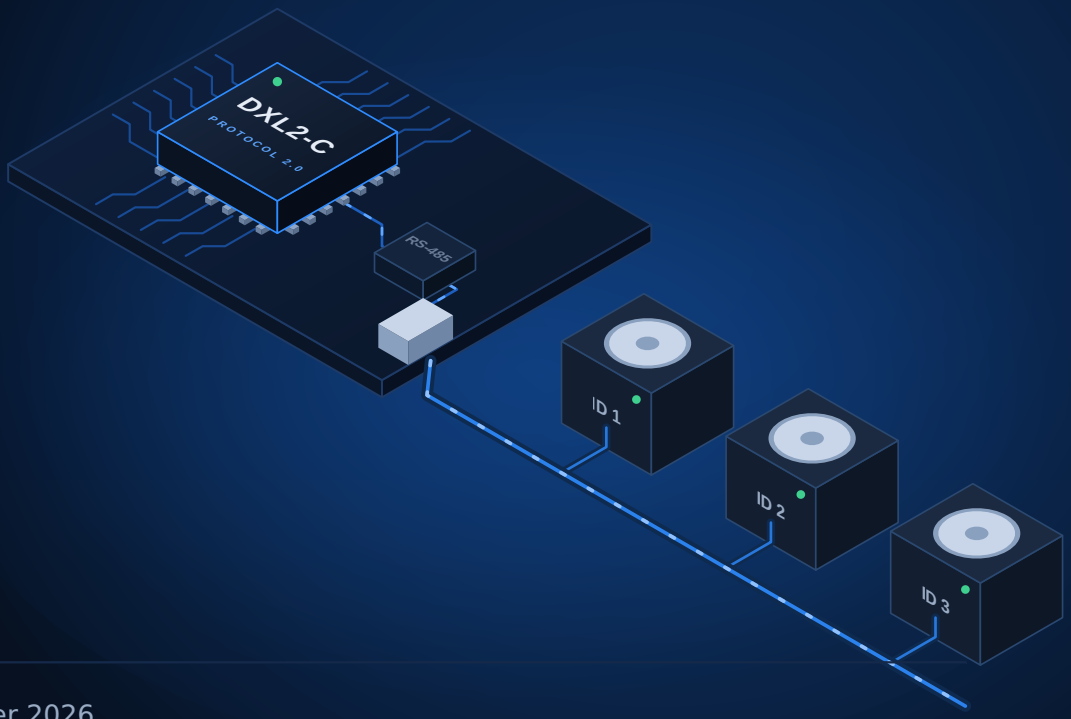
SystemFromScratch

OPEN-SOURCE EMBEDDED LIBRARY

DXL2-C

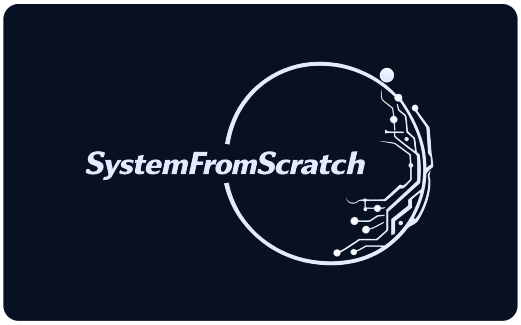
DYNAMIXEL Protocol 2.0 Library for Embedded Systems

USER MANUAL



Version 1.0.0 · October 2026
Anton Vynohradov
systemfromscratch.com

Imprint



DXL2-C User Manual
Library version 1.0.0
Edition: October 2026

Author
Anton Vynohradov
SystemFromScratch® — systemfromscratch.com
GitHub: github.com/AntonVynohradov
LinkedIn: linkedin.com/in/vynohradov
E-mail: avynohradov@systemfromscratch.com

Online
Documentation: https://antonvynohradov.github.io/DXL2-C/
Source code: https://github.com/AntonVynohradov/DXL2-C

Copyright © 2026 Anton Vynohradov.

The DXL2-C 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 DXL2-C without prior written permission.

DYNAMIXEL and ROBOTIS are trademarks of ROBOTIS Co., Ltd. They are used only to identify the protocol that DXL2-C implements. DXL2-C is an independent implementation and is not affiliated with or endorsed by ROBOTIS. 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 6
- 1.4 Key properties 7
- 1.5 Terminology 7
- 1.6 Conventions in this documentation 7

2 Getting Started 8
- 2.1 Get the library 8
- 2.2 Add it to your project 8
- 2.3 Prerequisites for development 8
- 2.4 Build with presets 9
- 2.5 Unit tests and coverage 9
- 2.6 Integration tests 9
- 2.7 Static analysis 9
- 2.8 Formatting 10
- 2.9 Versioning and releases 10
- 2.10 Documentation 10

3 How It Works 12
- 3.1 Layers 12
- 3.2 Data flow 13
- 3.3 The parser 14
- 3.4 The encoder 15
- 3.5 Memory model 15
- 3.6 Error handling 16
- 3.7 Threading and reentrancy 16
- 3.8 Design decisions 16

4 Using the Library 18
- 4.1 Setting up a link 18
- 4.2 Writing a device 19
- 4.3 Writing a controller 21
- 4.4 Using the codec without a link 22
- 4.5 Return values 23
- 4.6 Diagnostics 23
- 4.7 Common mistakes 23

5 Extending the Library 25
- 5.1 Where to extend 25
- 5.2 Adding a transport 25
- 5.3 Building a Control Table 26
- 5.4 Building instruction parameters 27
- 5.5 Vendor-specific instructions 28
- 5.6 Bridging two buses 28
- 5.7 Changing the configuration 29
- 5.8 Contributing to the library 29

6 Configuration 30
6.1 Memory footprint 30
6.2 Stack usage 30
6.3 CMake options 31

7 DYNAMIXEL Protocol 2.0 32
7.1 The bus 32
7.2 Packet 33
7.3 CRC-16 33
7.4 Byte stuffing 33
7.5 Instructions 34
7.6 Status Packet error byte 35
7.7 Parser behavior 35
7.8 Reference packets (from the specification, used in the tests) 35

8 Integration Guide 37
8.1 Zephyr: UART with an RX interrupt, byte queue and RX thread 37
8.2 Adding the library to a CMake build (Zephyr or any other) 37
8.3 Bare metal / other RTOS 38
8.4 Answering a device request 38

9 Examples 39
9.1 Bare metal 39
9.2 Host serial port 40
9.3 Zephyr 41

10 DXL2 Studio 42
10.1 Modes 42
10.2 Installation and start 43
10.3 Window layout 43
10.4 Communication debugger 47
10.5 How the tool uses the library 48

11 API Reference 49
11.1 Version — dxl2.h 49
11.2 Configuration — dxl2_config.h 49
11.3 Definitions — dxl2_defs.h 50
11.4 Codec — dxl2_protocol.h 51
11.5 Link — dxl2_link.h 53
11.6 Return codes summary 55

12 MISRA C:2012 compliance 56
12.1 Checking 56
12.2 Current status 56
12.3 Manual review notes 56
12.4 Deviation register 57

Author & Company 58
Author 58
SystemFromScratch® 59
Contact 59
Support and feedback 60

Trademarks and acknowledgements 60

Appendix A Changelog 61

A.1 Unreleased 61

A.2 1.0.0 - 2026-10-25 61

Appendix B License 62

B.1 What the license allows 62

B.2 What it requires 62

B.3 What it does not give 62

B.4 Trademarks 63

B.5 Full license text 63

CHAPTER 1

Introduction

DXL2-C is a C11 library that implements the wire format of the ROBOTIS DYNAMIXEL Protocol 2.0 (docs.robotis.com/docs/dxl/protocol/protocol2/): how packets are framed, protected with a CRC, byte-stuffed and parsed from a stream of bytes. It is written for embedded systems that must be predictable and reviewable, and it runs unchanged on microcontrollers, under an RTOS and on a PC.

1.1 What the library does

- **Encodes packets.** Given an ID, an instruction and parameters, it builds the complete wire packet: header, Length field, byte stuffing and CRC-16.
- **Parses a byte stream.** It accepts bytes in any chunking, finds packet headers, validates Length and CRC, removes byte stuffing and hands you complete packets. It resynchronizes by itself after noise or lost bytes.
- **Connects the codec to a transport.** The *link* layer owns a parser, calls your handler for every valid packet and sends replies through a write function you supply.
- **Defines the protocol constants.** Every instruction and status error value of the specification, plus helpers for little-endian fields.

1.2 What the library does not do

DXL2-C is deliberately limited to the wire format. These belong to your application:

Concern	Why it is not in the library
UART, RS-485, USB or socket drivers	They are hardware- and OS-specific. You pass bytes in and provide a write function.
Threads, queues, interrupts, locking	Every system has its own. The integration guide shows the usual patterns.
Control Tables and device models	Addresses and registers differ between models. Your handler interprets READ, WRITE and the other instructions.
Timeouts and retries	A controller decides how long to wait for a reply; the library has no clock.
Higher-level helpers (SYNC_WRITE builders, servo APIs)	They are easy to build on top of the codec; see Extending the Library.

1.3 Who it is for

- **Device firmware.** A board that should behave like a DYNAMIXEL device — a custom actuator, a sensor hub or a gripper — on an existing DYNAMIXEL bus.
- **Controller firmware.** A microcontroller that drives DYNAMIXEL servos directly.
- **Host tools.** Test benches, production tools and debuggers that need a codec which behaves exactly like the firmware. DXL2 Studio (p. 42) is such a tool.

1.4 Key properties

Property	Value
Language	C11 (`-std=c11`), compiles without warnings with `-Wall -Wextra -Wpedantic -Wconversion`
Dependencies	`<stddef.h>`, `<stdint.h>`, `<stdbool.h>`, `<string.h>` only
Memory	No heap; all state in caller-owned structures; sizes fixed by `DXL2_MAX_PACKET_SIZE`
RAM per link	about `2 × DXL2_MAX_PACKET_SIZE` + 40 bytes (≈ 550 bytes with the default 256)
Stack	at most about 600 bytes on Cortex-M4 for the transmit path (details)
Coding standard	MISRA C:2012, 0 Cppcheck addon violations, 3 advisory deviations
Tests	100 % line and branch coverage; C/Python cross-check; ROBOTIS reference packets
Thread safety	None built in, by design; see Threading and reentrancy
License	Apache License 2.0

1.5 Terminology

Controller
: The node that sends Instruction Packets and waits for replies. ROBOTIS documentation also calls it the *main controller* or *master*.

Device
: A node with an ID (0–252) that executes instructions and answers with Status Packets: a servo, a sensor, or your firmware built with DXL2-C.

Instruction Packet
: A packet sent by the controller. Its instruction byte says what to do (PING, READ, WRITE, ...).

Status Packet
: The reply of a device. Its instruction byte is `0x55` and its first parameter is the error byte.

Broadcast ID
: ID `254` (`0xFE`). Every device executes the instruction; only some instructions are answered.

Control Table
: The register map of a device. Not part of the protocol itself and not part of this library.

Byte stuffing
: The rule that keeps the header sequence out of the packet body. See Byte stuffing (p. 33).

1.6 Conventions in this documentation

- Hexadecimal bytes are written without a prefix inside packets (`FF FF FD 00`) and with `0x` in text and code (`0x55`).
- Multi-byte fields are little endian, as on the wire.
- `DXL2_MAX_PACKET_SIZE` is shown with its default value, 256 bytes.
- Code examples follow the library's own style: C11, MISRA-friendly, one `return` per function where it matters.

CHAPTER 2

Getting Started

This chapter shows how to get DXL2-C, add it to your project, and build and test it.

2.1 Get the library

Clone the repository, or download a release package from the GitHub releases (github.com/AntonVynohradov/DXL2-C/releases) page:

```
git clone https://github.com/AntonVynohradov/DXL2-C.git
```

Each release provides the source code, a prebuilt static library for Linux x86-64 with its headers, and the MISRA report.

2.2 Add it to your project

The library is two C files and five headers. Choose whichever fits your build:

CMake. Include `dxl2.cmake`; it defines the static library target `dxl2` with its include directories and version definitions:

```
include(path/to/DXL2-C/dxl2.cmake)
target_link_libraries(app PRIVATE dxl2)
```

Any other build system (Make, IDE projects, vendor tools). Compile `src/dxl2_protocol.c` and `src/dxl2_link.c` with your firmware and add the repository root and `inc/` to the include path. No other files and no special flags are needed.

Prebuilt package. Link `lib/libdxl2.a` and add `include/` to the include path. Use it only for the platform it was built for.

Then include the umbrella header and continue with Using the Library (p. 18):

```
#include "dxl2.h"
```

2.3 Prerequisites for development

- C11 compiler (GCC, Clang, or an embedded toolchain)
- CMake 3.20+
- Optional: GoogleTest and gcovr (unit tests, coverage), Python 3 and pytest (integration tests), Cppcheck (static analysis, MISRA C:2012 addon), clang-format, MkDocs with mkdoxy (documentation)

2.4 Build with presets

Preset	Purpose
`DXL2[Debug]`	Static library, `-g`, `DXL2_DEBUG` defined
`DXL2[Release]`	Static library, `-O2 -DNDEBUG`
`DXL2[UnitTests]`	GoogleTest executable with coverage instrumentation
`DXL2[IntegrationTests]`	pytest suite run through CTest
`DXL2[Examples]`	Host examples and the bare-metal simulation test

```
cmake --preset "DXL2[Debug]"
cmake --build "build/DXL2[Debug]"
```

All builds use `-std=c11 -Wall -Wextra -Wpedantic -Wconversion`.

2.5 Unit tests and coverage

```
cmake --preset "DXL2[UnitTests]"
cmake --build "build/DXL2[UnitTests]" --target coverage
```

The `coverage` target builds the tests, runs them through CTest and writes an HTML report to `build/DXL2[UnitTests]/Coverage/coverage.html`. It fails when line or branch coverage of `src/` is below `DXL2_COVERAGE_MIN` (100 %).

2.6 Integration tests

```
python -m pip install pytest
cmake --preset "DXL2[IntegrationTests]"
cmake --build "build/DXL2[IntegrationTests]" --target integration_tests
```

The suite in `tests/integration/` tests the Python codec and compiles `src/dxl2_protocol.c` with `gcc` to compare 300 random packets between the C and Python implementations. The cross-check is skipped when `gcc` is not on the path.

2.7 Static analysis

From any configured build directory:

Target	Tool
`cppcheck-analyze`	Cppcheck, all checks, report in `cppcheck.txt`
`misra-analyze`	Cppcheck MISRA C:2012 addon on `src/`, report in `misra.txt`
`codeql-analyze`	CodeQL `cpp-queries`, report in `codeql-report.sarif`

`misra-analyze` fails when the report contains a violation; see MISRA C:2012 (p. 56) for the deviation register.

MISRA rule texts are not distributed with Cppcheck. If you have a licensed copy, pass it with `-DDXL2_MISRA_RULE_TEXTS=<file>` to get rule headlines in the report.

2.8 Formatting

```
clang-format -i src/*.c inc/*.h dxl2.h tests/unit/*.cpp tests/unit/*.h examples/*/*.[ch]
examples/zephyr/src/*.c
```

CI rejects pull requests with unformatted C/C++ files.

2.9 Versioning and releases

`cmake/version.cmake` takes the version from the latest `vX.Y.Z` Git tag and passes it as `DXL2_VERSION_MAJOR/MINOR/PATCH` and the string `DXL2_VERSION`. Without a tag it uses `DXL2_DEFAULT_VERSION` and prints a warning; keep the fallback values in `dxl2.h` equal to that default (a unit test checks that the string matches the numbers).

To release version X.Y.Z:

1. Set `DXL2_DEFAULT_VERSION` in `cmake/version.cmake` and the fallbacks in `dxl2.h` to X.Y.Z.
2. Move the `[Unreleased]` entries in `CHANGELOG.md` to a `[X.Y.Z] - date` section.
3. Merge to `main`, then tag and push: `git tag vX.Y.Z && git push origin vX.Y.Z`.

The Release workflow refuses a tag that does not match these files, runs all quality gates and publishes the GitHub Release.

2.10 Documentation

The documentation is written in Markdown in `docs/` and published with MkDocs (Material theme); the API pages are generated from the source comments by Doxygen through mkdoxy. Install the tools into the project virtual environment (VS Code task *Python: Setup venv*, or `python -m pip install -r requirements-dev.txt`) and Doxygen.

Task	Command	VS Code task
Live preview	`mkdocs serve`	*Docs: Serve on localhost*
Build the website into `site/`	`mkdocs build`	*Docs: Build site*
Build the printable PDF manual	`python tools/docs/build_manual.py`	*Docs: Build PDF manual*

The PDF manual is written to `build/manual/DXL2-C_User_Manual_v<version>.pdf`. It is an A4 document with a cover page, an imprint, a table of contents with page numbers, numbered chapters, running headers, page numbers and PDF bookmarks. It needs WeasyPrint, which uses the Pango library: on Linux install `libpango-1.0-0` and `libpangoft2-1.0-0` (Debian, Ubuntu); on Windows install the MSYS2 package `mingw-w64-x86_64-pango` — the script finds `C:\msys64\mingw64\bin` by itself, or set `WEASYPRINT_DLL_DIRECTORIES`.

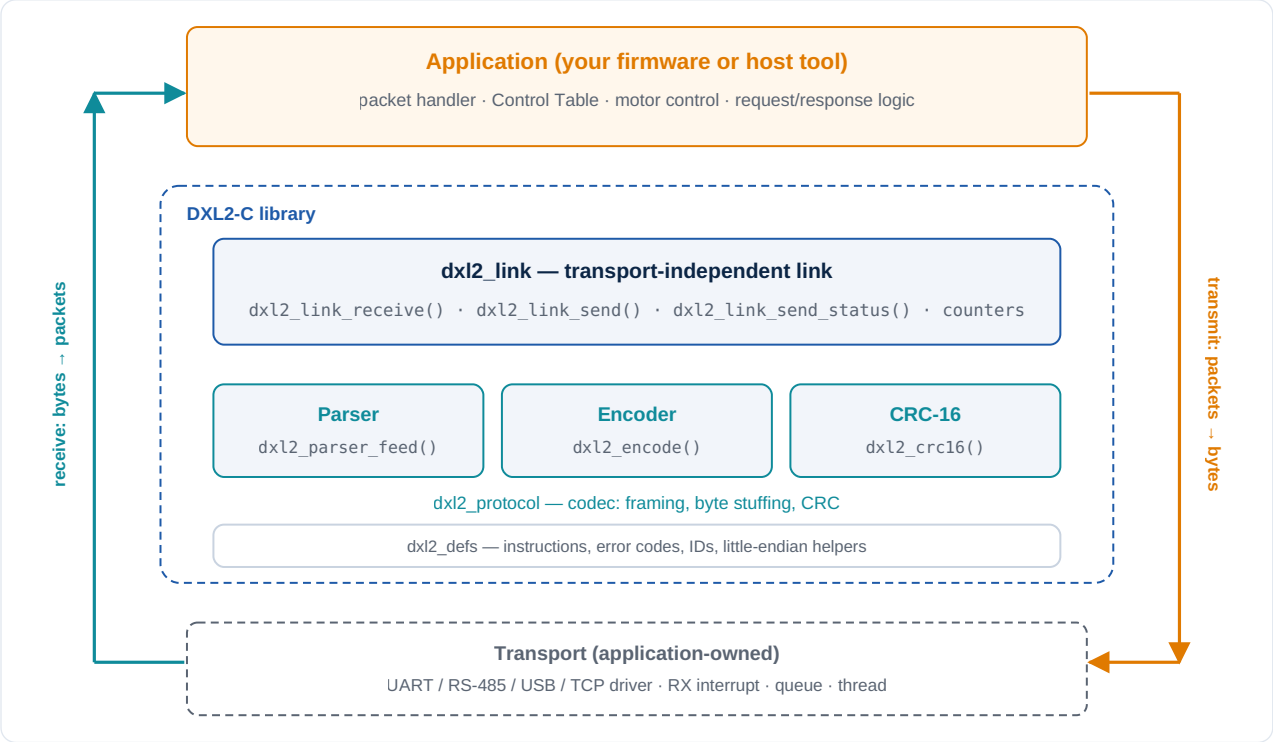
Diagrams are SVG files generated by `python tools/docs/diagrams.py`; edit the script, not the SVG files.

CHAPTER 3

How It Works

This chapter explains the internal structure of DXL2-C: its layers, how a byte becomes a packet and a packet becomes bytes, how the parser recovers from errors, and the rules for memory, errors and concurrency that follow from the design.

3.1 Layers



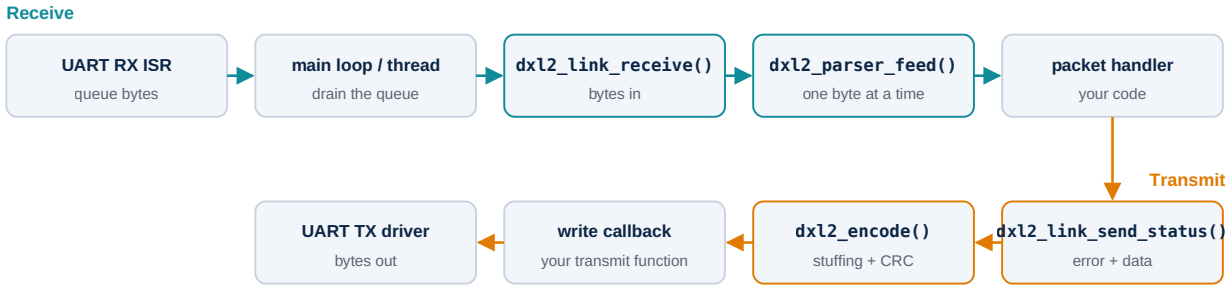
The library has three layers. Each one depends only on the layers below it.

Layer	Header	Responsibility
Definitions	`dxl2_defs.h`	Instruction and error values from the specification, broadcast ID, little-endian helpers. No state, no functions with side effects.
Codec	`dxl2_protocol.h`	CRC-16, the incremental stream parser and the encoder. Pure functions over caller-owned buffers.
Link	`dxl2_link.h`	Glue between the codec and a transport: owns a parser, dispatches packets to a handler, sends packets through a write function, counts good and bad packets.

`dxl2.h` includes all of them, together with the build-time configuration in `dxl2_config.h`. Applications normally include only `dxl2.h`.

You can use the codec without the link — for example in a tool that parses a log file, or in firmware that already has its own message dispatcher. The link is a thin convenience layer (about 100 lines) on top of the codec.

3.2 Data flow



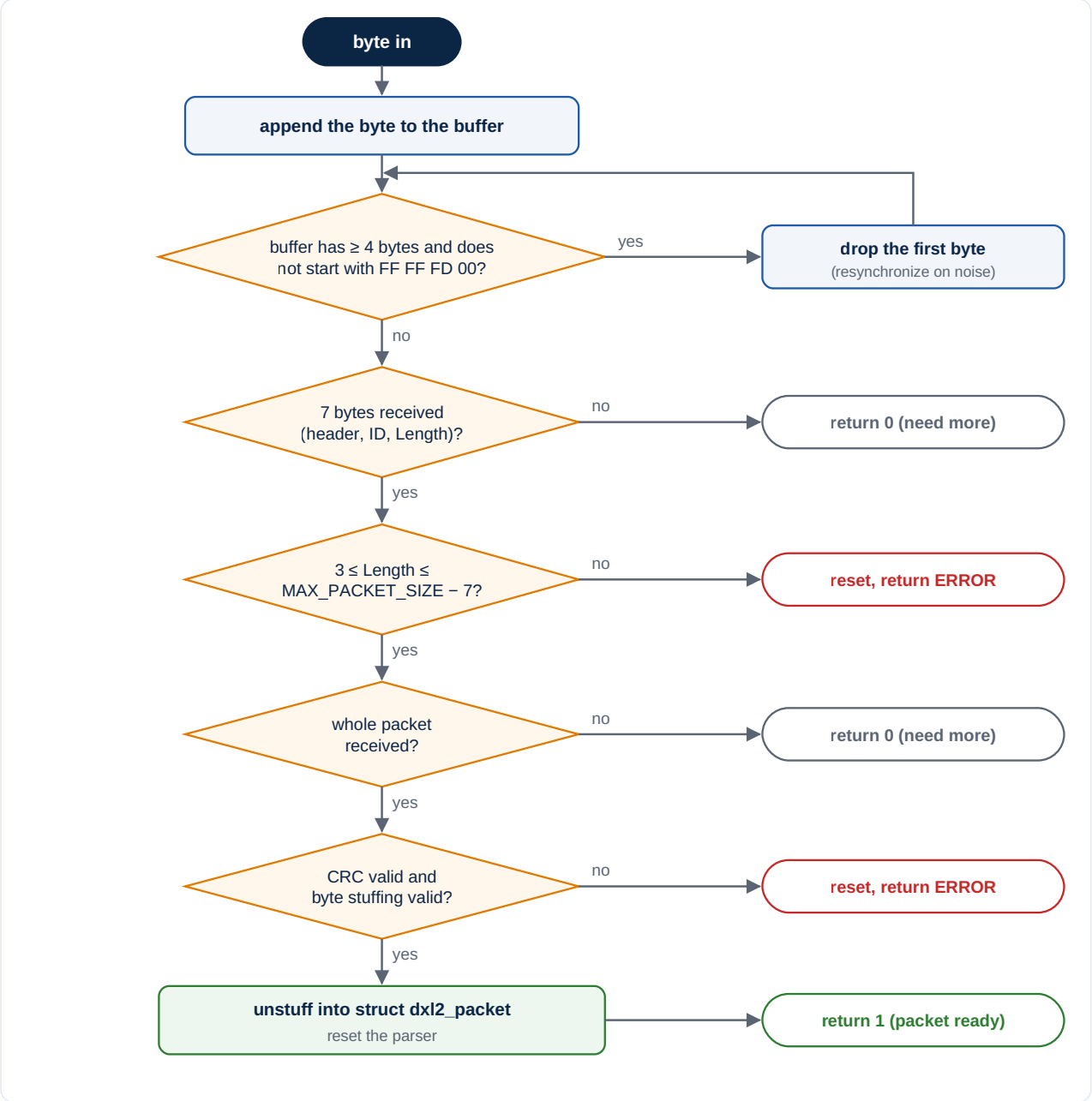
The handler runs inside dxl2_link_receive(), in the caller's context: reply from it, or queue the work and reply later.

Receive path. Your transport delivers bytes — typically a UART interrupt puts them in a queue, and a main loop or thread takes them out. You pass them to `dxl2_link_receive()` in chunks of any size. The link feeds them one at a time to `dxl2_parser_feed()`. Whenever the parser completes a valid packet, the link increments its `rx_packets` counter and calls your packet handler with a pointer to the decoded packet. Rejected packets increment `invalid_packets` and are otherwise ignored.

Transmit path. To send, you call `dxl2_link_send()` (any instruction) or `dxl2_link_send_status()` (a Status Packet). The link encodes the packet into a buffer on its own stack with `dxl2_encode()`, which adds the header, Length, byte stuffing and CRC, and then passes the complete packet to your write function in one call.

The handler runs inside `dxl2_link_receive()`, in the context that called it. It may send replies on the same link directly: transmitting does not touch the parser.

3.3 The parser



`dxl2_parser_feed()` consumes exactly one byte per call and returns `1` (packet ready), `0` (need more bytes) or `DXL2_PARSE_ERROR`. Internally it keeps the bytes of the packet in progress in `struct dxl2_parser`:

1. **Append.** The new byte is stored at the end of the buffer.
2. **Synchronize.** While the buffer holds at least four bytes that are not the header `FF FF FD 00`, the first byte is dropped. This sliding window discards noise and partial packets and lines the buffer up with the next header.
3. **Check the Length.** Once seven bytes are present (header, ID, Length), the Length field is checked: it must be at least 3 (instruction and CRC) and the whole packet must fit in `DXL2_MAX_PACKET_SIZE`. A bad Length is rejected at once, without waiting for bytes that may never come.
4. **Wait.** Until `7 + Length` bytes are present, the parser returns `0`.

5. **Validate and decode.** The CRC is checked over the stuffed bytes, exactly as they were sent. Then byte stuffing is removed while the instruction and parameters are copied into `struct dxl2_packet`. A header sequence without its stuffing byte is a malformed packet.
6. **Reset.** After a packet — valid or not — the parser buffer is emptied and the next byte starts a new search.

The cost per byte is constant except for the synchronization step, which moves at most `DXL2_MAX_PACKET_SIZE` bytes. There is no recursion and no data-dependent loop without a fixed bound, so the worst-case execution time is easy to bound.

> **Recovery after errors**
>
> The bytes of a rejected packet are discarded. If a lost byte makes the Length field reach into the next packet, that packet is rejected too, and the parser locks onto the one after it. Controllers should treat a missing reply as a timeout and retry.

3.4 The encoder

`dxl2_encode()` writes the header and ID, then copies the instruction and parameters into the output buffer. After every byte it checks whether the last three bytes written are `FF FF FD`; if so, it inserts an extra `FD`. It then fills in the Length field — which counts the stuffed bytes — and appends the CRC computed over everything before it.

The output capacity is checked before every byte is written, so an undersized buffer returns `DXL2_PARSE_ERROR` instead of overflowing. The worst case for stuffing is one extra byte per three parameter bytes; with `DXL2_MAX_PACKET_SIZE` as the capacity, every packet that the parser accepts can also be encoded.

3.5 Memory model

All state lives in two structures that you allocate — statically, on a stack or inside your own objects:

Structure	Contents	Size with the default configuration
`struct dxl2_parser`	bytes of the packet in progress, their count	256 bytes + `size_t`
`struct dxl2_packet`	ID, instruction, parameter count, unstuffed parameters	250 bytes
`struct dxl2_link`	configuration, one parser, the last packet, two counters	≈ 550 bytes on a 32-bit MCU

The library has no global variables except one read-only table (the header bytes), never calls `malloc()`, and allocates only fixed-size buffers on the stack: one packet buffer in `dxl2_link_send()` and one parameter buffer in `dxl2_link_send_status()`. The measured stack usage per function is listed in Configuration (p. 30).

Because nothing is shared, you can have as many links as you have buses, each with its own configuration.

3.6 Error handling

Every function reports errors through its return value; nothing is printed, logged or asserted.

Situation	Result
A `NULL` pointer or inconsistent length passed to a function	`DXL2_PARSE_ERROR` (or `0` for `dxl2_crc16()` and the counters); nothing is written
A packet with a bad Length, CRC or byte stuffing	`dxl2_parser_feed()` returns `DXL2_PARSE_ERROR`; the link counts it in `invalid_packets`
A packet that does not fit in the output buffer	`dxl2_encode()` and `dxl2_link_send*()` return `DXL2_PARSE_ERROR`
The write function fails	`dxl2_link_send*()` return the write function's own (negative) value
A corrupted parser structure (for example, never initialized)	`dxl2_parser_feed()` resets it and returns `DXL2_PARSE_ERROR` instead of writing out of bounds

Protocol-level errors that a *device* reports — an unknown instruction, an address out of range — are not library errors. Your handler sends them back as the error byte of a Status Packet (`DXL2_ERR_*`).

3.7 Threading and reentrancy

The library contains no locks, by design: the right primitive depends on your system, and a hidden mutex would make the library unusable in interrupt-only firmware.

- **Codec functions** are reentrant. They only touch the buffers you pass in, so they can be called from any context, concurrently, as long as each call has its own buffers.
- **A link** is not thread-safe. Call `dxl2_link_receive()` for a given link from one context only. Do not call it from an interrupt handler: the packet handler runs inside it and usually transmits.
- **Sending** from several contexts at once (for example an RX thread replying and an application thread sending commands) needs a mutex around your write function, or around the `dxl2_link_send*()` calls.
- **The packet handler** receives a pointer into the link. The data is valid only until the handler returns; copy what you need to keep.

3.8 Design decisions

Decision	Reason
One byte per parser call	Works with any transport chunking, including single-byte interrupts, with no internal buffering policy to configure.
CRC checked before unstuffing	The CRC is defined over the bytes on the wire; checking first rejects corrupted packets before any data is copied.
Fixed-size buffers from one macro	Memory use is known at build time and can be reviewed; one macro scales everything consistently.
Callbacks for I/O	The library never needs to know about drivers or operating systems, and it is trivial to test with mocks.

Decision	Reason
`int` status codes	Lets an existing driver `write()` function be used as the write callback without a wrapper; recorded as MISRA deviation DEV-01.

CHAPTER 4

Using the Library

This chapter shows how to use DXL2-C in the three typical roles: as a **device** that answers instructions, as a **controller** that drives devices, and as a plain **codec** without the link layer. All examples use only the public API from `dxl2.h`.

4.1 Setting up a link

Every link needs two things from you: a function that writes bytes to the transport and a function that handles received packets.

```c
#include "dxl2.h"

static struct dxl2_link bus;

/* Write one complete packet. Return 0 on success, a negative value on failure. */
static int bus_write(void* context, const uint8_t* data, size_t length)
{
    (void) context;
    return uart_write_blocking(data, length);
}

/* Called for every valid packet received on the bus. */
static void bus_handler(const struct dxl2_packet* packet, void* context)
{
    (void) context;
    /* see below */
}

void bus_init(void)
{
    const struct dxl2_link_config config = {
        .write = bus_write,
        .write_context = NULL,
        .handler = bus_handler,
        .handler_context = NULL,
    };

    (void) dxl2_link_init(&bus, &config);
}
```

- `dxl2_link_init()` copies the configuration, so `config` can be a local variable.
- The two context pointers are passed back unchanged to your callbacks. Use them to reach your driver or application object without global variables.
- A link without a write function is *receive-only*: sending returns `DXL2_PARSE_ERROR`. A link without a handler still parses and counts packets.

Then move received bytes into the link wherever your transport delivers them:

```c
void bus_poll(void)
{
    uint8_t bytes[64];
    const size_t count = uart_read_nonblocking(bytes, sizeof(bytes));
```

```
    dxl2_link_receive(&bus, bytes, count);
}
```

Chunk boundaries do not matter: one byte at a time, whole packets or several packets at once all give the same result.

4.2 Writing a device

A device answers Instruction Packets addressed to its ID. The handler receives each decoded packet; `packet->parameters` holds the unstuffed parameters and `packet->parameter_length` their number.

```
#define MY_ID 1U

static uint8_t control_table[128];

static void bus_handler(const struct dxl2_packet* packet, void* context)
{
    const bool broadcast = (packet->id == DXL2_BROADCAST_ID);

    (void) context;
    if ((packet->id != MY_ID) && !broadcast)
    {
        return; /* addressed to another device */
    }

    switch (packet->instruction)
    {
        case DXL2_INST_PING:
            handle_ping();            /* every device answers a broadcast PING */
            break;
        case DXL2_INST_READ:
            if (!broadcast)
            {
                handle_read(packet);
            }
            break;
        case DXL2_INST_WRITE:
            handle_write(packet, broadcast);
            break;
        case DXL2_INST_STATUS:
            break;                    /* a reply from another device on the bus */
        default:
            if (!broadcast)
            {
                (void) dxl2_link_send_status(&bus, MY_ID, DXL2_ERR_INSTRUCTION, NULL, 0U);
            }
            break;
    }
}
```

Reading parameters

Instruction parameters are little endian. Use the helpers from `dxl2_defs.h` instead of shifting bytes by hand:

```
/* READ: address (2 bytes), length (2 bytes) */
static void handle_read(const struct dxl2_packet* packet)
{
    if (packet->parameter_length != 4U)
    {
        (void) dxl2_link_send_status(&bus, MY_ID, DXL2_ERR_DATA_LENGTH, NULL, 0U);
```

```
    }
    else
    {
        const uint16_t address = dxl2_get_u16(&packet->parameters[0]);
        const uint16_t length = dxl2_get_u16(&packet->parameters[2]);

        if (((uint32_t) address + length) > sizeof(control_table))
        {
            (void) dxl2_link_send_status(&bus, MY_ID, DXL2_ERR_DATA_RANGE, NULL, 0U);
        }
        else
        {
            (void) dxl2_link_send_status(&bus, MY_ID, DXL2_ERR_NONE,
                                         &control_table[address], length);
        }
    }
}
```

Always check `parameter_length` before reading parameters: the library guarantees that the bytes up to `parameter_length` are valid, but it does not know the layout of each instruction.

Answering

`dxl2_link_send_status()` builds the Status Packet for you: instruction `0x55`, the error byte, then your data.

Error value	Meaning
`DXL2_ERR_NONE`	Success
`DXL2_ERR_RESULT_FAIL`	The instruction could not be processed
`DXL2_ERR_INSTRUCTION`	Undefined instruction, or ACTION without REG_WRITE
`DXL2_ERR_CRC`	CRC mismatch (DXL2-C drops CRC-invalid packets before your handler sees them, so a device built on it normally never sends this)
`DXL2_ERR_DATA_RANGE`	Data outside the register range
`DXL2_ERR_DATA_LENGTH`	Data shorter than required
`DXL2_ERR_DATA_LIMIT`	Data outside the configured limit
`DXL2_ERR_ACCESS`	Writing a read-only register, reading a write-only one, or a locked EEPROM area

OR the error with `DXL2_ERR_ALERT_BIT` when your device has an active hardware error, so the controller knows to read its Hardware Error Status.

Rules a well-behaved device follows

These rules come from the protocol specification. The library leaves them to you because they depend on your Control Table:

- **Broadcast.** Execute broadcast instructions, but do not answer them — except PING, which every device answers. On a real bus, devices answer a broadcast PING in ID order, each after a delay, to avoid collisions.
- **Status Return Level.** If you emulate the Status Return Level register, answer only PING (level 0), only PING and READ (level 1), or everything (level 2).
- **Return Delay Time.** If you emulate it, wait that long before transmitting.

- **Half duplex.** Release the bus (switch the transceiver to receive) as soon as the last byte has left the UART. Your write function is the right place for that.

The bare-metal example (p. 39) implements PING, READ and WRITE with range and access checks on a small Control Table.

4.3 Writing a controller

A controller sends Instruction Packets and receives Status Packets through the same link. Because the library has no clock, your application sends the request, then keeps feeding received bytes until the reply arrives or its own timeout expires.

```
struct pending_reply
{
    uint8_t id;          /* the device we expect an answer from */
    bool received;
    uint8_t error;
    uint8_t data[8];
    uint16_t data_length;
};

static void controller_handler(const struct dxl2_packet* packet, void* context)
{
    struct pending_reply* reply = context;

    /* Ignore our own Instruction Packets echoed by a single-wire adapter. */
    if ((packet->instruction == DXL2_INST_STATUS) && (packet->id == reply->id)
        && (packet->parameter_length >= 1U)
        && ((packet->parameter_length - 1U) <= sizeof(reply->data)))
    {
        reply->error = packet->parameters[0];
        reply->data_length = (uint16_t) (packet->parameter_length - 1U);
        (void) memcpy(reply->data, &packet->parameters[1], reply->data_length);
        reply->received = true;
    }
}

/* Read the 4-byte Present Position (address 132 on X-series servos). */
static int read_present_position(uint8_t id, int32_t* position)
{
    uint8_t parameters[4];
    int result = -1;

    pending.id = id;
    pending.received = false;
    dxl2_put_u16(&parameters[0], 132U); /* address */
    dxl2_put_u16(&parameters[2], 4U);   /* length */

    if (dxl2_link_send(&bus, id, DXL2_INST_READ, parameters, sizeof(parameters)) == 0)
    {
        const uint32_t start = millis();

        while (!pending.received && ((millis() - start) < 20U))
        {
            bus_poll(); /* feeds dxl2_link_receive() */
        }
        if (pending.received && (pending.error == DXL2_ERR_NONE) && (pending.data_length =
= 4U))
        {
            *position = (int32_t) dxl2_get_u32(pending.data);
            result = 0;
        }
    }
    return result;
}
```

Here `pending` is a `struct pending_reply` passed as `handler_context` when the link is initialized, and `millis()` is your system tick.

Points to keep in mind:

- **Echo.** On a single-wire half-duplex bus without echo cancellation, your own Instruction Packet comes back to you. It is a valid packet with your instruction byte, so filter on `DXL2_INST_STATUS`.
- **Timeouts.** Choose a timeout that covers the device's Return Delay Time plus the transmission time of the reply. Retry on timeout: a corrupted reply is dropped silently by the parser.
- **Broadcast PING.** Several devices answer one after another. Keep feeding bytes for a while and collect every Status Packet; the host serial example (p. 40) does exactly that.
- **SYNC and BULK instructions.** Build their parameter blocks with `dxl2_put_u16()` and `dxl2_put_u32()` and send them with `dxl2_link_send()`; see Extending the Library (p. 27).

4.4 Using the codec without a link

The codec functions work on plain buffers. This is useful in tools, tests, gateways, or firmware that already has its own message loop.

Encoding a packet into your own buffer:

```
uint8_t packet[DXL2_MAX_PACKET_SIZE];
size_t packet_length = 0U;

if (dxl2_encode(1U, DXL2_INST_PING, NULL, 0U, packet, sizeof(packet), &packet_length) ==
0)
{
    /* packet[0 .. packet_length - 1] = FF FF FD 00 01 03 00 01 19 4E */
}
```

Parsing a stream:

```
static struct dxl2_parser parser;
static struct dxl2_packet packet;

void stream_init(void)
{
    dxl2_parser_init(&parser);
}

void stream_bytes(const uint8_t* data, size_t length)
{
    for (size_t i = 0U; i < length; ++i)
    {
        const int result = dxl2_parser_feed(&parser, data[i], &packet);

        if (result == 1)
        {
            process(&packet);      /* a complete, valid packet */
        }
        else if (result < 0)
        {
            count_bad_packet();    /* keep feeding: the parser resynchronizes */
        }
        else
        {
            /* 0: need more bytes */
        }
```

```
    }
}
```

`packet` is valid only after a call that returned `1`; after an error its contents are undefined. Use one parser per byte stream.

Computing a CRC directly, for example to build a test vector:

```
static const uint8_t ping[] = {0xFF, 0xFF, 0xFD, 0x00, 0x01, 0x03, 0x00, 0x01};
const uint16_t crc = dxl2_crc16(ping, sizeof(ping)); /* 0x4E19, sent as 19 4E */
```

4.5 Return values

Function	Success	Failure
`dxl2_link_init()`	`0`	`DXL2_PARSE_ERROR` (`NULL` argument)
`dxl2_link_send()`, `dxl2_link_send_status()`	`0`	`DXL2_PARSE_ERROR`, or the negative value of your write function
`dxl2_encode()`	`0`	`DXL2_PARSE_ERROR` (invalid arguments, packet does not fit)
`dxl2_parser_feed()`	`1` packet ready, `0` need more	`DXL2_PARSE_ERROR`
`dxl2_crc16()`	the CRC	`0` for a `NULL` buffer
`dxl2_link_rx_packets()`, `dxl2_link_invalid_packets()`	the counter	`0` for a `NULL` link

`DXL2_PARSE_ERROR` is `-1`. Make your write function return other negative values for its own errors so that you can tell them apart.

4.6 Diagnostics

Each link counts the packets it accepted and rejected since `dxl2_link_init()`:

```
const uint32_t good = dxl2_link_rx_packets(&bus);
const uint32_t bad = dxl2_link_invalid_packets(&bus);
```

A steadily rising `invalid_packets` count usually means a wrong baud rate, a baud rate error above about 2 %, a missing termination, bus contention, or text (console output, logging) on the same UART.

4.7 Common mistakes

Symptom	Likely cause
Nothing is ever received	`dxl2_link_init()` not called, or received bytes never passed to `dxl2_link_receive()`

Symptom	Likely cause
Every packet is invalid	Wrong baud rate, inverted signal, or the transceiver direction is not switched
A controller receives its own commands	Single-wire bus echo: filter on `DXL2_INST_STATUS`
Replies corrupt each other	Two devices with the same ID, or a device answering a broadcast instruction
Crash or garbage in the handler after it returns	Keeping the `packet` pointer: copy the data instead
Random corruption under load	Two threads sending on one link without a lock, or `dxl2_link_receive()` called from an interrupt

CHAPTER 5

Extending the Library

DXL2-C is intentionally small: it implements the wire format and stops there. This chapter shows where the extension points are, how to build the common higher-level pieces on top of the public API, and how to change the library itself when that is the right thing to do.

5.1 Where to extend

You want to…	Do it in	Library change needed?
Support another transport (RS-485, USB CDC, TCP, CAN gateway)	your write function and receive loop	No
Emulate a device model	your packet handler and Control Table	No
Send SYNC, BULK or other multi-device instructions	parameter builders on top of `dxl2_link_send()`	No
Use a private, vendor-specific instruction	your handler; the codec passes any instruction byte through	No
Handle bigger packets	`DXL2_MAX_PACKET_SIZE`	No, configuration only
Run several buses	one `struct dxl2_link` per bus	No
Fix a bug or change wire-format behavior	the library sources	Yes — see Contributing

Almost everything is done *on top of* the library. Keep it that way: your extensions stay portable, and you can update DXL2-C without merging changes.

5.2 Adding a transport

A transport needs two pieces of code.

Receive. Deliver received bytes to `dxl2_link_receive()` from one context. Any chunking works. Typical shapes:

Transport	Receive loop
UART with RX interrupt	ISR writes to a ring buffer; main loop or thread drains it
UART with DMA	Idle-line or half/full-transfer interrupt signals a thread; the thread passes the new region
USB CDC / TCP socket	Pass each received block as it arrives
Log file or capture	Read the file in blocks and pass them in order

Transmit. Implement `dxl2_write_fn`:

```
typedef int (*dxl2_write_fn)(void* context, const uint8_t* data, size_t length);
```

The contract:

- `data` is one complete packet, header to CRC. Send all `length` bytes in order, without interleaving other output.
- Return `0` on success or a negative value on failure; the value is returned by `dxl2_link_send*()`. Avoid `-1`, which is `DXL2_PARSE_ERROR`.
- The buffer is valid only during the call. A buffered or DMA transmitter must copy it.
- On a half-duplex bus, enable the driver before the first byte and switch back to receive when the last byte has left the shift register — not when the FIFO is empty.
- If several contexts can send, serialize the calls with a mutex.

Use `write_context` to pass your driver instance, so one write function serves several buses:

```
struct uart_port; /* your driver */
int uart_port_write(struct uart_port* port, const uint8_t* data, size_t length);

static int link_write(void* context, const uint8_t* data, size_t length)
{
    return uart_port_write((struct uart_port*) context, data, length);
}

static struct dxl2_link bus_a;
static struct dxl2_link bus_b;

void buses_init(struct uart_port* port_a, struct uart_port* port_b)
{
    const struct dxl2_link_config config_a = {.write = link_write, .write_context =
port_a};
    const struct dxl2_link_config config_b = {.write = link_write, .write_context =
port_b};

    (void) dxl2_link_init(&bus_a, &config_a);
    (void) dxl2_link_init(&bus_b, &config_b);
}
```

5.3 Building a Control Table

A device is mostly a register map. A table-driven description keeps the READ and WRITE handlers generic and makes range and access checks automatic:

```
enum access
{
    ACCESS_R,  /* read-only */
    ACCESS_RW, /* read-write, RAM */
    ACCESS_RW_EEPROM, /* read-write only while torque is off */
};

struct item
{
    uint16_t address;
    uint8_t size;
    enum access access;
};

static const struct item items[] = {
    {0U, 2U, ACCESS_R},           /* Model Number */
```

```
    {6U, 1U, ACCESS_R},           /* Firmware Version */
    {7U, 1U, ACCESS_RW_EEPROM},   /* ID */
    {64U, 1U, ACCESS_RW},         /* Torque Enable */
    {65U, 1U, ACCESS_RW},         /* LED */
    {116U, 4U, ACCESS_RW},        /* Goal Position */
    {132U, 4U, ACCESS_R},         /* Present Position */
};

static uint8_t table[CONTROL_TABLE_SIZE];

/* DXL2_ERR_NONE if [address, address + length) may be written now. */
static uint8_t check_write(uint16_t address, uint16_t length, bool torque_on)
{
    uint8_t error = DXL2_ERR_NONE;

    for (size_t i = 0U; i < (sizeof(items) / sizeof(items[0])); ++i)
    {
        const struct item* item = &items[i];
        const bool overlaps = (item->address < (address + length))
                              && (address < (item->address + item->size));

        if (overlaps && ((item->access == ACCESS_R)
                         || ((item->access == ACCESS_RW_EEPROM) && torque_on)))
        {
            error = DXL2_ERR_ACCESS;
        }
    }
    return error;
}
```

Apply side effects after the write — for example, a new ID takes effect after the Status Packet has been sent with the old one. The simulator of DXL2 Studio (p. 42) (`tools/dxl2_gui/native/dxl2_sim.c`) is a complete reference implementation of an X-series Control Table, including REG_WRITE/ACTION, FACTORY_RESET, SYNC and BULK instructions.

5.4 Building instruction parameters

The codec encodes any instruction; it does not need to know its parameter layout. Build the parameters with the little-endian helpers and send them with `dxl2_link_send()`.

Instruction	Parameters
READ	address (2), length (2)
WRITE, REG_WRITE	address (2), data (n)
SYNC_READ, FAST_SYNC_READ	address (2), length (2), ID (1) × devices
SYNC_WRITE	address (2), length (2), then per device: ID (1), data (length)
BULK_READ, FAST_BULK_READ	per device: ID (1), address (2), length (2)
BULK_WRITE	per device: ID (1), address (2), length (2), data (length)
FACTORY_RESET, CLEAR, BACKUP	one or more option bytes, see the ROBOTIS specification

SYNC and BULK instructions are sent to the broadcast ID. A SYNC_WRITE builder that sets the 4-byte Goal Position (address 116 on X-series servos) of several servos at once:

```
/* Returns 0 on success, DXL2_PARSE_ERROR if the packet does not fit. */
int sync_write_u32(struct dxl2_link* link, uint16_t address, const uint8_t* ids,
                   const uint32_t* values, size_t count)
```

```
{
    uint8_t parameters[DXL2_MAX_PARAMETER_SIZE];
    size_t length = 4U;
    int result = DXL2_PARSE_ERROR;

    if ((4U + (count * 5U)) <= sizeof(parameters))
    {
        dxl2_put_u16(&parameters[0], address);
        dxl2_put_u16(&parameters[2], 4U); /* bytes per device */
        for (size_t i = 0U; i < count; ++i)
        {
            parameters[length] = ids[i];
            dxl2_put_u32(&parameters[length + 1U], values[i]);
            length += 5U;
        }
        result = dxl2_link_send(link, DXL2_BROADCAST_ID, DXL2_INST_SYNC_WRITE, parameters,
                                (uint16_t) length);
    }
    return result;
}
```

`dxl2_link_send()` also returns `DXL2_PARSE_ERROR` when byte stuffing makes the packet larger than `DXL2_MAX_PACKET_SIZE`, so the size check above is a fast path, not the only guard.

For SYNC_READ and BULK_READ, every addressed device answers with its own Status Packet, in the order of the request; collect them in your handler as shown for broadcast PING (p. 21). FAST_SYNC_READ and FAST_BULK_READ return one combined Status Packet whose parameter layout is described in the ROBOTIS specification; parse it in your handler.

5.5 Vendor-specific instructions

The parser and the encoder accept any instruction byte. If you control every node on a bus, you can define private instructions, for example to update firmware. Choose values that the specification does not use, keep them out of public buses, and document them — a future revision of the protocol may assign them. Do not add them to `dxl2_defs.h`, which only contains values from the specification.

5.6 Bridging two buses

A gateway that forwards packets between two buses needs no special support: each bus gets its own link, and each handler re-sends what it receives on the other link.

```
static void forward_a_to_b(const struct dxl2_packet* packet, void* context)
{
    (void) context;
    (void) dxl2_link_send(&bus_b, packet->id, packet->instruction, packet->parameters,
                          packet->parameter_length);
}
```

Re-encoding regenerates the stuffing and the CRC, so packets that arrived corrupted are never forwarded — the parser has already dropped them.

5.7 Changing the configuration

`DXL2_MAX_PACKET_SIZE` is the only build-time parameter. Raise it if your devices use large READ, SYNC or BULK transfers; lower it to save RAM on small MCUs. Define it for the whole build, so that the library and the application agree on the structure sizes:

```
target_compile_definitions(dxl2 PUBLIC DXL2_MAX_PACKET_SIZE=1024U)
```

The value must be between 10 and 65542; the build stops with an error otherwise. See Configuration (p. 30) for its effect on RAM and stack.

5.8 Contributing to the library

Changes to DXL2-C itself are welcome when they fix a bug, follow the specification more closely, or add something every user needs. The rules keep the library certifiable:

1. **Code.** C11, no new dependencies, no heap, no global mutable state. Public symbols use the `dxl2_` / `DXL2_` prefix and have Doxygen comments. Follow the MISRA rules used in the code: one `return` per function, parentheses around mixed operators, `(void)` for ignored results, `++`/`--` as separate statements, array indexing instead of pointer arithmetic.
2. **Tests.** Every behavior change comes with a GoogleTest test in `tests/unit/gtest_dxl2_*.cpp` (register new files in `dxl2.cmake`). Line and branch coverage must stay at 100 %. When the codec changes, update the Python twin in `tests/integration/dxl2/protocol.py`; the cross-check compares both.
3. **Analysis.** `misra-analyze` must report 0 violations. An unavoidable deviation is recorded in the deviation register (p. 57).
4. **Format.** `clang-format` with the repository `.clang-format` must report no changes.
5. **Documentation.** Update the affected pages in `docs/`, the stack table in Configuration (p. 30) when a function's frame changes, the diagrams (`python tools/docs/diagrams.py`) when the structure changes, and add an entry under `[Unreleased]` in `CHANGELOG.md`.
6. **Compatibility.** The public API is stable within a major version: do not remove or change public functions, structure members or return values without a major release.

Run the whole definition of done locally before opening a pull request:

```
cmake --preset "DXL2[UnitTests]" && cmake --build "build/DXL2[UnitTests]" --target
coverage
cmake --preset "DXL2[IntegrationTests]" && cmake --build "build/DXL2[IntegrationTests]" --
target integration_tests
cmake --preset "DXL2[Debug]" && cmake --build "build/DXL2[Debug]" --target misra-analyze
cmake --preset "DXL2[Examples]" && cmake --build "build/DXL2[Examples]" && ctest --test-
dir "build/DXL2[Examples]" -L example
```

Pull requests target the `development` branch. CI runs the same checks and blocks the merge if any of them fails.

CHAPTER 6

Configuration

Build-time parameters live in `inc/dxl2_config.h`. Each one is guarded by `#ifndef`, so override it with a compiler definition instead of editing the file:

```
target_compile_definitions(dxl2 PUBLIC DXL2_MAX_PACKET_SIZE=1024U)
```

Parameter	Default	Meaning
`DXL2_MAX_PACKET_SIZE`	`256U`	Largest complete wire packet (header to CRC) the parser and encoder accept; `10U..65542U`, checked at compile time

Derived values:

Macro	Value
`DXL2_MAX_PARAMETER_SIZE`	`DXL2_MAX_PACKET_SIZE - 10U`

6.1 Memory footprint

`DXL2_MAX_PACKET_SIZE` sizes three buffers:

Object	Size
`struct dxl2_parser`	`DXL2_MAX_PACKET_SIZE` + `size_t`
`struct dxl2_packet`	`DXL2_MAX_PARAMETER_SIZE` + 4 bytes
`struct dxl2_link`	parser + packet + configuration + two counters
`dxl2_link_send()` stack	`DXL2_MAX_PACKET_SIZE`
`dxl2_link_send_status()` stack	`DXL2_MAX_PARAMETER_SIZE` + the `dxl2_link_send()` frame

6.2 Stack usage

The library does not recurse and does not use variable-length arrays, so the stack depth is fixed. Measured with `-fstack-usage`, default `DXL2_MAX_PACKET_SIZE` (256), bytes per function frame:

Function	Cortex-M4 `-Os`	Cortex-M4 `-O0`	x86-64 `-O2`
`dxl2_link_send_status()`	272	296	336
`dxl2_link_send()`	296	312	368
`dxl2_encode()`	24	56	32
`dxl2_crc16()`	8	32	8

Function	Cortex-M4 `-Os`	Cortex-M4 `-O0`	x86-64 `-O2`
`dxl2_link_receive()`	24	32	80
`dxl2_parser_feed()` (with `dxl2_decode_packet()`)	32	32 + 56	80
`dxl2_link_init()`	16	32	48
other functions	0	16	8

Toolchains: Arm GNU Toolchain 15.2 (`-mcpu=cortex-m4 -mthumb`), GCC 14.2. At `-O0` the small static functions (`dxl2_get_u16()`, `dxl2_parser_reset()`) add 16 bytes each where they are called.

Worst-case call chains on Cortex-M4 `-Os`, without the callbacks:

Path	Chain	Stack
Transmit	`dxl2_link_send_status` → `dxl2_link_send` → `dxl2_encode` → `dxl2_crc16`	600 B + `write` call-back
Receive	`dxl2_link_receive` → `dxl2_parser_feed` → `dxl2_crc16`	64 B + packet handler
Receive and reply	`dxl2_link_receive` → handler → transmit chain	624 B + handler + `write`

The two large frames are the transmit buffers, so they scale with `DXL2_MAX_PACKET_SIZE`: about `2 × DXL2_MAX_PACKET_SIZE + 90` bytes for the transmit chain. Measure your own build with:

```
arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -Os -std=c11 -fstack-usage -Iinc -I. -c src/
dxl2_link.c src/dxl2_protocol.c
cat *.su
```

6.3 CMake options

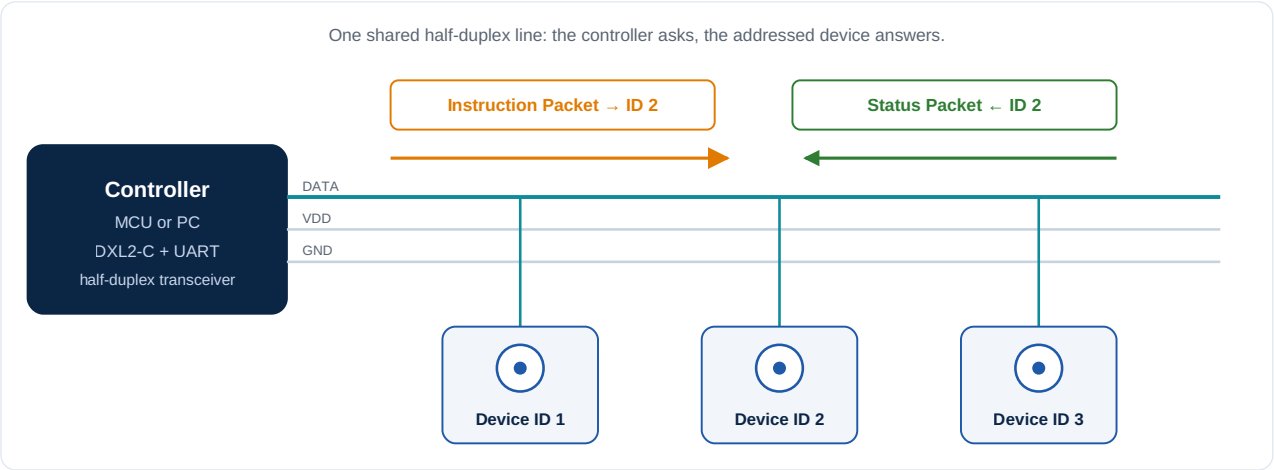
Option	De-fault	Meaning
`DXL2_UNIT_TESTS`	`OFF`	Build the GoogleTest suite and the `coverage` target
`DXL2_INTEGRATION_TESTS`	`OFF`	Register the pytest suite with CTest
`DXL2_MISRA_RULE_TEXTS`	empty	MISRA C:2012 rule texts file for the `misra-analyze` target
`DXL2_EXAMPLES`	`OFF`	Build the host examples (`examples/`) and register the bare-metal simulation with CTest
`DXL2_COVERAGE_MIN`	`100`	Line and branch coverage (%) below which the `coverage` target fails
`DXL2_MISRA_FAIL_ON_VIOLATION`	`ON`	Make `misra-analyze` fail when the report has a violation

CHAPTER 7

DYNAMIXEL Protocol 2.0

This chapter describes the Protocol 2.0 wire format and exactly what DXL2-C does with each part of it. The normative reference is the ROBOTIS Protocol 2.0 specification (docs.robotis.com/docs/dxl/protocol/protocol2/).

7.1 The bus

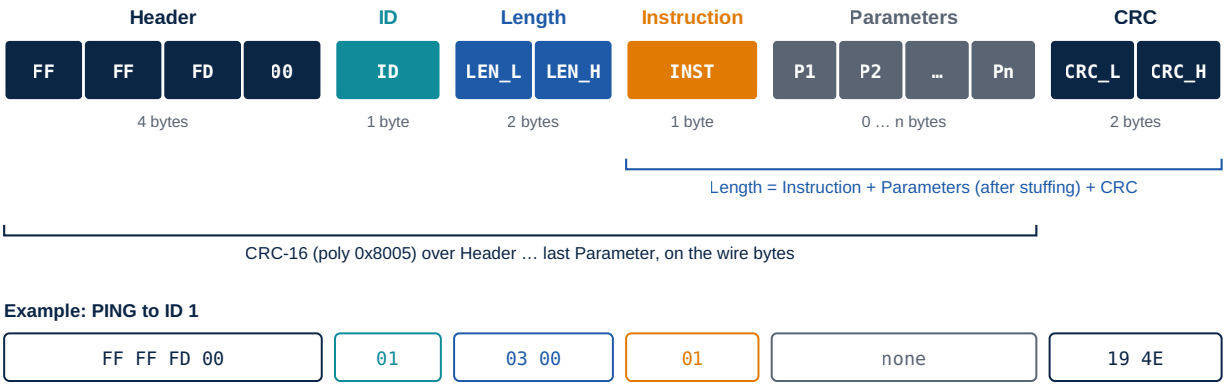


DYNAMIXEL devices share one serial line (TTL or RS-485) in a daisy chain. The line is half duplex: only one node may transmit at a time. Communication is always a request/response exchange:

1. The **controller** sends an **Instruction Packet** addressed to one ID (0–252) or to the broadcast ID 254.
2. The addressed **device** executes it and, unless the instruction was broadcast or its Status Return Level says otherwise, answers with a **Status Packet** after its Return Delay Time.
3. The controller waits for the reply with a timeout before it sends the next request.

The UART format is 8 data bits, no parity, one stop bit; the bit rate is configured per device (57600 bit/s by default on most models, up to several Mbit/s).

7.2 Packet



Field	Offset	Size	Meaning
Header	0	4	Always `FF FF FD 00`
ID	4	1	Device ID `0..252`; `0xFE` = broadcast (`DXL2_BROADCAST_ID`)
Length	5	2	Little endian. Counts the instruction, the stuffed parameters and the CRC, so the minimum is 3
Instruc-tion	7	1	`DXL2_INST_*`; `0x55` marks a Status Packet
Paramet-ers	8	n	Instruction parameters; for a Status Packet the first byte is the error
CRC	7 + Length − 2	2	Little endian, over all bytes from the header up to the CRC (stuffed)

A complete packet is therefore `7 + Length` bytes long; the shortest one (no parameters) is 10 bytes. All multi-byte values inside the parameters — addresses, lengths, register values — are little endian too: use `dxl2_get_u16/u32` and `dxl2_put_u16/u32`.

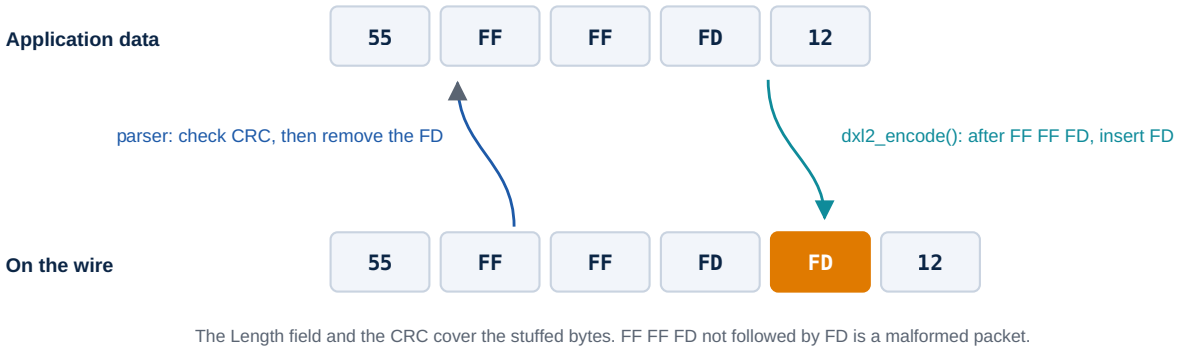
In `struct dxl2_packet` the library gives you the ID, the instruction and the **unstuffed** parameters with their count. The header, Length and CRC are consumed by the parser.

7.3 CRC-16

Polynomial `0x8005`, initial value `0`, not reflected, no final XOR (the CRC-16/BUYPASS variant). It is computed over the stuffed packet. The ROBOTIS PING reference packet `FF FF FD 00 01 03 00 01` has CRC `0x4E19` (transmitted as `19 4E`). The CRC is computed bit by bit without a lookup table, which keeps the code small; its execution time depends only on the length.

7.4 Byte stuffing

So that a header never appears inside a packet, the sender inserts an extra `0xFD` after every `FF FF FD` sequence in the instruction/parameter area, and the receiver removes it. The Length field counts the stuffed bytes.



The rule applies to the whole area after the Length field — instruction and parameters — so a sequence that starts in the instruction byte is stuffed too.

`dxl2_encode()` stuffs while it writes. The parser checks the CRC over the stuffed bytes first, then unstuffs. An `FF FF FD` that is not followed by `FD` is a malformed packet (`DXL2_PARSE_ERROR`).

7.5 Instructions

Value	Name	Value	Name
`0x01`	PING	`0x55`	STATUS (reply)
`0x02`	READ	`0x82`	SYNC_READ
`0x03`	WRITE	`0x83`	SYNC_WRITE
`0x04`	REG_WRITE	`0x8A`	FAST_SYNC_READ
`0x05`	ACTION	`0x92`	BULK_READ
`0x06`	FACTORY_RESET	`0x93`	BULK_WRITE
`0x08`	REBOOT	`0x9A`	FAST_BULK_READ
`0x10`	CLEAR		
`0x20`	BACKUP		

Parameter layouts of the common instructions:

Instruction	Parameters
PING	none; reply data: model number (2), firmware version (1)
READ	address (2), length (2); reply data: `length` bytes
WRITE / REG_WRITE	address (2), data (n)
SYNC_READ	address (2), length (2), ID list
SYNC_WRITE	address (2), length (2), then per device: ID (1) + data (`length`)

The library encodes and decodes every instruction the same way; interpreting the parameters (the Control Table) is up to the application.

7.6 Status Packet error byte

Value	DXL2_ERR_*	Meaning
0x00	NONE	OK
0x01	RESULT_FAIL	Instruction could not be processed
0x02	INSTRUCTION	Undefined instruction, or ACTION without REG_WRITE
0x03	CRC	CRC mismatch
0x04	DATA_RANGE	Value outside the register range
0x05	DATA_LENGTH	Data shorter than the register
0x06	DATA_LIMIT	Value outside the configured limit
0x07	ACCESS	Read-only / write-only / torque-locked register
bit 7	ALERT_BIT	Device has a hardware error (Hardware Error Status)

7.7 Parser behavior

`dxl2_parser_feed()` takes one byte at a time:

1. Bytes before a complete `FF FF FD 00` header are discarded (sliding window), so the parser resynchronizes after noise or a lost byte.
2. After 7 bytes it reads Length. A Length below 3 or one that would exceed `DXL2_MAX_PACKET_SIZE` (256 bytes on the wire by default, see `inc/dxl2_config.h`) is rejected at once.
3. When the whole packet is in, it checks the CRC, then unstuffs into `struct dxl2_packet` and returns `1`.
4. Any error returns `DXL2_PARSE_ERROR` and resets the parser; keep feeding bytes, the next packet is found normally.

The bytes of a rejected packet are discarded, including any that follow its header. If a byte was lost so that the Length field counts into the next packet, that packet is rejected too and the parser resynchronizes on the one after it. A controller should therefore treat a missing reply as a timeout and retry, as the ROBOTIS DynamixelSDK does.

The output packet is not touched while a packet is incomplete or when its CRC is wrong. After any error its contents are undefined: use it only when `dxl2_parser_feed()` returns `1`.

7.8 Reference packets (from the specification, used in the tests)

Packet	Bytes
PING to ID 1	`FF FF FD 00 01 03 00 01 19 4E`

Packet	Bytes
Status of ID 1 (model 1030, firmware 38)	`FF FF FD 00 01 07 00 55 00 06 04 26 65 5D`

CHAPTER 8

Integration Guide

The library does not touch hardware. An integration has three parts:

1. **Receive:** move bytes from the transport into `dxl2_link_receive()`. Do it in a thread or the main loop, not in an interrupt handler: the packet handler runs in the caller's context and usually sends a reply.
2. **Transmit:** a `dxl2_write_fn` that writes a complete packet. Serialize it (mutex) if more than one thread can send.
3. **Application:** a `dxl2_packet_handler_t` that interprets the instruction for its IDs (Control Table) and answers with `dxl2_link_send_status()`.

Complete, buildable programs for each pattern below are described in Examples (p. 39).

8.1 Zephyr: UART with an RX interrupt, byte queue and RX thread

`examples/zephyr` is the structure used in the SmartManipulator-X firmware, reduced to the library API. The ISR only queues bytes; a thread owns the link. Select the UART with a `dxl2,uart` chosen node (see `app.overlay.example`) and build it with:

```
west build -b <board> examples/zephyr -- -DDTC_OVERLAY_FILE=<board>.overlay
```

Notes from the hardware bring-up:

- Size the byte queue for at least one maximum packet plus the time the handler blocks (for example while it waits for a motor driver). A dropped byte makes the next packet fail its CRC; the parser recovers on the next header.
- `uart_poll_out()` is simple and deterministic. At 230400 bit/s a 256-byte packet takes about 11 ms; use interrupt or DMA TX if that is too long.
- Check the real bit rate of the UART divisor. A 16550-style UART at a 50 MHz clock gives 223214 bit/s for 230400 (-3.1 %), which is too far off; some UARTs (for example the SmartFusion2 MMUART) have a fractional divisor.
- Text output (console, logs, `printk`) on the same UART corrupts the framing.

8.2 Adding the library to a CMake build (Zephyr or any other)

`dxl2.cmake` defines the `dxl2` static library target with its include directories (`.` and `inc/`) and the version definitions:

```
include(path/to/dxl2/dxl2.cmake)
target_link_libraries(app PRIVATE dxl2)
```

In a Zephyr application also link the library to `zephyr_interface`, so it is compiled with the board's architecture flags (see `examples/zephyr/CMakeLists.txt`):

```
target_link_libraries(dxl2 PUBLIC zephyr_interface)
```

Without CMake, compile `src/*.c` and add the repository root and `inc/` to the include path. Buffer sizes are set in `inc/dxl2_config.h` and can be overridden with compiler definitions (for example `-DDXL2_MAX_PACKET_SIZE=1024U`).

8.3 Bare metal / other RTOS

The same pattern applies (`examples/bare_metal`): ring buffer from the RX interrupt, drain it from the main loop into `dxl2_link_receive()`, and a blocking or buffered write function. On a host (Linux/Windows serial port, `examples/host_serial`) read whatever the port returns and pass it to `dxl2_link_receive()` in one call.

8.4 Answering a device request

A device implementation typically does this in the handler:

```
static void on_packet(const struct dxl2_packet *packet, void *context)
{
    const uint8_t *p = packet->parameters;

    if (packet->id != MY_ID && packet->id != DXL2_BROADCAST_ID) {
        return;
    }
    switch (packet->instruction) {
    case DXL2_INST_PING:
        /* model number (2), firmware version (1) */
        break;
    case DXL2_INST_READ:
        if (packet->parameter_length != 4U) {
            dxl2_link_send_status(&link, MY_ID, DXL2_ERR_DATA_LENGTH, NULL, 0U);
            return;
        }
        /* address = dxl2_get_u16(&p[0]), length = dxl2_get_u16(&p[2]) */
        break;
    default:
        dxl2_link_send_status(&link, MY_ID, DXL2_ERR_INSTRUCTION, NULL, 0U);
        break;
    }
}
```

Do not answer instructions sent to the broadcast ID, except PING (all devices answer, ordered by ID) and the read instructions that name the device in their ID list. Respect the device's Status Return Level and Return Delay Time registers if it emulates them.

CHAPTER 9

Examples

The repository contains three complete example programs in `examples/` (github.com/AntonVynohradov/DXL2-C/tree/main/examples). They show the integration patterns from the integration guide (p. 37) in working code.

Example	Role	Platform	Built and run in CI
Bare metal	Device	Any MCU; runs on a PC against a simulated board	Yes, as a self-checking test
Host serial port	Controller	Linux, macOS, Windows	Built
Zephyr	Device	Any Zephyr board with a UART	No, needs a Zephyr workspace

Build the host examples and run the bare-metal test:

```
cmake --preset "DXL2[Examples]"
cmake --build "build/DXL2[Examples]"
ctest --test-dir "build/DXL2[Examples]" -L example --output-on-failure
```

9.1 Bare metal

`examples/bare_metal` is a DYNAMIXEL device for a microcontroller without an operating system. It answers PING, READ and WRITE on a 128-byte Control Table whose first 64 bytes are read-only.

File	Contents
`main.c`	The application: ring buffer, main loop, packet handler, Control Table
`board.h`	The four board functions the application needs
`board_host.c`	A PC implementation of `board.h` that turns the example into a test

Structure. The UART interrupt calls `app_uart_rx_isr()` for every byte. It stores the byte in a single-producer, single-consumer ring buffer and returns. The main loop takes bytes out of the buffer and passes them to `dxl2_link_receive()`; when the buffer is empty it calls `board_idle()`, which on a real MCU executes `WFI` until the next interrupt. The packet handler therefore runs in the main loop, never in the interrupt.

```
for (;;)
{
    if (rx_tail != rx_head)
    {
        const uint8_t byte = rx_queue[rx_tail];

        rx_tail = (uint16_t) ((rx_tail + 1U) & (RX_QUEUE_SIZE - 1U));
        dxl2_link_receive(&link, &byte, 1U);
    }
```

```
    else
    {
        board_idle();
    }
}
```

The ring buffer needs no lock on a single-core 32-bit MCU: the interrupt writes only `rx_head`, the main loop writes only `rx_tail`, and 16-bit loads and stores are atomic. On an 8-bit MCU, disable the RX interrupt around the comparison.

Porting. Implement `board.h` for your MCU:

Function	What it must do
`board_init(baud_rate)`	Configure the UART (8N1) and enable its RX interrupt; call `app_uart_rx_isr()` from it
`board_uart_write(data, length)`	Transmit the bytes and wait until the last one has left the shift register, then switch a half-duplex transceiver back to receive
`board_idle()`	Sleep until the next interrupt (`__WFI()` on Cortex-M)

The simulated board. `board_host.c` acts as the controller. Each time the main loop goes idle, it checks the reply to the previous request and injects the next one — preceded by line noise, so the parser must resynchronize. The test covers PING (direct and broadcast), a PING for another ID, READ, WRITE, a broadcast WRITE without a reply, a write to a read-only address, out-of-range and malformed READs, and an unsupported instruction:

```
bare-metal example on the host simulator, 57600 bit/s
PING                        ok
broadcast PING              ok
PING to another ID          ok
READ model number           ok
WRITE RAM                   ok
READ RAM                    ok
broadcast WRITE             ok
READ after broadcast WRITE  ok
WRITE read-only ID          ok
READ out of range           ok
READ with bad length        ok
unsupported instruction     ok
0 failure(s)
```

9.2 Host serial port

`examples/host_serial` builds `dxl2_ping`, a command-line controller that sends a PING through a serial port and lists every device that answers.

```
dxl2_ping /dev/ttyUSB0            # broadcast PING at 57600 bit/s
dxl2_ping COM3 1000000 1          # PING ID 1 at 1 Mbit/s
```

Example output with two XL430-W250 servos on the bus:

```
ID   1: model 1060, firmware 52
ID   2: model 1060, firmware 52
2 device(s) answered, 0 invalid packet(s)
```

File	Contents
`main.c`	Argument parsing, the link, the Status Packet handler
`serial_port.h`	Four functions: open, write, read with timeout, close
`serial_port_posix.c`	Linux and macOS implementation with `termios` and `poll()`
`serial_port_win32.c`	Windows implementation with the Win32 communications API

After sending, the program reads with a 50 ms timeout and passes whatever arrives to `dxl2_link_receive()`. It stops after four empty reads in a row, which leaves enough time for every device to answer a broadcast PING. The handler ignores packets whose instruction is not `0x55`, which removes the echo of the request on single-wire adapters. The exit code is 0 if at least one device answered.

The POSIX version supports 9600, 57600, 115200 and 230400 bit/s, and 1, 2, 3 and 4 Mbit/s where `termios` defines them (Linux). The Windows version accepts any rate the adapter supports.

9.3 Zephyr

`examples/zephyr` is a Zephyr application that makes a board a DYNAMIXEL device answering PING. It is the structure used in production firmware, reduced to the library API.

- The UART interrupt reads the FIFO and puts each byte into a `k_msgq`. It never calls the library.
- An RX thread, created with `K_THREAD_DEFINE`, owns the link: it waits on the queue and passes each byte to `dxl2_link_receive()`. The packet handler runs in this thread.
- The write function sends with `uart_poll_out()` under a mutex, so other threads may send on the same link.

Select the UART with a `dxl2,uart` chosen node in a board overlay (a template is in `app.overlay.example`) and build with `west`:

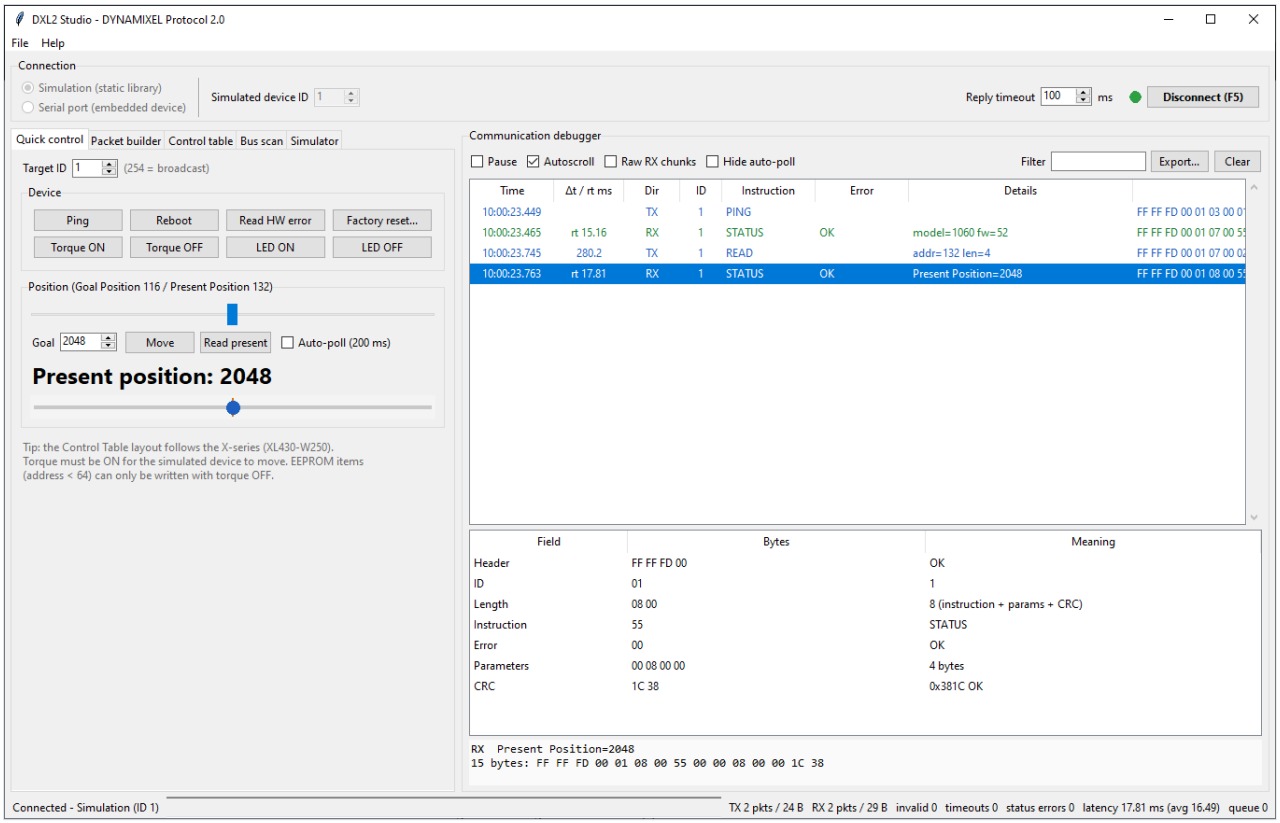
```
west build -b <board> examples/zephyr -- -DDTC_OVERLAY_FILE=<board>.overlay
```

The application's `CMakeLists.txt` includes `dxl2.cmake` and links the `dxl2` target to `zephyr_interface`, so the library is compiled with the board's architecture flags. The console must not use the same UART: text output corrupts the packet framing.

CHAPTER 10

DXL2 Studio

DXL2 Studio is a desktop application for testing and debugging DYNAMIXEL Protocol 2.0 communication. It is built on the DXL2-C static library itself: every Instruction Packet it sends is encoded by `dxl2_link_send()` and every byte it receives goes through `dxl2_link_receive()`. What you see in the tool is exactly what the library does.



10.1 Modes

Mode	What runs on each side	Hardware
Simulation	The controller and a simulated DYNAMIXEL device (XL430-W250 Control Table) each run the library on their own link	None
Serial port	The controller runs on the library and talks to a real device over a COM port: a board with DXL2-C, a U2D2 adapter, or a servo	USB-to-serial adapter

The simulation makes it possible to learn the protocol, try instructions and test error handling without any hardware. The serial mode turns the same tool into a bus analyzer for your own firmware.

10.2 Installation and start

DXL2 Studio needs Python 3.8 or newer with Tkinter, CMake 3.20 or newer and a C11 compiler (GCC/MinGW-w64, Clang or MSVC, with the same bitness as Python).

```
python -m pip install -r tools/dxl2_gui/requirements.txt   # pyserial, for the serial mode
python tools/dxl2_gui/dxl2_gui.py
```

On the first start the tool offers to build its native bridge, a shared library that links the static `libdxl2.a`. You can also build it by hand:

```
cmake -S tools/dxl2_gui/native -B tools/dxl2_gui/build
cmake --build tools/dxl2_gui/build --config Release
```

Command-line options select the mode and connect at once:

```
python tools/dxl2_gui/dxl2_gui.py --mode sim --id 1 --connect
python tools/dxl2_gui/dxl2_gui.py --mode serial --port COM3 --baud 1000000 --connect
```

In VS Code, open the task menu (**Ctrl+Shift+B**), run *Python: Setup venv* once, then *GUI: Run (Simulation)* or *GUI: Run (Serial port)*.

On Debian and Ubuntu, install the system packages first and add your user to the `dialout` group to use USB serial adapters without root:

```
sudo apt install python3-venv python3-tk cmake build-essential
sudo usermod -aG dialout $USER   # log out and back in afterwards
```

10.3 Window layout

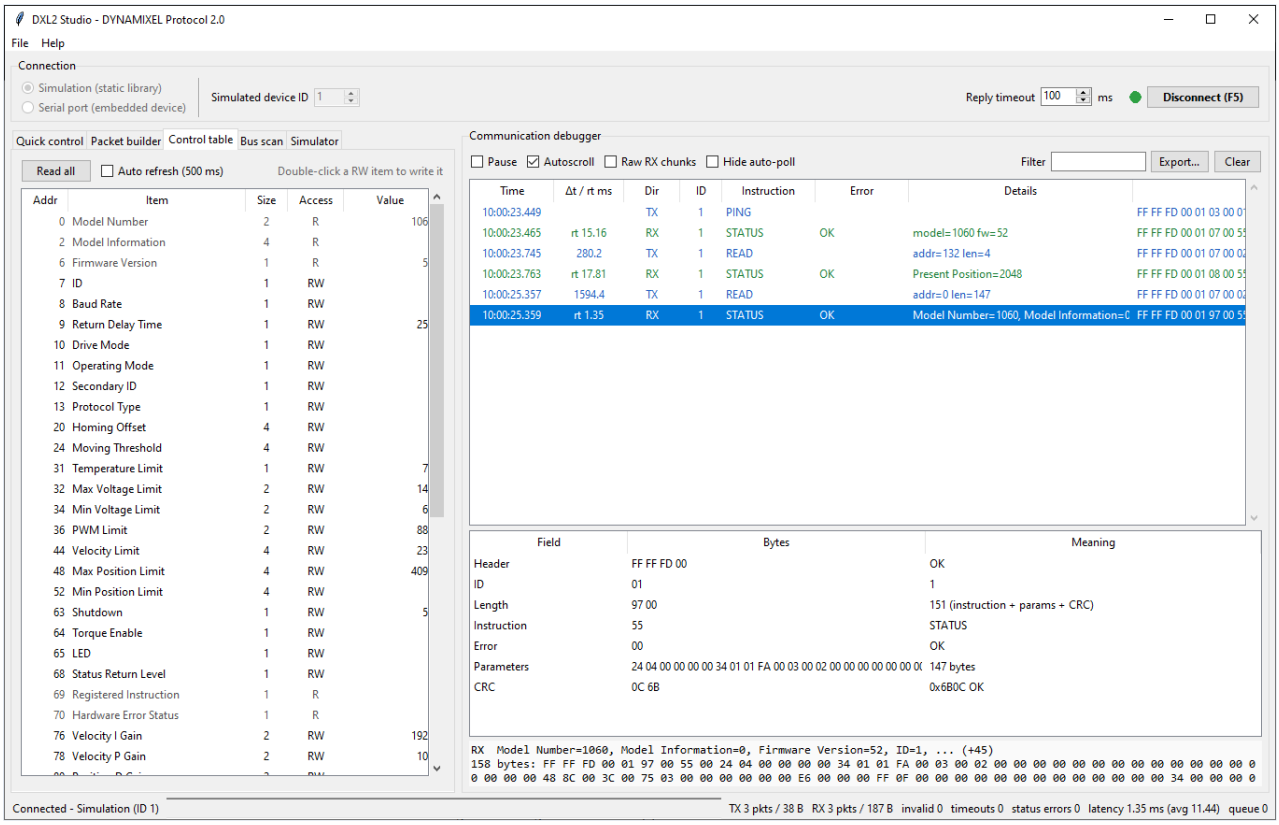
The window has three areas:

- **Connection bar** (top): mode, simulated device ID or serial port and baud rate, reply timeout, and the *Connect* button (**F5**).
- **Control tabs** (left): quick control, packet builder, Control Table, bus scan and simulator settings.
- **Communication debugger** (right): the packet log, the field-by-field breakdown of the selected packet and its raw bytes. The status bar below shows packet and byte counters, invalid packets, timeouts, status errors and latency.

Quick control

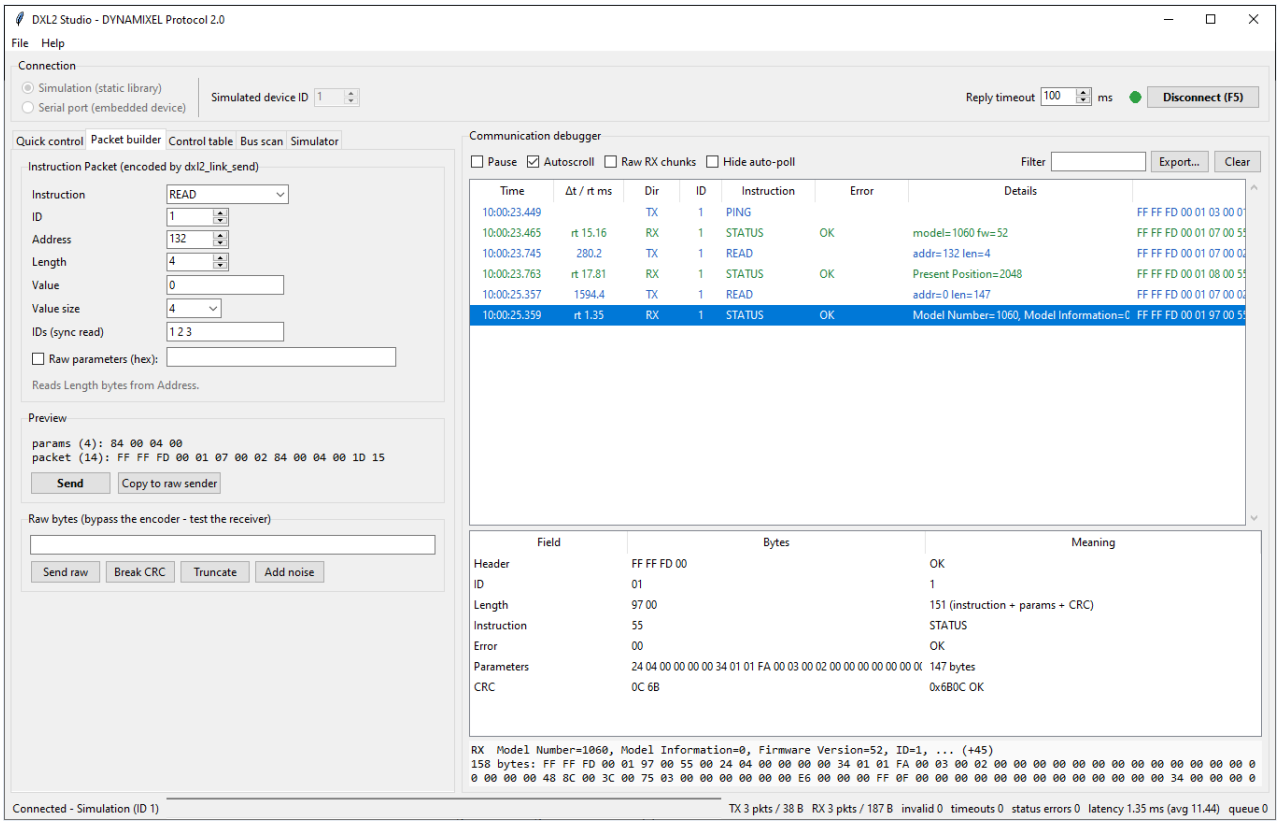
Ping, reboot, factory reset, torque and LED on and off, and the Hardware Error Status. A slider sends the Goal Position; *Auto-poll* reads the Present Position every 200 ms.

Control Table



Read all reads every item with one READ instruction. Items whose value changed since the last read are highlighted, and *Auto refresh* repeats the read every 500 ms. Double-click a read-write item to write a new value. The view uses the X-series layout; for other models use the packet builder.

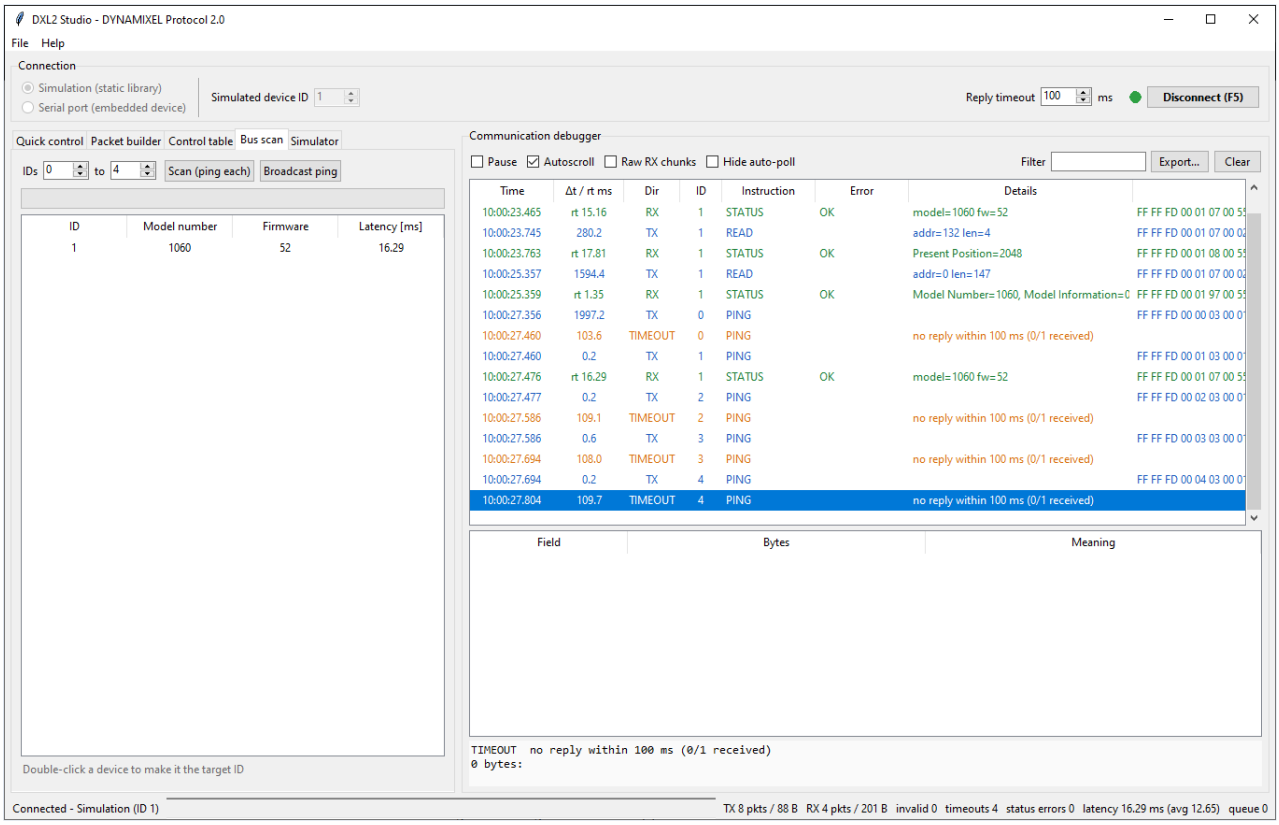
Packet builder



Choose an instruction and fill in its fields — or type raw parameters in hex for SYNC and BULK instructions — and the preview shows the exact packet that `dxl2_encode()` produces, including byte stuffing and CRC.

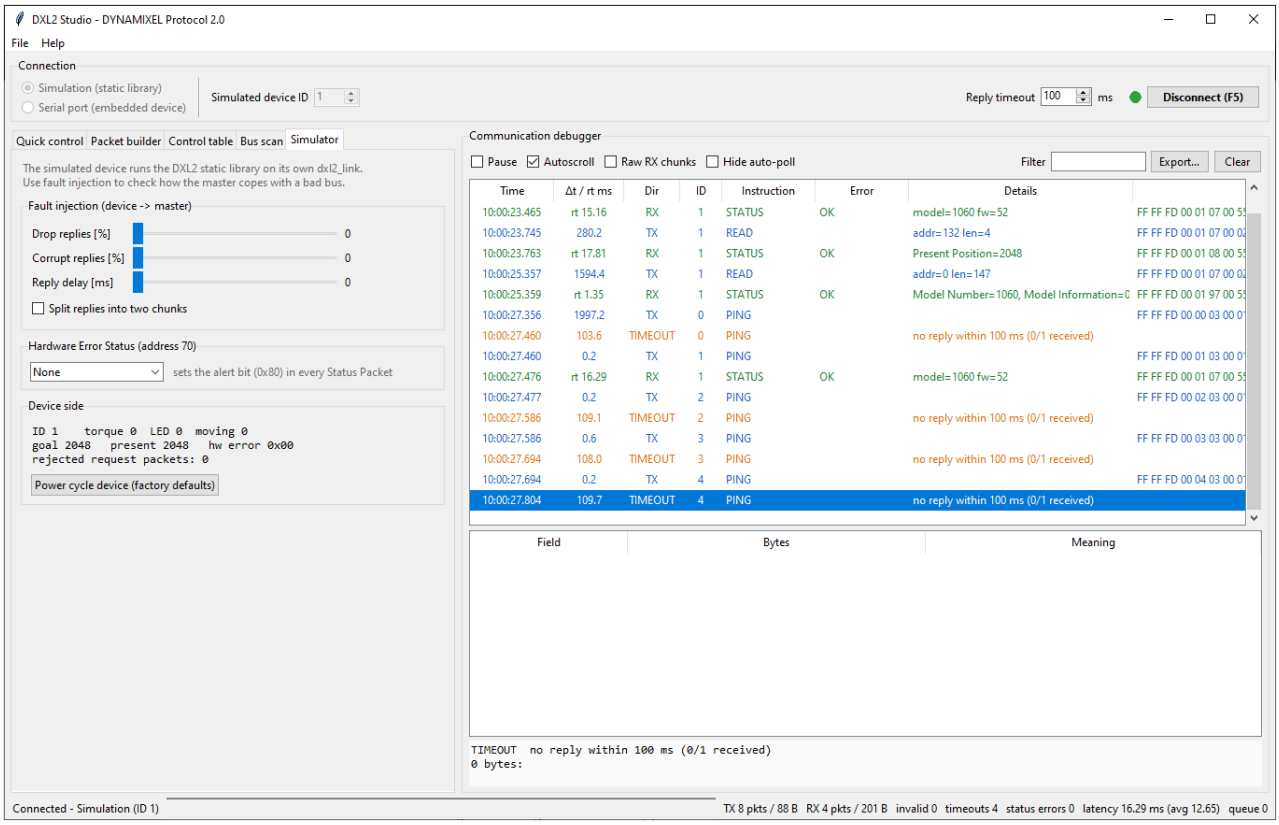
The *Raw bytes* sender bypasses the encoder. Its *Break CRC*, *Truncate* and *Add noise* buttons create invalid input, so you can watch how a receiver — the simulated device or your own firmware — rejects it and recovers.

Bus scan



Ping every ID in a range, or send one broadcast PING. The results list the model number, firmware version and latency of every device. Double-click a result to make it the target of the other tabs.

Simulator



In simulation mode this tab injects faults into the replies of the simulated device: drop a percentage of replies, corrupt one bit, add a delay, or split a reply into two chunks. You can also set a Hardware Error Status (which sets the alert bit in every Status Packet), watch the device state and power-cycle the device.

The simulated device implements PING, READ, WRITE, REG_WRITE/ACTION, FACTORY_RESET, REBOOT, CLEAR, BACKUP, SYNC_READ/WRITE and BULK_READ/WRITE, applies Status Return Level, the EEPROM lock while torque is on, and the ID, torque and LED range checks. It answers FAST_SYNC_READ and FAST_BULK_READ with an Instruction Error.

10.4 Communication debugger

Every log entry has a time stamp, the time since the previous entry and, for replies, the round-trip time (`rt`). Entries are color coded:

Entry	Meaning
TX	Instruction Packet sent
RX	Status Packet received; interpreted against its request (for example `Present Position=2048`)
TIMEOUT	No reply within the reply timeout
ERR	Bytes rejected by `dxl2_link_receive()` — on the controller or on the simulated device
INFO	Connection and tool messages

Select an entry to see it field by field — header, ID, Length check, instruction, error, parameters (with a note when byte stuffing is present) and CRC check. *Raw RX chunks* shows the bytes exactly as the transport delivered them, before decoding. The log can be filtered, paused, cleared and exported to CSV or text (**Ctrl+E**).

For serial ports, *Discard echo* removes your own bytes from the receive stream on single-wire half-duplex buses, and the port field accepts pyserial URLs such as `socket://192.168.1.10:5000` for serial-over-TCP bridges.

10.5 How the tool uses the library

```
tools/dxl2_gui/
├── dxl2_gui.py          entry point
├── dxl2gui/
│   ├── app.py           Tkinter user interface
│   ├── session.py       request queue, timeouts, statistics, log entries
│   ├── transport.py     simulation and serial transports, fault injection
│   ├── native.py        ctypes binding and CMake build helper
│   └── decode.py        names, packet breakdown, value formatting
└── native/
    ├── dxl2_host.c/.h   controller side: dxl2_link_send / dxl2_link_receive
    ├── dxl2_sim.c/.h    simulated device on its own dxl2_link
    └── CMakeLists.txt   links the static dxl2 target into dxl2_gui_native
```

The native bridge exports only `dxl2_host_*` and `dxl2_sim_*` functions. Python never depends on the memory layout of the library structures: it asks the bridge for their sizes and treats them as opaque. `dxl2_sim.c` is also a useful reference if you write your own device firmware: it is a complete X-series Control Table on top of the public API.

CHAPTER 11

API Reference

This chapter documents every public function, type and macro of DXL2-C 1.0. Include the umbrella header to use them:

```
#include "dxl2.h"
```

Header	Contents
`dxl2.h`	Includes all headers below; library version
`dxl2_config.h`	Build-time configuration
`dxl2_defs.h`	IDs, instructions, error codes, little-endian helpers
`dxl2_protocol.h`	CRC-16, stream parser, encoder
`dxl2_link.h`	Transport-independent link

The reference generated from the source comments by Doxygen is available on the website under *Reference → Doxygen*.

11.1 Version — `dxl2.h`

Macro	Value in 1.0.0	Meaning
`DXL2_VERSION_MAJOR`	`1`	Incompatible API changes
`DXL2_VERSION_MINOR`	`0`	Compatible additions
`DXL2_VERSION_PATCH`	`0`	Fixes
`DXL2_VERSION`	`"1.0.0"`	Version string

CMake builds take the values from the `vX.Y.Z` Git tag; other builds use the fallbacks in `dxl2.h`.

11.2 Configuration — `dxl2_config.h`

Macro	Default	Meaning
`DXL2_MAX_PACKET_SIZE`	`256U`	Largest complete wire packet, header to CRC, that the parser accepts and the link sends. Range `10U..65542U`, checked at compile time. Override with a compiler definition.

11.3 Definitions — `dxl2_defs.h`

Device IDs

Macro	Value	Meaning
`DXL2_MAX_ID`	`252U`	Highest ID a device may use
`DXL2_BROADCAST_ID`	`0xFEU`	Broadcast ID: every device executes the instruction

Instructions

Macro	Value	Macro	Value
`DXL2_INST_PING`	`0x01U`	`DXL2_INST_STATUS`	`0x55U`
`DXL2_INST_READ`	`0x02U`	`DXL2_INST_SYNC_READ`	`0x82U`
`DXL2_INST_WRITE`	`0x03U`	`DXL2_INST_SYNC_WRITE`	`0x83U`
`DXL2_INST_REG_WRITE`	`0x04U`	`DXL2_INST_FAST_SYNC_READ`	`0x8AU`
`DXL2_INST_ACTION`	`0x05U`	`DXL2_INST_BULK_READ`	`0x92U`
`DXL2_INST_FACTORY_RESET`	`0x06U`	`DXL2_INST_BULK_WRITE`	`0x93U`
`DXL2_INST_REBOOT`	`0x08U`	`DXL2_INST_FAST_BULK_READ`	`0x9AU`
`DXL2_INST_CLEAR`	`0x10U`		
`DXL2_INST_BACKUP`	`0x20U`		

Status error byte

Macro	Value	Meaning
`DXL2_ERR_NONE`	`0x00U`	No error
`DXL2_ERR_RESULT_FAIL`	`0x01U`	The instruction could not be processed
`DXL2_ERR_INSTRUCTION`	`0x02U`	Undefined instruction, or ACTION without REG_WRITE
`DXL2_ERR_CRC`	`0x03U`	CRC mismatch
`DXL2_ERR_DATA_RANGE`	`0x04U`	Data outside the register range
`DXL2_ERR_DATA_LENGTH`	`0x05U`	Data shorter than the register
`DXL2_ERR_DATA_LIMIT`	`0x06U`	Data outside the configured limit
`DXL2_ERR_ACCESS`	`0x07U`	Access denied (read-only, write-only, locked)
`DXL2_ERR_ALERT_BIT`	`0x80U`	Flag ORed into the error byte: the device has a hardware error

Little-endian helpers

```c
static inline uint16_t dxl2_get_u16(const uint8_t* data);
static inline uint32_t dxl2_get_u32(const uint8_t* data);
```

```c
static inline void dxl2_put_u16(uint8_t* data, uint16_t value);
static inline void dxl2_put_u32(uint8_t* data, uint32_t value);
```

Read or write a 16- or 32-bit value in little-endian byte order, independent of the CPU's byte order and alignment. `data` must point to 2 (or 4) accessible bytes; the helpers do not check it, like `memcpy()`.

11.4 Codec — `dxl2_protocol.h`

Limits and errors

Macro	Value	Meaning
`DXL2_MAX_PARAMETER_SIZE`	`DXL2_MAX_PACKET_SIZE - 10U`	Largest unstuffed parameter block (246 by default)
`DXL2_PARSE_ERROR`	`-1`	Invalid argument, malformed packet, CRC mismatch or insufficient capacity

`struct dxl2_packet`

One decoded packet, with byte stuffing removed.

```c
struct dxl2_packet
{
    uint8_t id;                                 /* source or destination ID */
    uint8_t instruction;                        /* instruction, 0x55 for Status */
    uint16_t parameter_length;                  /* number of valid bytes in parameters */
    uint8_t parameters[DXL2_MAX_PARAMETER_SIZE]; /* unstuffed parameters */
};
```

For a Status Packet, `parameters[0]` is the error byte and the data follows from `parameters[1]`.

`struct dxl2_parser`

State of the incremental parser for one byte stream. Treat the members as private. Initialize it with `dxl2_parser_init()` before use.

```c
struct dxl2_parser
{
    uint8_t buffer[DXL2_MAX_PACKET_SIZE]; /* packet in progress, header to CRC */
    size_t length;                        /* number of bytes in buffer */
};
```

`dxl2_crc16()`

```c
uint16_t dxl2_crc16(const uint8_t* data, size_t length);
```

Computes the Protocol 2.0 CRC-16: polynomial `0x8005`, initial value `0`, not reflected, no final XOR.

Parameter	Description
`data`	Bytes covered by the CRC
`length`	Number of bytes

Returns the CRC, or `0` (the initial value) when `data` is `NULL`.

`dxl2_parser_init()`

```
void dxl2_parser_init(struct dxl2_parser* parser);
```

Resets a parser before a new byte stream, or to discard a partial packet. `NULL` is ignored.

`dxl2_parser_feed()`

```
int dxl2_parser_feed(struct dxl2_parser* parser, uint8_t byte, struct dxl2_packet*
packet);
```

Feeds one byte to the parser. The parser searches for the header, checks the Length field, waits for the complete packet, validates the CRC and removes byte stuffing. Keep feeding bytes after an error: the parser resynchronizes on the next header.

Parameter	Description
`parser`	Initialized parser state
`byte`	Next byte of the stream
`packet`	Output packet; its contents are valid only when the call returns `1`

Return	Meaning
`1`	A complete, valid packet is in `packet`
`0`	More bytes are needed
`DXL2_PARSE_ERROR`	`NULL` argument, invalid Length, CRC mismatch or invalid byte stuffing; the parser has been reset

`dxl2_encode()`

```
int dxl2_encode(uint8_t id, uint8_t instruction, const uint8_t* parameters,
                uint16_t parameter_length, uint8_t* output, size_t output_capacity,
                size_t* output_length);
```

Encodes a complete wire packet: header, ID, Length, instruction, parameters with byte stuffing, and CRC.

Parameter	Description
`id`	Packet ID

Parameter	Description
`instruction`	Instruction byte
`parameters`	Unstuffed parameters; may be `NULL` when `parameter_length` is 0
`parameter_length`	Number of parameter bytes, at most `DXL2_MAX_PARAMETER_SIZE`
`output`	Buffer that receives the packet
`output_capacity`	Size of `output`; at least 10
`output_length`	Receives the packet length on success

Return	Meaning
`0`	Packet encoded; `*output_length` bytes of `output` are valid
`DXL2_PARSE_ERROR`	`NULL` argument, too many parameters, or the stuffed packet does not fit in `output_capacity`

A buffer of `DXL2_MAX_PACKET_SIZE` bytes holds every packet the link can send. The encoder never writes beyond `output_capacity`.

11.5 Link — `dxl2_link.h`

`dxl2_write_fn`

```
typedef int (*dxl2_write_fn)(void* context, const uint8_t* data, size_t length);
```

Writes one complete wire packet to the transport. Called by `dxl2_link_send()` and `dxl2_link_send_status()`.

Parameter	Description
`context`	`write_context` from the link configuration
`data`	The encoded packet; valid only during the call
`length`	Number of bytes

Returns `0` on success or a negative value on failure; the value is passed back to the caller of the send function.

`dxl2_packet_handler_t`

```
typedef void (*dxl2_packet_handler_t)(const struct dxl2_packet* packet, void* context);
```

Called by `dxl2_link_receive()` for every valid packet. `packet` is valid only until the handler returns; copy what you need. The handler may send on the same link.

Parameter	Description
`packet`	The decoded packet
`context`	`handler_context` from the link configuration

`struct dxl2_link_config`

```
struct dxl2_link_config
{
    dxl2_write_fn write;           /* NULL makes the link receive-only */
    void* write_context;           /* passed to write */
    dxl2_packet_handler_t handler; /* NULL discards packets (they are still counted) */
    void* handler_context;         /* passed to handler */
};
```

`struct dxl2_link`

The state of one link: a copy of the configuration, a parser, the last packet and two counters. Allocate it yourself and treat its members as private.

`dxl2_link_init()`

```
int dxl2_link_init(struct dxl2_link* link, const struct dxl2_link_config* config);
```

Initializes a link: copies the configuration, resets the parser and the counters.

Return	Meaning
`0`	Link ready
`DXL2_PARSE_ERROR`	`link` or `config` is `NULL`

`dxl2_link_receive()`

```
void dxl2_link_receive(struct dxl2_link* link, const uint8_t* data, size_t length);
```

Feeds received bytes, in any chunking, to the link's parser. For each valid packet it increments the receive counter and calls the handler; each rejected packet increments the invalid-packet counter. Does nothing if `link` or `data` is `NULL`.

Call it for a given link from one context only, and not from an interrupt handler.

`dxl2_link_send()`

```
int dxl2_link_send(struct dxl2_link* link, uint8_t id, uint8_t instruction,
                   const uint8_t* parameters, uint16_t parameter_length);
```

Encodes a packet into a stack buffer of `DXL2_MAX_PACKET_SIZE` bytes and passes it to the write function.

Return	Meaning
`0`	Packet written
`DXL2_PARSE_ERROR`	`NULL` link, no write function, invalid parameters, or the packet does not fit
other negative value	Returned by the write function

`dxl2_link_send_status()`

```
int dxl2_link_send_status(struct dxl2_link* link, uint8_t id, uint8_t error,
                          const uint8_t* data, uint16_t data_length);
```

Sends a Status Packet with instruction `0x55`: the error byte followed by `data`.

Parameter	Description
`id`	ID of the answering device
`error`	`DXL2_ERR_*`, optionally ORed with `DXL2_ERR_ALERT_BIT`
`data`	Status data; may be `NULL` when `data_length` is 0
`data_length`	Number of data bytes, at most `DXL2_MAX_PARAMETER_SIZE - 1`

Returns the same values as `dxl2_link_send()`. Invalid `data` or `data_length` returns `DXL2_PARSE_ERROR` before anything is written.

`dxl2_link_rx_packets()`, `dxl2_link_invalid_packets()`

```
uint32_t dxl2_link_rx_packets(const struct dxl2_link* link);
uint32_t dxl2_link_invalid_packets(const struct dxl2_link* link);
```

Return the number of valid and of rejected packets since `dxl2_link_init()`, or `0` for a `NULL` link. The counters wrap around at 2^32.

11.6 Return codes summary

Value	Meaning
`0`	Success (`dxl2_parser_feed()`: need more bytes)
`1`	`dxl2_parser_feed()` only: a complete valid packet is available
`DXL2_PARSE_ERROR` (`-1`)	Invalid argument, malformed packet, CRC mismatch, or insufficient capacity
other negative	`dxl2_link_send*()` only: value returned by the write function

CHAPTER 12

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 `dxl2.h`. Tests, examples and tools are not in scope.

12.1 Checking

Check	Where	Gate
Cppcheck MISRA C:2012 addon	`misra-analyze` target, CI job *MISRA C:2012 Analysis*	Fails on any violation
Cppcheck, all checks	`cppcheck-analyze` target, CI job *Cppcheck Analysis*	Report
CodeQL `cpp-queries`	`codeql-analyze` target, CI job *CodeQL Analysis*	Report
Compiler	`-std=c11 -Wall -Wextra -Wpedantic -Wconversion`	No warnings
Manual review	Pull request review, see `.github/.copilot-instructions.md`	Reviewer

`misra-analyze` fails when the report contains a violation. Set `-DDXL2_MISRA_FAIL_ON_VIOLATION=OFF` to get the report without failing the build. A violation that cannot be fixed must be added to the deviation register below before the pull request is merged.

The Cppcheck addon checks most decidable rules, but not the directives or rules that need knowledge of the whole program or the intent of the code. Those are covered by the manual review notes below.

12.2 Current status

Item	Result
Cppcheck MISRA addon (2.13 in CI, 2.16)	0 violations
Unit test coverage of `src/`	100 % lines, 100 % branches
Deviations	3, all advisory, listed below

12.3 Manual review notes

Guideline	How the library complies
Dir 4.1 (run-time failures)	Every array index is bounded by a length check or by the `DXL2_MAX_PACKET_SIZE` invariant described in `dxl2_decode_packet()`. `inc/dxl2_config.h` rejects a `DXL2_MAX_PACKET_SIZE` outside `10..65542` at compile time.

Guideline	How the library complies
Dir 4.11 (validity of arguments)	Every public function checks its pointer arguments and lengths and returns `DXL2_PARSE_ERROR` (or `0` for `dxl2_crc16()` and the counters). The inline little-endian helpers in `dxl2_defs.h` do not check: they take a pointer to 2 or 4 readable bytes, like `memcpy()`.
Dir 4.12, Rule 21.3 (dynamic memory)	No heap. All state is in caller-owned structures.
Rule 15.5 (single point of exit)	Every function has one `return` at the end.
Rule 17.2 (recursion)	No recursion; the stack depth is fixed, see Configuration.
Rule 21.6 (standard I/O)	No `<stdio.h>`. The library includes only `<stddef.h>`, `<stdint.h>`, `<stdbool.h>` and `<string.h>`.

12.4 Deviation register

DEV-01: `int` as the status type (Dir 4.6, advisory)

- **Location:** return type of `dxl2_parser_feed()`, `dxl2_encode()`, `dxl2_link_init()`, `dxl2_link_send()`, `dxl2_link_send_status()` and of the `dxl2_write_fn` callback.
- **Rationale:** the value is a status code (`-1`, `0`, `1`, or the code the write callback returned), not a quantity whose size matters. `int` lets an integrator pass an existing driver `write()` function as `dxl2_write_fn` without a wrapper. `int` is at least 16 bits on every conforming compiler, which holds every value the library returns.
- **Risk control:** the library never does arithmetic on these values; it only compares them with `0` or `DXL2_PARSE_ERROR`.

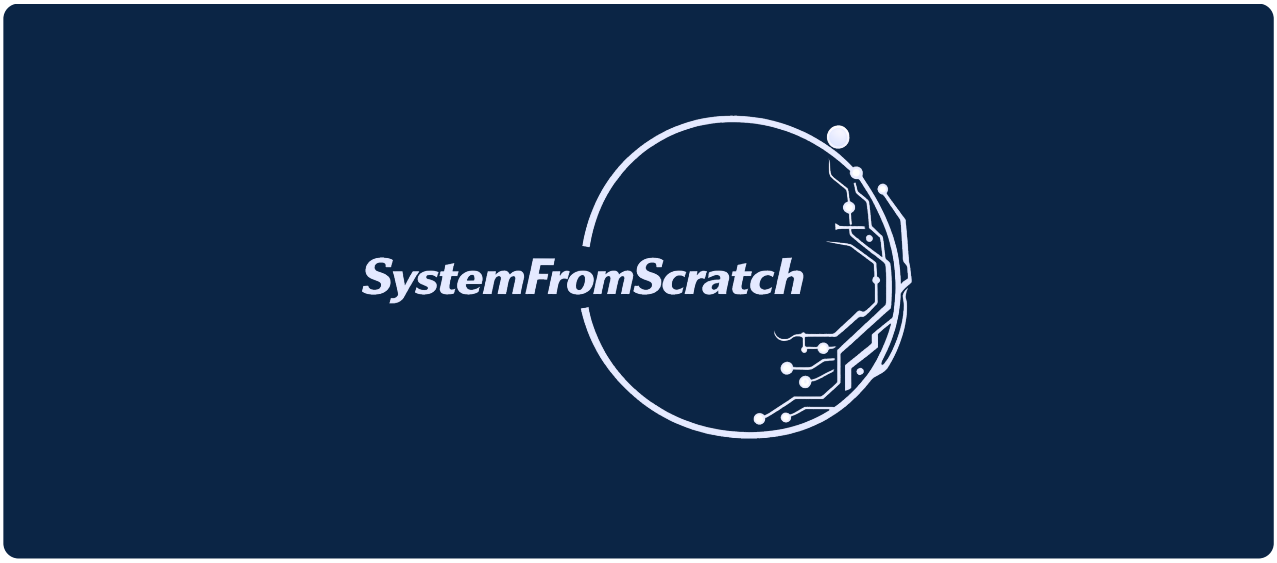
DEV-02: macros not used inside the library (Rule 2.5, advisory)

- **Location:** `DXL2_INST_*` (except `DXL2_INST_STATUS`), `DXL2_ERR_*`, `DXL2_MAX_ID`, `DXL2_BROADCAST_ID` in `inc/dxl2_defs.h`; `DXL2_VERSION_*` in `dxl2.h`.
- **Rationale:** these are the public names of the protocol constants. They are used by applications, not by the codec, which handles every instruction the same way.

DEV-03: external functions referenced in one translation unit (Rule 8.7, advisory)

- **Location:** the public API functions that the library itself does not call, for example `dxl2_link_rx_packets()` and `dxl2_link_invalid_packets()`.
- **Rationale:** they are the library interface and are called by applications. Giving them internal linkage would remove them from the API.

Author & Company



DXL2-C is designed, written and maintained by **Anton Vynohradov** and published by **SystemFromScratch®** — secure embedded systems and compliance-ready engineering, from concept to certification.

Author

Anton Vynohradov

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 DXL2-C, Anton wrote the library, its unit and integration tests, the examples, the DXL2 Studio tool and this documentation.

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
Embedded software development	Bare-metal, RTOS and Linux firmware for performance, reliability and long-term maintainability
BSP & device drivers	Board support packages, bootloaders and low-level drivers for MCUs, SoCs, FPGAs and custom hardware
RTOS & Embedded Linux	Zephyr, FreeRTOS and Embedded Linux (Yocto, Buildroot): porting, configuration, integration, reproducible builds
Hardware bring-up & integration	Schematic review, first boot, validation and production support, including SoC FPGA and motor-control platforms
Cybersecurity & CRA readiness	Secure boot, firmware integrity, threat modelling and a secure development lifecycle for the EU Cyber Resilience Act
Compliance & consulting	MISRA C/C++, static analysis, code and architecture reviews, compliance roadmaps

DXL2-C 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	avynohradov@systemfromscratch.com
GitHub	github.com/AntonVynohradov
GitHub organization	github.com/SystemFromScratch
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/DXL2-C/issues). Please include the library version, your compiler and target, and — for communication problems — a log exported from DXL2 Studio (p. 42).
- **Contributions:** see Contributing to the library (p. 29).
- **Commercial support, integration and custom development:** contact SystemFromScratch® 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 DXL2-C; the Apache License 2.0 grants no rights to use them, and they may not be used to endorse or promote products derived from DXL2-C without prior written permission.

DXL2-C implements the protocol as published by ROBOTIS in the DYNAMIXEL Protocol 2.0 specification (docs.robotis.com/docs/dxl/protocol/protocol2/). The reference packets used in the tests come from that specification. DYNAMIXEL and ROBOTIS are trademarks of ROBOTIS Co., Ltd.; DXL2-C is an independent implementation and is not affiliated with or endorsed by ROBOTIS.

APPENDIX A

Changelog

All notable changes to DXL2-C are listed here. The format follows Keep a Changelog (keep-achangelog.com/en/1.1.0/) and the project uses Semantic Versioning (semver.org/): the public API in `dxl2.h` and `inc/` stays compatible within a major version.

A.1 Unreleased

A.2 1.0.0 - 2026-10-25

First public release.

Added

- DYNAMIXEL Protocol 2.0 codec: CRC-16 (polynomial `0x8005`), byte stuffing and unstuffing, incremental byte-stream parser that resynchronizes on the next header, encoder with output-capacity checks.
- Transport-independent link (`dxl2_link_*`): bytes in, packets out, through a user write callback and packet handler; valid and invalid packet counters.
- All instruction and status error values of the ROBOTIS specification, and little-endian helpers (`dxl2_get_u16/u32`, `dxl2_put_u16/u32`).
- Build-time configuration `DXL2_MAX_PACKET_SIZE` (default 256 bytes), checked at compile time.
- Examples: bare metal (UART interrupt, ring buffer, main loop; runs in CI on a simulated board), host serial port PING tool (Linux, macOS, Windows), Zephyr application.
- DXL2 Studio GUI (`tools/dxl2_gui`) for simulation and serial-port debugging.
- Documentation website (MkDocs Material): introduction, getting started, architecture, usage, extension guide, wire format with diagrams, integration guide, examples, DXL2 Studio, API reference (hand-written and Doxygen), configuration with measured stack usage, MISRA C: 2012 compliance, author and license pages.
- Printable A4 PDF user manual with cover, table of contents and page numbers (`tools/docs/build_manual.py`), published with the website and every release.

Quality

- MISRA C:2012: 0 violations reported by the Cppcheck addon; three advisory deviations recorded in `docs/misra.md`. CI fails on any violation.
- Unit tests (GoogleTest): 100 % line and branch coverage of `src/`, enforced in CI.
- Integration tests (pytest): Python codec and a C/Python cross-check on random packets; ROBOTIS reference packets as regression vectors.
- Cppcheck and CodeQL static analysis in CI.

APPENDIX B

License

DXL2-C 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 (`dxl2.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 DXL2-C 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 (DXL2-C 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 SystemFromScratch® to endorse your product.

B.4 Trademarks

DYNAMIXEL and ROBOTIS are trademarks of ROBOTIS Co., Ltd. They are used in this documentation only to identify the protocol that DXL2-C implements. DXL2-C is an independent implementation and is not affiliated with or endorsed by ROBOTIS.

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.
```