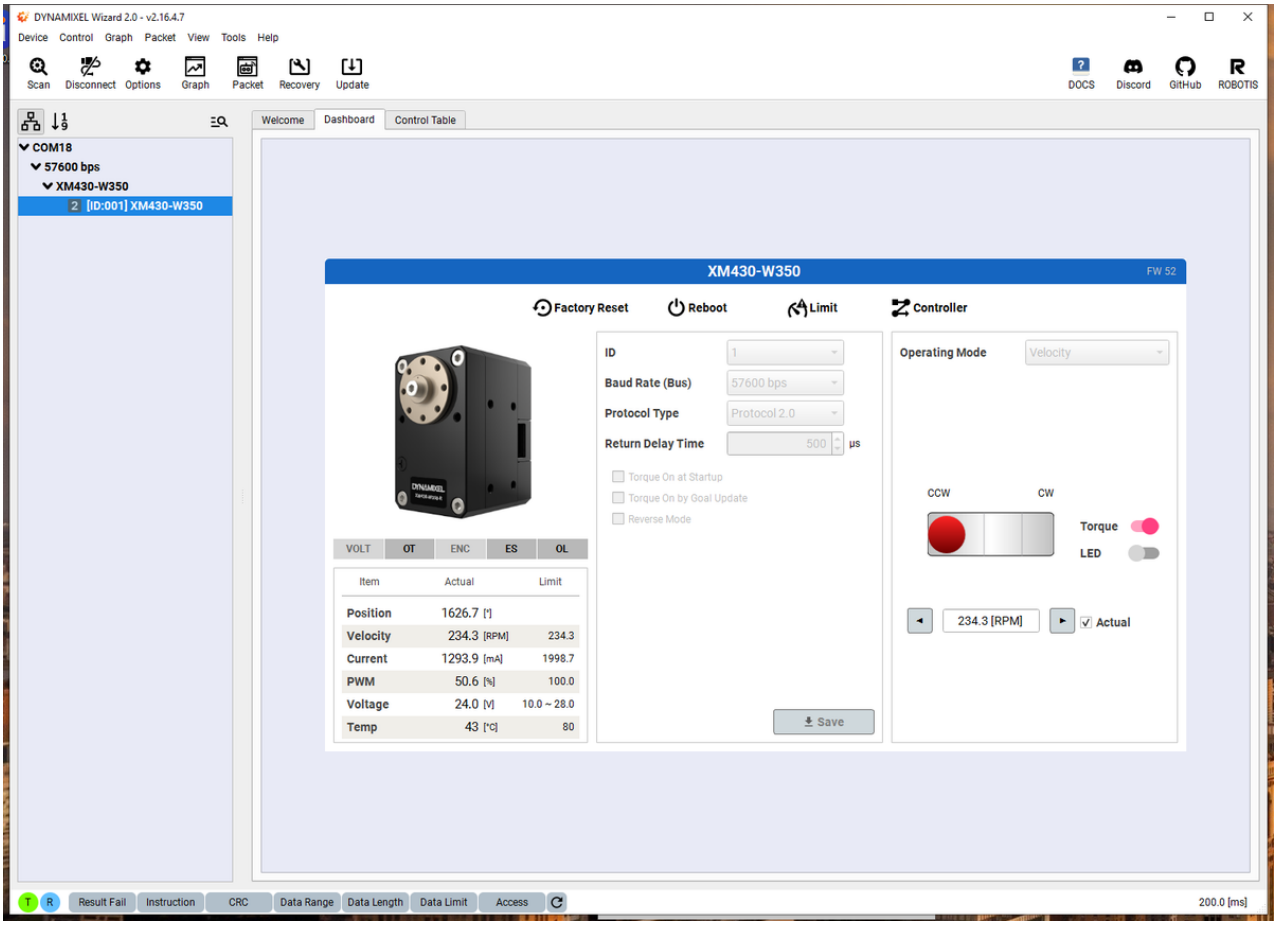
The interface pings each ID, loads the model file by Model Number, writes Torque Enable and Operating Mode, programs Indirect Address blocks and reads them with Fast Sync Read (falling back to Sync Read). All of this is emulated. Points to check:

- The Model Number must exist in its `param/dxl_model/dynamixel.model`. The XW430-T200 (1,280) is not listed there: use `CONFIG_X430_MODEL_NUMBER_OVERRIDE=1270` or add a model file.
- Fast reads answer for the IDs of this bus only: keep all IDs of one hardware component on the same emulator bus.
- The Fast Sync Read answer must fit `DXL2_MAX_PACKET_SIZE`: `10 + devices × (bytes + 4)`.

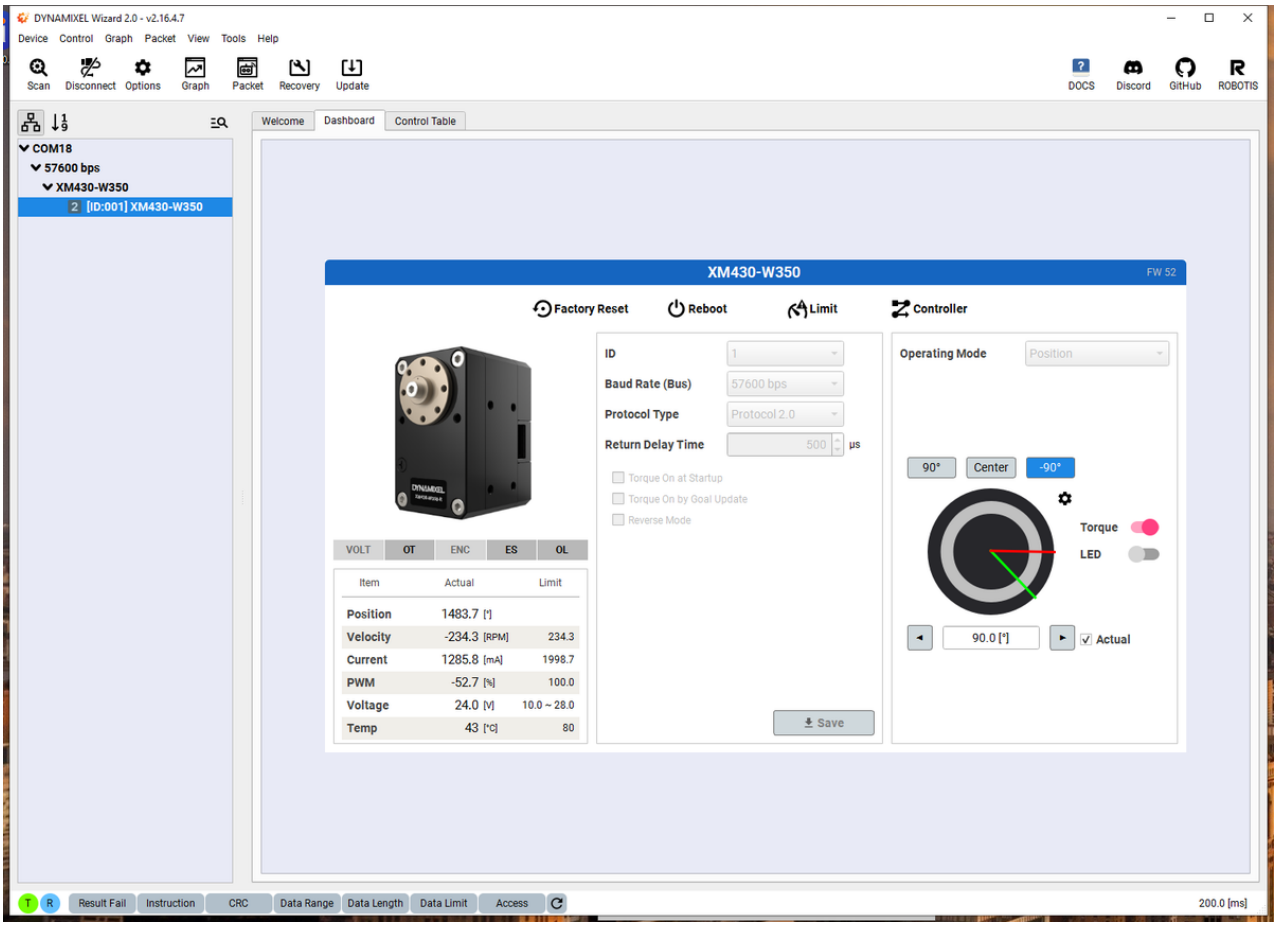
DYNAMIXEL Wizard 2.0

The factory baud rate is 57,600 bps, as on an X430, so the default scan finds the device. After the scan the Dashboard and the Control Table tab read the whole Control Table (addresses 0–661) in one READ of 662 bytes, which the default `DXL2_MAX_PACKET_SIZE` of 768 allows; below that the Wizard shows TIME OUT (Packet size (p. 25)).

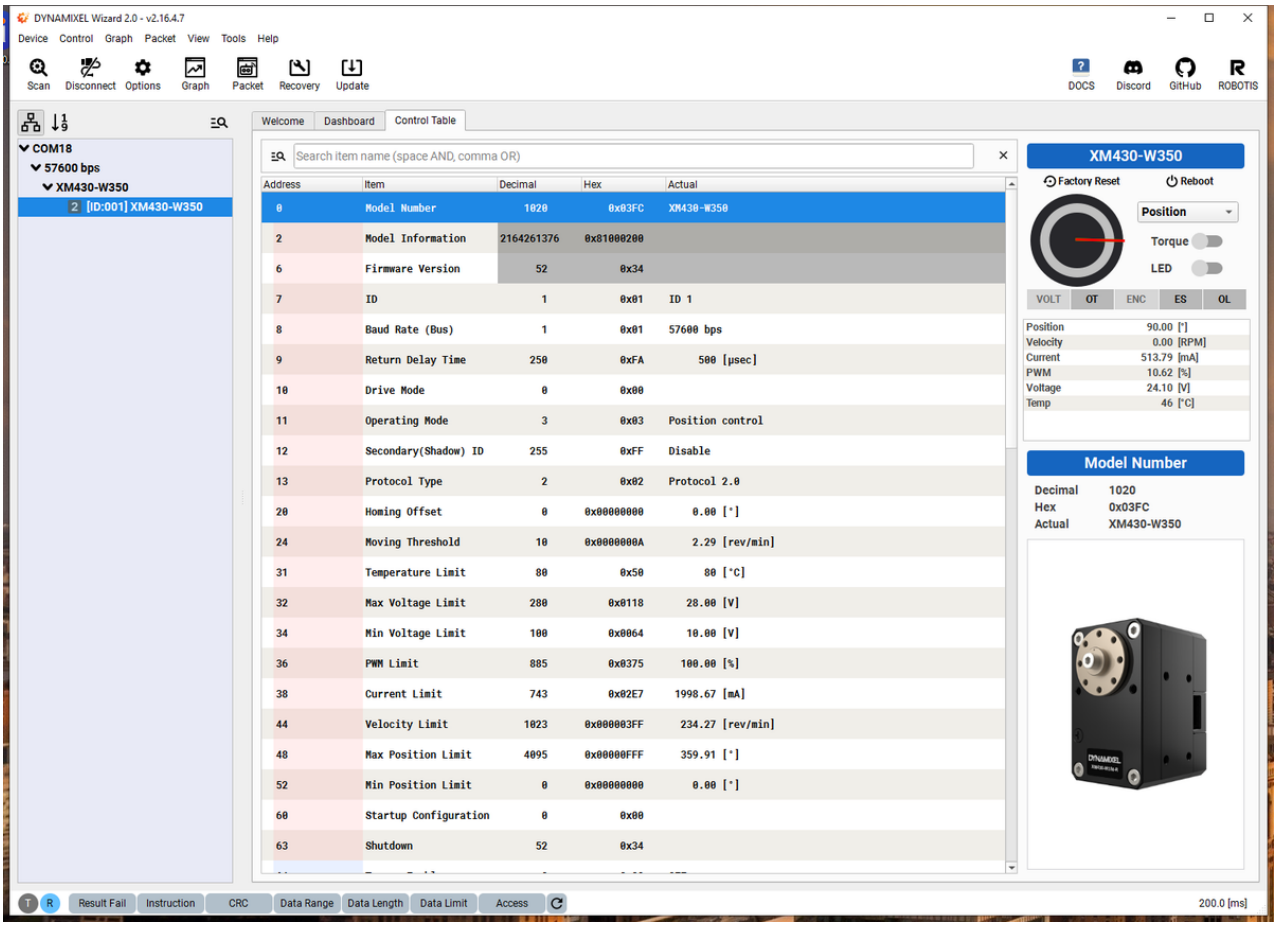
The screenshots below were taken with DYNAMIXEL Wizard 2.0 (v2.16.4.7) and a stepper joint firmware built on this library (an STM32G431 board with a 17HS19-2004S1 stepper motor at 24 V, emulating an XM430-W350, Model Information = firmware version, voltage limits raised with an application override (p. 25)):



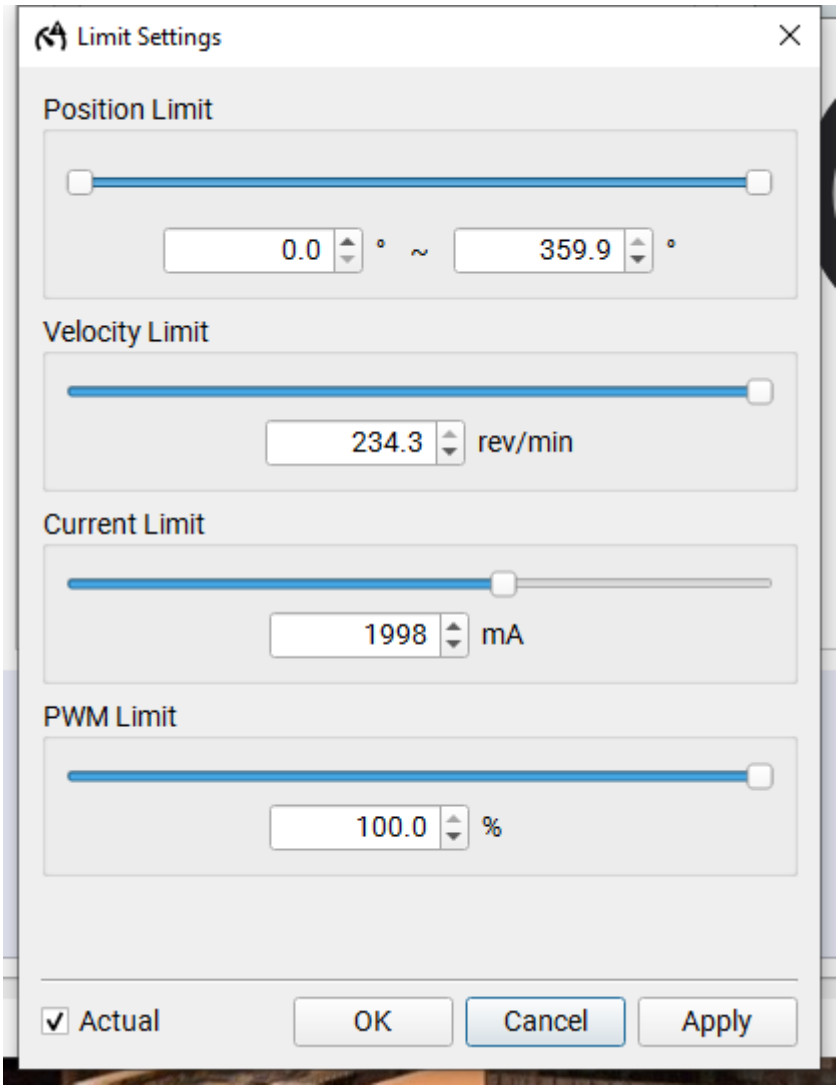
Velocity mode at the Velocity Limit. Present Current and Present PWM are written by the firmware from its current measurement and duty cycle.



Position mode during a move: the sign of Present Velocity and Present PWM follows the direction.



Control Table: the EEPROM area (pink addresses) is locked while the torque is on, as on an XM430.



Limit Settings with the factory values of that firmware.

Without hardware: the host simulator

`examples/host_sim/x430_host_sim.c` serves simulated devices on a POSIX serial port or pseudo-terminal (Linux, macOS, WSL):

```
cmake --preset "X430[Examples]" && cmake --build "build/X430[Examples]"
"build/X430[Examples]/examples/x430_host_sim" --pty --link /tmp/ttyDXL --ids 1,2,3
# point DynamixelSDK or ROS 2 at /tmp/ttyDXL
```

8.8 Bring-up checklist

1. PING with DynamixelSDK or the Wizard at 57,600 bps: the Model Number and Firmware Version come back.
2. READ the whole table (Wizard Control Table tab): no TIME OUT.
3. WRITE Torque Enable 1 and 0: the write hook switches the motor; Goal Position takes Present Position.
4. Move in Position mode: Present Position and Moving follow the motor.

5. Write an EEPROM register (Return Delay Time) with the torque off, power-cycle: the value survives (`nvm_write` / `nvm_read`).
6. Set Bus Watchdog to 10 and stop the host: after 200 ms the motor stops and Bus Watchdog reads −1.
7. Raise a hardware error with its Shutdown bit: the torque goes off, statuses carry the alert bit, REBOOT clears it.
8. Change Baud Rate: the answer comes at the old rate, the next packet at the new one.

CHAPTER 9

API Reference

This chapter documents every public function, type and macro of the X430 emulator. Include one header; it declares every function and includes the public type headers:

```
#include "x430.h"
```

Header	Contents
`x430.h`	All public functions, `x430_sim_hooks`; library version; includes the headers below
`x430_config.h`	Build-time configuration (Kconfig or compiler definitions)
`x430_defs.h`	Status codes, write options, events, persistent slots, Control Table addresses and values
`x430_types.h`	Port, hooks, configuration structures, opaque instances and their sizes
`x430_model.h`	Types of the generated model descriptor, `x430_model_active`
`x430_model_info.h`	Generated: name, Model Number and table size of the selected model
`dxl2.h`	DXL2-C: `DXL2_ERR_*` protocol errors, `dxl2_write_fn`, `struct dxl2_packet`

Everything else (`src/`) is private and not installed with the library.

9.1 Version — `x430.h`

Macro	Value	Meaning
`X430_VERSION_MAJOR`, `…_MINOR`, `…_PATCH`	`0`, `0`, `0`	Version numbers; set by CMake from the latest `vX.Y.Z` tag and stamped into the release package; `0.0.0` means unknown (a header from the repository built without CMake)
`X430_VERSION`	`"0.0.0"`	Version string

9.2 Definitions — `x430_defs.h`

Status codes (return value of every function):

Macro	Value	Meaning
`X430_OK`	`0`	Success
`X430_ERR_PARAM`	`-1`	`NULL` pointer or a value outside its documented range
`X430_ERR_STATE`	`-2`	Instance not initialized; device already attached
`X430_ERR_CONFIG`	`-3`	Bus full, or the ID is already used on the bus
`X430_ERR_HAL`	`-4`	The byte writer of the bus returned an error

Macro	Value	Meaning
`X430_ERR_UNSUPPORTED`	`-5`	The feature is not compiled in

Write options of `x430_device_write()` and `x430_device_reg_write()`: `X430_WRITE_SECONDARY` (0x01, packet addressed by the Secondary ID: the EEPROM area is locked), `X430_WRITE_VALIDATE_ONLY` (0x02, check only, store nothing, call no hook).

Events (`enum x430_event`, argument of the event hook):

Value	When	Return of the hook
`X430_EVENT_REBOOT`	After REBOOT: RAM area reset, errors cleared	Ignored
`X430_EVENT_FACTORY_RESET`	After FACTORY_RESET (then like REBOOT)	Ignored
`X430_EVENT_CLEAR_MULTI_TURN`	CLEAR while the device is not moving	`DXL2_ERR_NONE` after bringing Present Position into one turn, or an error for the host
`X430_EVENT_BUS_WATCHDOG`	Bus Watchdog expired (Bus Watchdog now reads −1)	Ignored
`X430_EVENT_BAUD_RATE`	Baud Rate (8) changed; after the status packet	Ignored
`X430_EVENT_TORQUE_SHUTDOWN`	A hardware error in Shutdown (63) switched the torque off	Ignored
`X430_EVENT_BACKUP_RESTORED`	BACKUP restore copied the backup into the table	Ignored

Persistent slots (`enum x430_nvm_slot`): `X430_NVM_EEPROM` (0, `X430_NVM_EEPROM_SIZE` = 76 bytes), `X430_NVM_BACKUP` (1, `X430_NVM_BACKUP_SIZE` = 328 bytes); `X430_NVM_HEADER_SIZE` = 10.

Control Table constants: `X430_ADDR_*` (addresses of the registers used by the library and by applications, for example `X430_ADDR_GOAL_POSITION` = 116, `X430_RAM_START` = 64), `X430_MODE_*` (operating modes), `X430_DRIVE_MODE_*`, `X430_STARTUP_*`, `X430_HW_ERROR_*`, `X430_MOVING_STATUS_*`, the instruction options `X430_FACTORY_RESET_*`, `X430_CLEAR_MULTI_TURN`, `X430_BACKUP_*`, and `X430_BUS_WATCHDOG_ERROR`, `X430_BUS_WATCHDOG_UNIT_MS` (20), `X430_SECONDARY_ID_DISABLED` (253), `X430_EXTENDED_POSITION_LIMIT` (1,048,575), `X430_REALTIME_TICK_PERIOD` (32,768), `X430_RETURN_DELAY_UNIT_US` (2).

9.3 Types — `x430_types.h`

Type	Members	Meaning
`struct x430_port`	`context`, `lock`, `unlock`, `millis`, `delay_us`	Platform services (Configuration)
`struct x430_hooks`	`write`, `read`, `event`, `nvm_read`, `nvm_write`	Application hooks of a device (Using the Library)
`struct x430_bus_config`	`write`, `write_context`, `port`	Bus configuration

Type	Members	Meaning
`struct x430_device_config`	`id`, `firmware_version`, `model_information`, `hooks`, `user_data`	Device configuration
`struct x430_sim_config`	`input_voltage`, `temperature`	Simulator configuration

Callback types:

Type	Signature	Return
`x430_lock_fn`	`void (*)(void* context)`	–
`x430_millis_fn`	`uint32_t (*)(void* context)`	Milliseconds, wrapping at 2^32
`x430_delay_fn`	`void (*)(void* context, uint32_t us)`	–
`x430_write_hook`	`uint8_t (*)(struct x430_device* device, uint16_t address, uint8_t size, int32_t value, void* user_data)`	`DXL2_ERR_NONE` to accept, a `DXL2_ERR_*` code to refuse
`x430_read_hook`	`void (*)(struct x430_device* device, uint16_t address, uint16_t length, void* user_data)`	–
`x430_event_hook`	`uint8_t (*)(struct x430_device* device, enum x430_event event, void* user_data)`	See the events above
`x430_nvm_read_hook`	`int (*)(struct x430_device* device, enum x430_nvm_slot slot, uint16_t offset, uint8_t* data, uint16_t length, void* user_data)`	0 when `length` bytes from `offset` were read, negative without data
`x430_nvm_write_hook`	`int (*)(struct x430_device* device, enum x430_nvm_slot slot, uint16_t offset, const uint8_t* data, uint16_t length, void* user_data)`	0, or negative (the remaining parts are not written, the host gets Result Fail)

Instances. `struct x430_bus`, `struct x430_device` and `struct x430_sim` are opaque arrays of `X430_BUS_WORDS`, `X430_DEVICE_WORDS` and `X430_SIM_WORDS` 64-bit words (1,664 B, 384 B and 32 B on Cortex-M4 with the default options). Allocate them, pass their addresses, never read or write their content.

9.4 Variables — `x430.h`

```
extern const struct x430_hooks x430_sim_hooks;
```

The hooks of the simulated actuator: pass them with `x430_device_config::user_data` pointing at an initialized `struct x430_sim`. All members are `NULL` when `X430_FEATURE_SIMULATOR` is 0.

9.5 Functions — `x430.h`

Every function except `x430_bus_init()`, `x430_device_init()` and `x430_sim_init()` returns `X430_ERR_STATE` for an instance without a successful initialization and then calls no hook. Functions marked *lock-free* do not take the bus lock: call them from hooks or between `x430_bus_lock()` and `x430_bus_unlock()`.

Bus

x430_bus_init

```
int x430_bus_init(struct x430_bus* bus, const struct x430_bus_config* config);
```

Initialize a bus. Calls no callback except x430_port::millis.

Parameter	Description
`bus`	Bus.
`config`	Configuration; copied (the port it points at is not).

Return	Meaning
`X430_OK`	Bus ready.
`X430_ERR_PARAM`	`NULL` pointer or no writer.

x430_bus_attach

```
int x430_bus_attach(struct x430_bus* bus, struct x430_device* device);
```

Attach an initialized device to an initialized bus. Takes the bus lock.

Parameter	Description
`bus`	Bus.
`device`	Device, not attached to any bus.

Return	Meaning
`X430_OK`	Attached.
`X430_ERR_PARAM`	`NULL` pointer.
`X430_ERR_STATE`	Bus or device not initialized, or the device is attached.
`X430_ERR_CONFIG`	Bus full (`X430_MAX_DEVICES`) or the ID is used on the bus.

x430_bus_receive

```
int x430_bus_receive(struct x430_bus* bus, const uint8_t* data, size_t length);
```

Feed received bytes in any chunking; answers every complete instruction packet. Takes the bus lock; runs the hooks.

Parameter	Description
`bus`	Bus.

Parameter	Description
`data`	Received bytes (may be `NULL` when `length` is 0).
`length`	Number of bytes.

Return	Meaning
`X430_OK`	Bytes processed.
`X430_ERR_PARAM`	`NULL` bus, or `NULL` data with a non-zero length.
`X430_ERR_STATE`	Bus not initialized.
`X430_ERR_HAL`	The writer failed for at least one status packet.

x430_bus_handle_packet

```
int x430_bus_handle_packet(struct x430_bus* bus, const struct dxl2_packet* packet);
```

Handle one decoded instruction packet, for applications that run their own DXL2-C link. Takes the bus lock; runs the hooks.

Parameter	Description
`bus`	Bus.
`packet`	Packet decoded by DXL2-C. Status packets are ignored.

Return	Meaning
`X430_OK`	Success
`X430_ERR_PARAM`	`NULL` pointer
`X430_ERR_STATE`	Bus not initialized
`X430_ERR_HAL`	The writer failed

x430_bus_poll

```
int x430_bus_poll(struct x430_bus* bus);
```

Periodic service: Bus Watchdog of every device. Call every few milliseconds (the watchdog unit is 20 ms). Takes the bus lock.

Parameter	Description
`bus`	Bus.

Return	Meaning
`X430_OK`	Success

Return	Meaning
`X430_ERR_PARAM`	`NULL`
`X430_ERR_STATE`	Not initialized

x430_bus_lock

```
int x430_bus_lock(struct x430_bus* bus);
```

Take the bus lock (x430_port::lock; nothing without it). Use it around the live register updates of the application.

Parameter	Description
`bus`	Bus.

Return	Meaning
`X430_OK`	Success
`X430_ERR_PARAM`	`NULL`
`X430_ERR_STATE`	Not initialized

x430_bus_unlock

```
int x430_bus_unlock(struct x430_bus* bus);
```

Release the bus lock (x430_port::unlock).

Parameter	Description
`bus`	Bus.

Return	Meaning
`X430_OK`	Success
`X430_ERR_PARAM`	`NULL`
`X430_ERR_STATE`	Not initialized

x430_bus_find

```
int x430_bus_find(const struct x430_bus* bus, uint8_t id, struct x430_device** device);
```

Attached device with the primary ID `id` (lock-free).

Parameter	Description
`bus`	Bus.
`id`	DYNAMIXEL ID.
`device`	Set to the device, or to `NULL` when no device has that ID.

Return	Meaning
`X430_OK`	Found.
`X430_ERR_PARAM`	`NULL` pointer, or no attached device has the ID.
`X430_ERR_STATE`	Bus not initialized.

Device: lifecycle and lock-free access

x430_device_init

```
int x430_device_init(struct x430_device* device, const struct x430_device_config* config);
```

Initialize a device: factory values, stored EEPROM area (nvm_read), the RAM area and the Startup Configuration (which may call the write hook). Initialize a device before x430_bus_attach(), never while a bus serves it: the content of the instance before the call is not trusted (it may be the garbage of a stack instance), so an attachment cannot be detected.

Parameter	Description
`device`	Device, not attached to a bus.
`config`	Configuration; copied (the hooks it points at are not).

Return	Meaning
`X430_OK`	Device ready.
`X430_ERR_PARAM`	`NULL` pointer or ID above 252.

x430_device_get

```
int x430_device_get(const struct x430_device* device, uint16_t address, int32_t* value);
```

Value of the register that starts at `address` (lock-free).

Parameter	Description
`device`	Device.
`address`	First address of a register.
`value`	Sign-extended value.

Return	Meaning
`X430_OK`	Value read.
`X430_ERR_PARAM`	`NULL` pointer, or no register starts at `address`.
`X430_ERR_STATE`	Device not initialized.

x430_device_set

```
int x430_device_set(struct x430_device* device, uint16_t address, int32_t value);
```

Store a value into the register at `address`, ignoring access rights (lock-free). For live registers (Present Position, Moving, ...): no hook is called, no range is checked.

Parameter	Description
`device`	Device.
`address`	First address of a register.
`value`	Value; truncated to the register size.

Return	Meaning
`X430_OK`	Value stored.
`X430_ERR_PARAM`	`NULL` pointer, or no register starts at `address`.
`X430_ERR_STATE`	Device not initialized.

x430_device_id

```
int x430_device_id(const struct x430_device* device, uint8_t* id);
```

Current ID (7) of the device (lock-free).

Parameter	Description
`device`	Device.
`id`	ID.

Return	Meaning
`X430_OK`	Success
`X430_ERR_PARAM`	`NULL`
`X430_ERR_STATE`	Not initialized

x430_device_raise_hardware_error

```
int x430_device_raise_hardware_error(struct x430_device* device, uint8_t bits);
```

Report hardware errors (Hardware Error Status bits, lock-free). The bits stay set until REBOOT. Bits enabled in Shutdown (63) switch the torque off (event `X430_EVENT_TORQUE_SHUTDOWN`) and keep it off until REBOOT.

Parameter	Description
`device`	Device.
`bits`	`X430_HW_ERROR_*` bits.

Return	Meaning
`X430_OK`	Success
`X430_ERR_PARAM`	`NULL`
`X430_ERR_STATE`	Not initialized

x430_device_flush_events

```
int x430_device_flush_events(struct x430_device* device);
```

Deliver deferred events (`X430_EVENT_BAUD_RATE`) after the status packet. The bus calls it; call it after x430_device_write() of your own.

Parameter	Description
`device`	Device.

Return	Meaning
`X430_OK`	Success
`X430_ERR_PARAM`	`NULL`
`X430_ERR_STATE`	Not initialized

Device: instruction semantics

x430_device_read

```
int x430_device_read(struct x430_device* device, uint16_t address, uint16_t length,
uint8_t* data,
                     uint8_t* error);
```

READ semantics: read hook, Indirect Data, Realtime Tick.

Parameter	Description
`device`	Device.
`address`	First address.
`length`	Bytes to read.
`data`	Destination of `length` bytes.
`error`	Protocol result: `DXL2_ERR_NONE`, `DXL2_ERR_DATA_LENGTH` (length 0) or `DXL2_ERR_ACCESS` (beyond the table).

Return	Meaning
`X430_OK`	Success
`X430_ERR_PARAM`	`NULL`
`X430_ERR_STATE`	Not initialized

x430_device_write

```
int x430_device_write(struct x430_device* device, uint16_t address, uint16_t length,
                      const uint8_t* data, uint8_t options, uint8_t* error);
```

WRITE semantics: checks every touched register, then calls the write hook and stores in address order; stores the EEPROM area when it changed.

Parameter	Description
`device`	Device.
`address`	First address.
`length`	Bytes to write.
`data`	`length` bytes.
`options`	`X430_WRITE_SECONDARY`, `X430_WRITE_VALIDATE_ONLY`.
`error`	Protocol result: `DXL2_ERR_NONE` or the `DXL2_ERR_*` of the first failure.

Return	Meaning
`X430_OK`	Success
`X430_ERR_PARAM`	`NULL`
`X430_ERR_STATE`	Not initialized

x430_device_reg_write

```
int x430_device_reg_write(struct x430_device* device, uint16_t address, uint16_t length,
                          const uint8_t* data, uint8_t options, uint8_t* error);
```

REG_WRITE: validate a write and register it for ACTION.

Parameter	Description
`device`	Device.
`address`	First address.
`length`	Bytes, at most `X430_REG_WRITE_SIZE`.
`data`	`length` bytes.
`options`	`X430_WRITE_SECONDARY`.
`error`	Protocol result.

Return	Meaning
`X430_OK`	Success
`X430_ERR_PARAM`	`NULL`
`X430_ERR_STATE`	Not initialized

x430_device_action

```
int x430_device_action(struct x430_device* device, uint8_t* error);
```

ACTION: execute the registered write.

Parameter	Description
`device`	Device.
`error`	Protocol result; `DXL2_ERR_INSTRUCTION` when nothing is registered.

Return	Meaning
`X430_OK`	Success
`X430_ERR_PARAM`	`NULL`
`X430_ERR_STATE`	Not initialized

x430_device_reboot

```
int x430_device_reboot(struct x430_device* device);
```

REBOOT: RAM area reset, errors cleared, Startup Configuration applied, event `X430_EVENT_REBOOT`.

Parameter	Description
`device`	Device.

Return	Meaning
`X430_OK`	Success
`X430_ERR_PARAM`	`NULL`
`X430_ERR_STATE`	Not initialized

x430_device_factory_reset

```
int x430_device_factory_reset(struct x430_device* device, uint8_t option, uint8_t* error);
```

FACTORY_RESET, then like REBOOT; event `X430_EVENT_FACTORY_RESET`.

Parameter	Description
`device`	Device.
`option`	`X430_FACTORY_RESET_ALL`, `X430_FACTORY_RESET_EXCEPT_ID` or `X430_FACTORY_RESET_EXCEPT_ID_BAUD`.
`error`	Protocol result; `DXL2_ERR_DATA_RANGE` for another option.

Return	Meaning
`X430_OK`	Success
`X430_ERR_PARAM`	`NULL`
`X430_ERR_STATE`	Not initialized

x430_device_clear

```
int x430_device_clear(struct x430_device* device, uint8_t option, uint8_t* error);
```

CLEAR with option `X430_CLEAR_MULTI_TURN` (event `X430_EVENT_CLEAR_MULTI_TURN`).

Parameter	Description
`device`	Device.
`option`	`X430_CLEAR_MULTI_TURN`.
`error`	Protocol result: `DXL2_ERR_RESULT_FAIL` while moving, `DXL2_ERR_DATA_RANGE` for another option.

Return	Meaning
`X430_OK`	Success
`X430_ERR_PARAM`	`NULL`
`X430_ERR_STATE`	Not initialized

x430_device_backup

```
int x430_device_backup(struct x430_device* device, uint8_t option, uint8_t* error);
```

Control Table BACKUP: store, or restore with the torque off.

Parameter	Description
`device`	Device.
`option`	`X430_BACKUP_STORE` or `X430_BACKUP_RESTORE`.
`error`	Protocol result.

Return	Meaning
`X430_OK`	Success
`X430_ERR_PARAM`	`NULL`
`X430_ERR_STATE`	Not initialized
`X430_ERR_UNSUPPORTED`	`X430_FEATURE_BACKUP` is 0

Simulator

x430_sim_init

```
int x430_sim_init(struct x430_sim* sim, struct x430_device* device,
                  const struct x430_sim_config* config);
```

Prepare a simulator for `device`. Call it before x430_device_init() of a device whose hooks are x430_sim_hooks and whose user data is `sim`.

Parameter	Description
`sim`	Simulator.
`device`	Device it simulates (may be initialized later).
`config`	Configuration, `NULL` for the defaults.

Return	Meaning
`X430_OK`	Success
`X430_ERR_PARAM`	`NULL` sim or device
`X430_ERR_UNSUPPORTED`	`X430_FEATURE_SIMULATOR` is 0

x430_sim_step

```
int x430_sim_step(struct x430_sim* sim, uint32_t elapsed_ms);
```

Advance the simulation by `elapsed_ms` one-millisecond steps and refresh the live registers. Takes the bus lock of the device when it is attached. The time is proportional to `elapsed_ms`: call it periodically.

Parameter	Description
`sim`	Simulator.
`elapsed_ms`	Time since the previous call [ms].

Return	Meaning
`X430_OK`	Success
`X430_ERR_PARAM`	`NULL`
`X430_ERR_STATE`	Simulator or device not initialized
`X430_ERR_UNSUPPORTED`	`X430_FEATURE_SIMULATOR` is 0

9.6 Return codes summary

Function	OK	ERR_PARAM	ERR_STATE	ERR_CONFIG	ERR_HAL	ERR_UNSUPPORTED
`x430_bus_init()`, `x430_device_init()`	✓	✓				
`x430_bus_attach()`	✓	✓	✓	✓		
`x430_bus_receive()`, `x430_bus_handle_packet()`	✓	✓	✓		✓	
`x430_device_backup()`	✓	✓	✓			✓
`x430_sim_init()`	✓	✓				✓
`x430_sim_step()`	✓	✓	✓			✓
Every other function	✓	✓	✓			

CHAPTER 10

MISRA C:2012 compliance

This page records how the library is checked against MISRA C:2012 and every deviation from it. It covers the library code only: `src/`, `inc/` and `x430.h`. The generated model table is constant data written by `tools/x430_gen_model.py`; the host example and the tests are not part of the library.

10.1 Checking

Check	Where	Gate
Cppcheck MISRA C:2012 addon	`misra-analyze`, CI *MISRA C:2012 Analysis*	Fails on any violation
Cppcheck, all checks	`cppcheck-analyze`, CI *Cppcheck Analysis*	Report
CodeQL `cpp-queries`	`codeql-analyze`, CI *CodeQL Analysis*	Report
Compiler	`-std=c11 -Wall -Wextra -Wpedantic -Wconversion` (`-Werror` in the unit tests)	No warnings
Sanitizers	AddressSanitizer and UndefinedBehaviorSanitizer on both unit test presets	No finding
Manual review	Pull request review	Reviewer

The analysis sees the sources without the Kconfig header, so Cppcheck checks every configuration of the `#if` blocks; the generated model header and the DXL2-C headers are given to it as copies in `<build>/x430/analysis-include`.

10.2 Current status

Item	Result
Cppcheck MISRA addon	0 violations (Cppcheck 2.13.0, Ubuntu 24.04)
Unit test coverage of `src/`	100 % lines and branches, `X430[UnitTests]` and `X430[UnitTestsMinimal]`
Deviations	2 (DEV-01, DEV-02)

10.3 Manual review notes

Guideline	How the library complies
Dir 4.1 (run-time failures)	Every table access is bounded by the table size: addresses and lengths of instructions are checked before use, Indirect Address targets outside the table are ignored, packet parameters are read only within their length; sanitizers in CI.
Dir 4.11 (validity of arguments)	Every public function checks its pointers and the initialization of the instance before any access.

Guideline	How the library complies
Dir 4.12, Rule 21.3 (dynamic memory)	No heap.
Rule 8.9 (objects at block scope)	No file-scope object: the only constant data are the model table and `x430_sim_hooks`, both used from several functions or by the application.
Rule 10.4 (essential types)	Build-time options may be signed literals on the command line; the code uses the unsigned copies `X430_MAX_DEVICES_U`, `X430_REG_WRITE_SIZE_U`, `X430_BAUD_RATE_CODE_MAX_U`.
Rule 15.5 (single point of exit)	One `return` per function.
Rule 17.2 (recursion)	No recursion; `tools/docs/stack_usage.py` fails on a call cycle.
Rule 21.6 (standard I/O)	No `<stdio.h>`; no `<string.h>` either (byte copies are explicit loops).

10.4 Deviation register

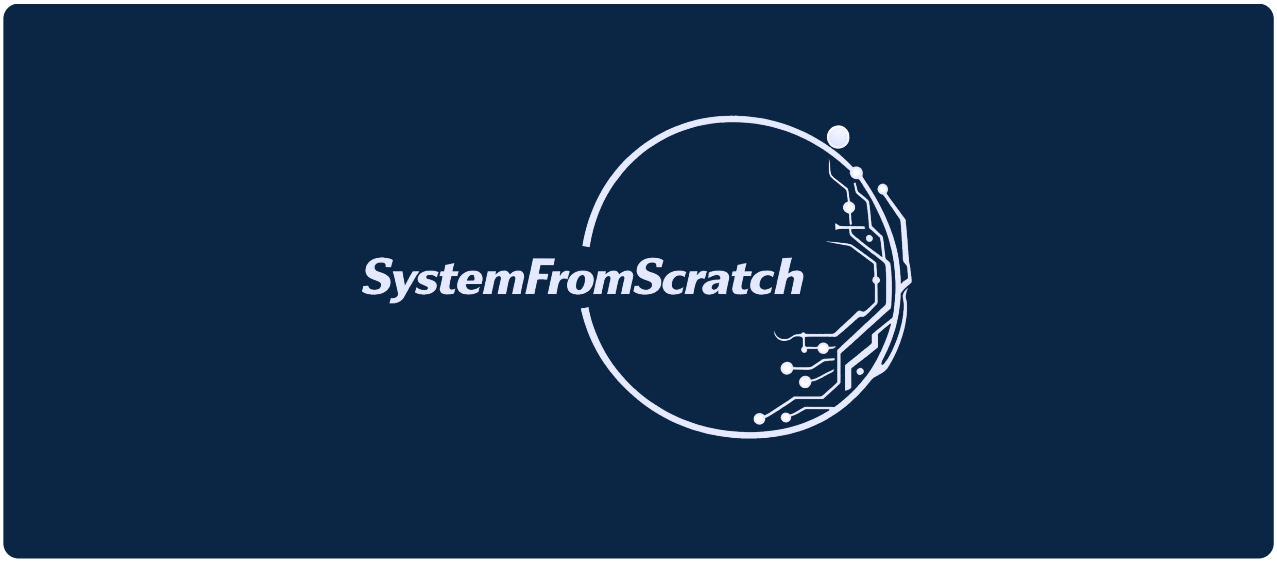
DEV-01: Rule 11.3 (required), opaque instances

- **Location:** `x430_bus_ctx_of()`, `x430_bus_ctx_of_const()`, `x430_device_ctx_of()`, `x430_device_ctx_of_const()` and `x430_sim_ctx_of()` in `src/x430_internal.h`, the only casts from the public instance types to the private state types. Marked with `cppcheck-suppress misra-c2012-11.3`.
- **Rationale:** The application allocates the instances (no heap, no global state), so it must know their size, but not their layout. The public `struct x430_bus`, `struct x430_device` and `struct x430_sim` are arrays of `uint64_t` of that size; the private `struct x430_bus_ctx`, `struct x430_device_ctx` and `struct x430_sim_ctx` hold the state. The internal types stay out of the application code and can change without breaking the API.
- **Risk control:** `x430_bus.c`, `x430_device.c` and `x430_sim.c` check at compile time that each public type is at least as large and as aligned as its private state, and that the DXL2-C link has the size the public header assumes; a unit test checks that the public size wastes less than one word. Every public function converts the pointer before any member access and checks it for NULL and for a successful initialization (`ready`).

DEV-02: Rule 11.5 (advisory), context of a callback

- **Location:** `x430_handler()` in `src/x430_bus.c` (the packet handler given to DXL2-C) and `x430_sim_of()` in `src/x430_sim.c` (the user data of the simulator hooks). Marked with `cppcheck-suppress misra-c2012-11.5`.
- **Rationale:** DXL2-C and the hook interface pass a `void*` context back to the callback, the usual C way to give a callback its object without global state. The library converts it back to the type it registered.
- **Risk control:** The bus registers its own private state as the DXL2-C context in `x430_bus_init()`; nothing else can reach these callbacks. The simulator hooks accept only an initialized `struct x430_sim` (the `ready` field is checked; NULL and an uninitialized simulator are ignored), which the application passes as `x430_device_config::user_data` as documented.

Author & Company



DYNAMIXEL-X430-Control-Table-Emulator is designed, written and maintained by **Anton Vynohradov** and published by **SystemFromScratch®** — secure embedded systems and compliance-ready engineering, from concept to certification-ready product.

Author

Anton Vynohradov · ORCID 0009-0005-6965-0911 (orcid.org/0009-0005-6965-0911)

Embedded Systems Engineer · Secure Product Architect · Founder of SystemFromScratch®

Embedded systems built from scratch — from bare metal to Linux, from first schematic to certification.

Anton designs and builds reliable, secure embedded products end to end: firmware, board support packages and drivers, RTOS and Embedded Linux platforms, hardware bring-up, and the security and compliance work needed to ship them. He is based in Poland and works remotely worldwide.

Area	Focus
Embedded software	Bare-metal, RTOS and Linux firmware in C/C++, built to be maintained for years
BSP & drivers	Bootloaders, Zephyr drivers and Linux kernel work for MCUs, SoCs and SoC FPGAs
Security & compliance	Secure boot, threat modelling, MISRA C/C++, EU Cyber Resilience Act (CRA) readiness
Industries	Industrial IoT, automotive, robotics, medical and consumer electronics

Technology: C · C++ · Python · Zephyr RTOS · FreeRTOS · Embedded Linux · Yocto · Buildroot · ARM Cortex-M · STM32 · Microchip FPGA · RISC-V · CAN / ISO-TP · ROS 2 · Docker · Git · CI/CD · MISRA C/C++ · EU CRA

For DYNAMIXEL-X430-Control-Table-Emulator, Anton wrote the library, the model data and its generator, the unit tests and the end-to-end check with DynamixelSDK, the host simulator and this documentation, and verified the library on a stepper joint with DYNAMIXEL Wizard 2.0.

SystemFromScratch®

SystemFromScratch® was founded by Anton Vynohradov. The company combines deep embedded engineering with cybersecurity and certification-focused product development, for teams in industrial IoT, automotive, robotics, medical and consumer electronics that need a partner who understands the whole stack — silicon, firmware, operating system and the regulations that govern them.

Service	What it covers
Hardware design with partners	Component selection, second sourcing and hardware design with partner engineers: proven, available parts instead of one expensive supplier
Hardware bring-up & integration	From the first power-on of a new board to a working system: first boot and validation (STM32, SmartFusion2)
BSP & device drivers	Board support packages, bootloaders (MCUboot) and drivers for Zephyr, the Linux kernel and FPGAs that keep timing exact
RTOS & Embedded Linux	A maintained, secure foundation on Zephyr, FreeRTOS or Embedded Linux (Yocto, Buildroot)
Embedded software development	Devices that talk to each other reliably over CAN, ISO-TP, RS-485, UART, SPI and I²C, even when the hardware underneath changes
System integration	Systems that work end to end and can be tested on a laptop before the hardware exists: software-in-the-loop, Docker, CI/CD, ROS 2
Cybersecurity, CRA & compliance	Secure boot, signed updates, threat modelling and MISRA C:2012 designed in from the start, so EU Cyber Resilience Act conformity is not a last-minute patch

DYNAMIXEL-X430-Control-Table-Emulator is one of the open-source projects in the SystemFromScratch® portfolio: public engineering work that can be inspected line by line, built with the same rules — MISRA C, full test coverage, documented decisions — that the company applies to customer products.

Contact

Channel	Address
Website	systemfromscratch.com
E-mail	contact@systemfromscratch.com
GitHub	github.com/AntonVynohradov
GitHub organization	github.com/SystemFromScratch

Channel	Address
LinkedIn	linkedin.com/in/vynohradov
ORCID	0009-0005-6965-0911 (orcid.org/0009-0005-6965-0911)
Location	Poland — working remotely worldwide

Building a connected product, bringing up new hardware, or preparing for the Cyber Resilience Act? Get in touch through systemfromscratch.com or by e-mail.

Support and feedback

- **Bugs and feature requests:** open an issue on GitHub (github.com/AntonVynohradov/DYNAMIXEL-X430-Control-Table-Emulator/issues). Please include the library version, your compiler and target, the configuration values, and a minimal program or test that shows the problem.
- **Contributions:** see Contributing to the library (p. 22).
- **Commercial support, integration and custom development:** contact SystemFrom-Scratch® through systemfromscratch.com.

Trademarks and acknowledgements

SystemFromScratch® is a registered trademark. The SystemFromScratch® name and logo are used in this documentation to identify the publisher of DYNAMIXEL-X430-Control-Table-Emulator; the Apache License 2.0 grants no rights to use them, and they may not be used to endorse or promote products derived from DYNAMIXEL-X430-Control-Table-Emulator without prior written permission.

DYNAMIXEL and ROBOTIS are trademarks of ROBOTIS Co., Ltd. The X430 model names identify the emulated devices. DYNAMIXEL-X430-Control-Table-Emulator is an independent implementation of the documented Protocol 2.0 and Control Tables; it is not affiliated with or endorsed by ROBOTIS. STM32 is a registered trademark of STMicroelectronics International N.V.; it names the test hardware only.

The Control Table data were transcribed from the ROBOTIS e-Manual; the Protocol 2.0 codec is DXL2-C by the same author.

APPENDIX A

Changelog

All notable changes to this project are documented here. The format follows Keep a Changelog (keepachangelog.com/en/1.1.0/) and versions follow Semantic Versioning (semver.org/).

A.1 1.2.0 - 2026-11-22

Less RAM per bus with DXL2-C 2.0.0. The public API is unchanged; the instance of a bus is smaller, so code built against 1.1.0 must be recompiled.

Changed

- DXL2-C 2.0.0 (was 1.0.0, `X430_DXL2_GIT_TAG`): the parser removes the byte stuffing in place, and the decoded packet points into its receive buffer instead of holding a copy of the parameters. `X430_DXL2_LINK_BYTES` follows the new `struct dxl2_link`.
- `DXL2_MAX_PACKET_SIZE` has one default per build system: `Kconfig.x430` and `x430.cmake` (768). `x430_config.h` no longer sets its own default; a build without Kconfig, without `x430.cmake` and without the definition stops with `#error` instead of using a size DXL2-C may not have been compiled with.

Quality

- Cortex-M4 (arm-none-eabi-gcc 13.2.1, -Os, XM430-W350): 1,664 B RAM per bus with 768-byte packets (was 2,424 B), 640 B with 256-byte packets (was 888 B); 384 B per device and 32 B per simulator as before. Flash 10,119 B with the model table (was 10,191 B), DXL2-C 842 B (was 850 B). Worst-case stack unchanged (424 B, x430_bus_receive()).
- The unit tests (138 CTest tests, 105 without the optional features) with 100 % line and branch coverage of src/, all 17 models, the DynamixelSDK end-to-end check and MISRA C: 2012 (0 violations, Cppcheck 2.13) pass with DXL2-C 2.0.0.

A.2 1.1.0 - 2026-11-15

Several emulator buses on one wire: one bus per MCU, connected to one half-duplex DYNAMIXEL bus (for example one stepper drive per joint), now behave like X series actuators on that wire.

Changed

- SYNC_READ and BULK_READ take turns: a hosted device answers only after the status packet of the ID listed just before it in the request (at once after a device of the same bus). The bus now decodes the status packets of other devices on the wire for this. Before, every hosted device answered at once and IDs of other buses were skipped, so the answers of several buses collided on a shared wire.
- If that status does not arrive within X430_TURN_TIMEOUT_MS, or the host sends its next instruction first, or a hosted device does not answer reads (Status Return Level 0), the remaining hosted devices of the read stay silent.

- FAST_SYNC_READ and FAST_BULK_READ: when the request lists an ID that the bus does not host, the bus stays silent (one bus cannot build the chained answer of several), and the ROBOTIS ROS 2 hardware interface falls back to SYNC_READ. Before, the bus answered for its own IDs only, which the host decoded as a malformed answer. With all IDs on one bus nothing changes.
- struct x430_bus grows by the pending read: 2,424 B on Cortex-M4 with the default options (was 2,368 B); x430_bus.c 3,090 B flash at -Os (was 2,820 B). The stack of every public function is unchanged. Rebuild the application with the new headers.

Added

- X430_TURN_TIMEOUT_MS / CONFIG_X430_TURN_TIMEOUT_MS (default 100 ms, 1..60000): the longest wait for the status of the ID before, from the last progress of the read.

A.3 1.0.0 - 2026-11-14

First release: device-side emulation of DYNAMIXEL X430 actuators over Protocol 2.0. The public API (x430.h and inc/) is stable within major version 1.

Added

- Bus (struct x430_bus): decodes the instruction packets of one serial line with DXL2-C 1.0.0 and answers for up to X430_MAX_DEVICES devices. PING, READ, WRITE, REG_WRITE, ACTION, FACTORY_RESET, REBOOT, CLEAR, BACKUP, SYNC_READ/WRITE, BULK_READ/WRITE, FAST_SYNC_READ and FAST_BULK_READ; broadcast, Secondary ID, Status Return Level, Return Delay Time, Bus Watchdog. x430_bus_handle_packet() for applications with their own DXL2-C link.
- Device (struct x430_device): the Control Table of the model selected at build time with the rules of the real actuator: access rights, ranges and limit registers, EEPROM lock while the torque is on, all-or-nothing checks of multi-register writes, Indirect Address/Data, hardware error shutdown and alert bit, Startup Configuration, Drive Mode "torque on by goal update", Goal Position := Present Position on torque on, factory reset, reboot, multi-turn clear and Control Table backup.
- Application hooks per device (write, read, event, nvm_read, nvm_write) and an optional platform port (lock, millisecond clock, delay). Persistent blocks carry a header (model, format, slot, size) and a CRC-16; a foreign or damaged block is ignored. The device reads and writes them in parts at an offset and keeps no copy of a block in RAM.
- Memory layout for small MCUs: a device keeps a compact image of its Control Table (316 of 662 bytes: the long reserved runs are left out); the bus builds every status packet in one buffer and stuffs it in place (no packet buffer on the stack); the model table has no address map and generates the 56 Indirect Address registers from the indirect blocks. A status whose byte-stuffed form exceeds DXL2_MAX_PACKET_SIZE is answered with a Data Length Error.
- All 17 X430 models (XM430-W350/W210, XH430-W350/W210/V350/V210, XD430-T350/T210, XW430-T333/T200, XL430-W250, 2XL430-W250, XC430-W150/W240/T150BB/T240BB, 2XC430-W250): Control Tables generated from models/*.yaml and checked against the ROBOTIS e-Manual tables. Application overrides of the model (X430_MODEL_OVERRIDES / CONFIG_X430_MODEL_OVERRIDES), for example a supply above 16 V. The generator needs only the Python standard library (PyYAML is used when installed).

- Simulator (struct x430_sim, x430_sim_hooks): a kinematic model with the X430 velocity profiles for tests and for setting up a system without hardware; POSIX host simulator examples/host_sim serving simulated devices on a serial port or pseudo-terminal.
- Configuration with plain compiler definitions (COMPILE_DEFINITIONS) or, when X430_KCONFIG is switched on, Kconfig (Kconfiglib, menuconfig); optional features BULK, FAST_READ, BACKUP and SIMULATOR; COMPILE_OPTIONS for the flags of the library sources. Packets of up to 768 bytes by default, so a host can read the whole Control Table at once (DYNAMIXEL Wizard).
- UNIT_TESTS builds the unit tests also when the library is a submodule of another project (executable x430_gtest, CTest label x430); the x430 library never contains them.
- Opaque, caller-owned instances; every function returns X430_OK or an X430_ERR_* code (PARAM, STATE, CONFIG, HAL, UNSUPPORTED) and refuses an instance without a successful initialization. Protocol errors go to the host in the status packet.
- Documentation website and PDF user manual: overview, architecture, usage, the emulated device in detail, integration (UART, RTOS, flash, ROS 2, DYNAMIXEL Wizard), API reference, MISRA deviation register.

Quality

- Cortex-M4 (arm-none-eabi-gcc 13.2.1, -Os, XM430-W350, 768-byte packets): 384 B RAM per device, 2,368 B per bus (832 B with 256-byte packets), 32 B per simulator; 9,921 B flash with the model table (+850 B DXL2-C); .data and .bss empty; worst-case stack 424 B (x430_bus_receive(), with any packet size).
- 131 GoogleTest cases with 100 % line and branch coverage of src/, and 100 cases with 100 % without the optional features; unit tests for all 17 models; model data against the e-Manual; 52 checks end to end with the official ROBOTIS DynamixelSDK; the simulator's trapezoidal move within 0.7 % of the closed form.
- MISRA C:2012: 0 violations with Cppcheck 2.13, deviations DEV-01 (Rule 11.3, opaque instances) and DEV-02 (Rule 11.5, callback contexts). No warnings with -Wall -Wextra -Wpedantic -Wconversion; AddressSanitizer and UndefinedBehaviorSanitizer clean.
- Verified on hardware: a stepper joint firmware (STM32G431, Zephyr) emulating an XM430-W350 with DYNAMIXEL Wizard 2.0.

APPENDIX B

License

DYNAMIXEL-X430-Control-Table-Emulator is free and open-source software, licensed under the **Apache License, Version 2.0** (apache.org/licenses/LICENSE-2.0).

Item	Value
Copyright	© 2026 Anton Vynohradov
License	Apache License 2.0
SPDX identifier	`Apache-2.0`
Applies to	The library (`x430.h`, `inc/`, `src/`), tests, examples, tools and this documentation

B.1 What the license allows

In short — the full text below is authoritative:

- **Use** the library in any project, open source or proprietary, commercial or not.
- **Modify** it and **distribute** it, in source or binary form, alone or as part of a product.
- **Patent grant:** each contributor grants you a license to any of their patents that the contribution uses.

B.2 What it requires

When you distribute DYNAMIXEL-X430-Control-Table-Emulator or a modified version of it:

- Include a copy of the license.
- Keep the copyright, patent, trademark and attribution notices in the source files.
- Mark the files you changed.
- If you redistribute a NOTICE file, keep its attribution notices (DYNAMIXEL-X430-Control-Table-Emulator has none at the moment).

Firmware that links the library in binary form only needs to make the license and the copyright notice available, for example in its documentation or "about" screen.

B.3 What it does not give

- No warranty: the software is provided "as is".
- No trademark rights: the license does not let you use the names of the author or of System-FromScratch® to endorse your product.

B.4 Trademarks

SystemFromScratch® is a registered trademark; the license grants no rights to use the name or logo. DYNAMIXEL and ROBOTIS are trademarks of ROBOTIS Co., Ltd.; STM32 is a registered trademark of STMicroelectronics International N.V. The library is not affiliated with or endorsed by either company.

B.5 Full license text


```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

    Copyright 2026 Zephyr SmartFusion2 contributors

    TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

    1. Definitions.

       "License" shall mean the terms and conditions for use, reproduction,
       and distribution as defined by Sections 1 through 9 of this document.

       "Licensor" shall mean the copyright owner or entity authorized by
       the copyright owner that is granting the License.

       "Legal Entity" shall mean the union of the acting entity and all
       other entities that control, are controlled by, or are under common
       control with that entity. For the purposes of this definition,
       "control" means (i) the power, direct or indirect, to cause the
       direction or management of such entity, whether by contract or
       otherwise, or (ii) ownership of fifty percent (50%) or more of the
       outstanding shares, or (iii) beneficial ownership of such entity.

       "You" (or "Your") shall mean an individual or Legal Entity
       exercising permissions granted by this License.

       "Source" form shall mean the preferred form for making modifications,
       including but not limited to software source code, documentation
       source, and configuration files.

       "Object" form shall mean any form resulting from mechanical
       transformation or translation of a Source form, including but
       not limited to compiled object code, generated documentation,
       and conversions to other media types.

       "Work" shall mean the work of authorship, whether in Source or
       Object form, made available under the License, as indicated by a
       copyright notice that is included in or attached to the work
       (an example is provided in the Appendix below).

       "Derivative Works" shall mean any work, whether in Source or Object
       form, that is based on (or derived from) the Work and for which the
       editorial revisions, annotations, elaborations, or other modifications
       represent, as a whole, an original work of authorship. For the purposes
       of this License, Derivative Works shall not include works that remain
       separable from, or merely link (or bind by name) to the interfaces of,
       the Work and Derivative Works thereof.

       "Contribution" shall mean any work of authorship, including
       the original version of the Work and any modifications or additions
       to that Work or Derivative Works thereof, that is intentionally
       submitted to Licensor for inclusion in the Work by the copyright owner
       or by an individual or Legal Entity authorized to submit on behalf of
       the copyright owner. For the purposes of this definition, "submitted"
       means any form of electronic, verbal, or written communication sent
       to the Licensor or its representatives, including but not limited to
       communication on electronic mailing lists, source code control systems,
       and issue tracking systems that are managed by, or on behalf of, the
       Licensor for the purpose of discussing and improving the Work, but
       excluding communication that is conspicuously marked or otherwise
```

designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

```
   6. Trademarks. This License does not grant permission to use the trade
      names, trademarks, service marks, or product names of the Licensor,
      except as required for reasonable and customary use in describing the
      origin of the Work and reproducing the content of the NOTICE file.

   7. Disclaimer of Warranty. Unless required by applicable law or
      agreed to in writing, Licensor provides the Work (and each
      Contributor provides its Contributions) on an "AS IS" BASIS,
      WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
      implied, including, without limitation, any warranties or conditions
      of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
      PARTICULAR PURPOSE. You are solely responsible for determining the
      appropriateness of using or redistributing the Work and assume any
      risks associated with Your exercise of permissions under this License.

   8. Limitation of Liability. In no event and under no legal theory,
      whether in tort (including negligence), contract, or otherwise,
      unless required by applicable law (such as deliberate and grossly
      negligent acts) or agreed to in writing, shall any Contributor be
      liable to You for damages, including any direct, indirect, special,
      incidental, or consequential damages of any character arising as a
      result of this License or out of the use or inability to use the
      Work (including but not limited to damages for loss of goodwill,
      work stoppage, computer failure or malfunction, or any and all
      other commercial damages or losses), even if such Contributor
      has been advised of the possibility of such damages.

   9. Accepting Warranty or Additional Liability. While redistributing
      the Work or Derivative Works thereof, You may choose to offer,
      and charge a fee for, acceptance of support, warranty, indemnity,
      or other liability obligations and/or rights consistent with this
      License. However, in accepting such obligations, You may act only
      on Your own behalf and on Your sole responsibility, not on behalf
      of any other Contributor, and only if You agree to indemnify,
      defend, and hold each Contributor harmless for any liability
      incurred by, or claims asserted against, such Contributor by reason
      of your accepting any such warranty or additional liability.

   END OF TERMS AND CONDITIONS

   APPENDIX: How to apply the Apache License to your work.

      To apply the Apache License to your work, attach the following
      boilerplate notice, with the fields enclosed by brackets "[]"
      replaced with your own identifying information. (Don't include
      the brackets!) The text should be enclosed in the appropriate
      comment syntax for the file format. We also recommend that a
      file or class name and description of purpose be included on the
      same "printed page" as the copyright notice for easier
      identification within third-party archives.

   Copyright [yyyy] [name of copyright owner]

   Licensed under the Apache License, Version 2.0 (the "License");
   you may not use this file except in compliance with the License.
   You may obtain a copy of the License at

       http://www.apache.org/licenses/LICENSE-2.0

   Unless required by applicable law or agreed to in writing, software
   distributed under the License is distributed on an "AS IS" BASIS,
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   See the License for the specific language governing permissions and
   limitations under the License.
```