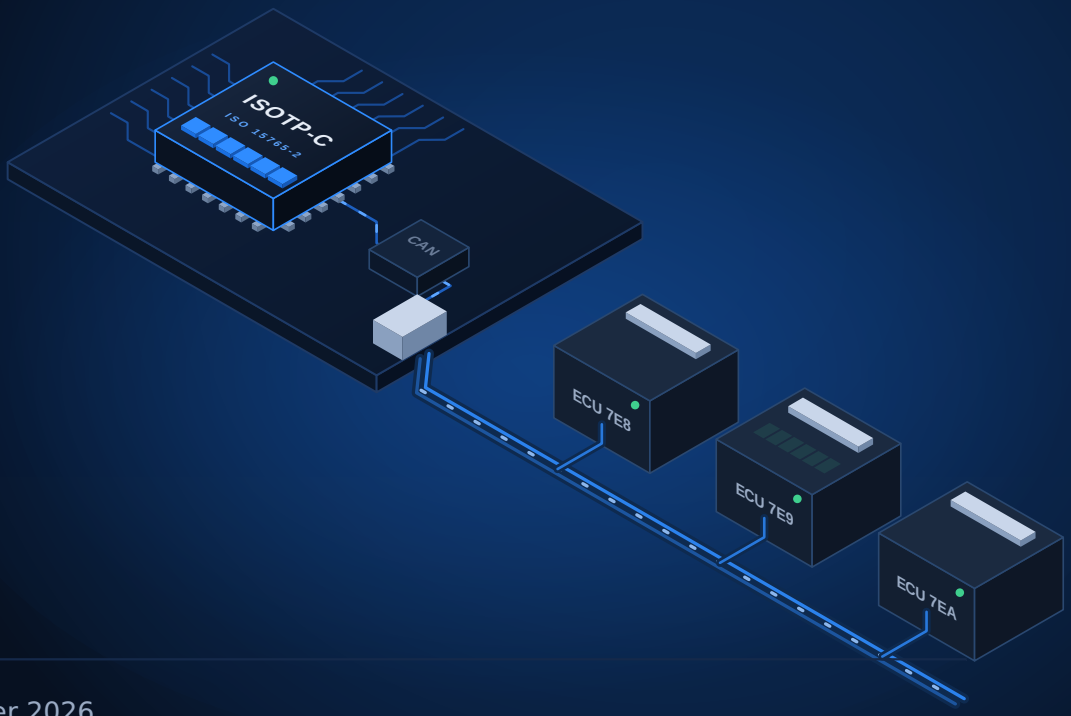
SystemFromScratch

OPEN-SOURCE EMBEDDED LIBRARY

# ISOTP-C

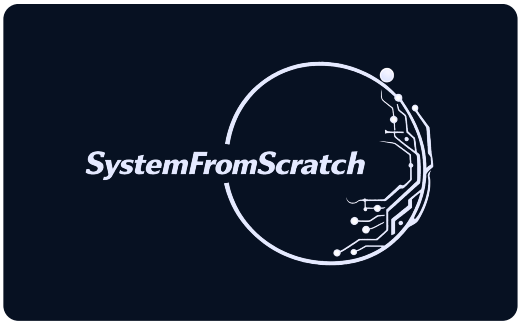
ISO 15765-2 (ISO-TP) Library for Embedded Systems

USER MANUAL




Version 2.0.1 · October 2026
Anton Vynohradov
systemfromscratch.com

# Imprint




**ISOTP-C User Manual**
Library version 2.0.1
Edition: October 2026

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

**Online**
Documentation: https://antonvynohradov.github.io/isotp-c/
Source code: https://github.com/AntonVynohradov/isotp-c

**Copyright © 2026 Anton Vynohradov.**

The ISOTP-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 ISOTP-C without prior written permission.

ISO 15765-2 is a standard of the International Organization for Standardization (ISO); the name is used only to identify the protocol that ISOTP-C implements. ISOTP-C is an independent implementation and is not affiliated with or endorsed by ISO. 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 ..... 8

**2 Getting Started ..... 9**

2.1 Get the library ..... 9
2.2 Add it to your project ..... 9
2.3 Prerequisites for development ..... 9
2.4 Build with presets ..... 9
2.5 Unit tests and coverage ..... 10
2.6 Integration tests ..... 10
2.7 Static analysis ..... 10
2.8 Formatting ..... 10
2.9 Versioning and releases ..... 11
2.10 Documentation ..... 11

**3 How It Works ..... 12**

3.1 Layers ..... 12
3.2 Data flow ..... 13
3.3 Segmentation and flow control ..... 13
3.4 The receiver ..... 14
3.5 The transmitter ..... 15
3.6 Time ..... 15
3.7 Memory model ..... 16
3.8 Error handling ..... 16
3.9 Threading and reentrancy ..... 16
3.10 Design decisions ..... 17

**4 Using the Library ..... 18**

4.1 Setting up a link ..... 18
4.2 Writing an ECU ..... 18
4.3 Writing a tester ..... 19
4.4 Using the codec without a link ..... 20
4.5 Return values ..... 20
4.6 Diagnostics ..... 21
4.7 Common mistakes ..... 21

**5 Extending the Library ..... 22**

5.1 Where to extend ..... 22
5.2 Adding a CAN driver ..... 22
5.3 Talking to several ECUs ..... 23
5.4 Adding an addressing mode or CAN FD ..... 23
5.5 Contributing to the library ..... 23

**6 Configuration ..... 25**
6.1 Run-time configuration ..... 25
6.2 Memory footprint ..... 26
6.3 Stack usage ..... 26
6.4 CMake options ..... 27

**7 ISO 15765-2 (ISO-TP) ..... 28**
7.1 The bus ..... 28
7.2 Frames ..... 29
7.3 Flow status, block size and separation time ..... 29
7.4 Timers and N_Result ..... 30
7.5 Receiver behavior ..... 30
7.6 Scope of this implementation ..... 31
7.7 Reference frames used in the tests ..... 31

**8 Integration Guide ..... 32**
8.1 The time source ..... 32
8.2 RTOS: interrupt, queue and CAN task ..... 32
8.3 Adding the library to a CMake build ..... 32
8.4 Bare metal and host ..... 33
8.5 Notes from hardware bring-up ..... 33

**9 Examples ..... 34**
9.1 Bare metal ..... 34
9.2 Linux SocketCAN ..... 35
9.3 Zephyr ..... 35

**10 API Reference ..... 37**
10.1 Version — isotp.h ..... 37
10.2 Configuration — isotp_config.h ..... 37
10.3 Definitions — isotp_defs.h ..... 38
10.4 Codec — isotp_protocol.h ..... 38
10.5 Link — isotp_link.h ..... 42
10.6 Return codes summary ..... 46

**11 MISRA C:2012 compliance ..... 47**
11.1 Checking ..... 47
11.2 Current status ..... 47
11.3 Manual review notes ..... 48
11.4 Deviation register ..... 48

**Author & Company ..... 50**
Author ..... 50
SystemFromScratch® ..... 51
Contact ..... 51
Support and feedback ..... 52
Trademarks and acknowledgements ..... 52

**Appendix A Changelog ..... 53**
A.1 Unreleased ..... 53
A.2 2.0.1 - 2026-10-26 ..... 53
A.3 2.0.0 - 2026-10-26 ..... 53

A.4 1.6.0 and earlier ..... 54

**Appendix B License ..... 55**

B.1 What the license allows ..... 55
B.2 What it requires ..... 55
B.3 What it does not give ..... 55
B.4 Trademarks ..... 56
B.5 Full license text ..... 56

CHAPTER 1

# Introduction

ISOTP-C is a C11 library that implements the transport protocol of ISO 15765-2 (ISO-TP) (iso.org/standard/84211.html) for classical CAN, for embedded systems that must be predictable and reviewable. It carries messages of up to 4095 bytes — or longer, with the 32-bit length of ISO 15765-2:2016 — over CAN frames of eight bytes, which is what diagnostics (UDS, ISO 14229), OBD and CAN bootloaders need.

ISOTP-C started as a fork of the MIT-licensed isotp-c by Li Shen and Simon Cahill. Version 2 is a rewrite with a new API; see the changelog (p. 53) for the differences.

## 1.1 What the library does

- **Encodes and decodes frames.** Single, First, Consecutive and Flow Control frames, including the 32-bit FF_DL escape, optional padding, and the STmin encoding with its reserved values.
- **Segments and reassembles messages.** A link sends a long message as a First Frame followed by Consecutive Frames, at the block size and separation time the receiver asks for, and reassembles incoming messages while checking every sequence number.
- **Detects protocol errors.** Timeouts N_Bs and N_Cr, wrong sequence numbers, unexpected frames, too many Wait frames, overflow and unknown flow status are reported as the N_Result codes of ISO 15765-2.
- **Connects the protocol to any CAN driver.** One write callback; received frames and the current time are function arguments.

## 1.2 What the library does not do

| Concern | Why it is not in the library |
|---|---|
| CAN drivers, acceptance filters | Hardware- and OS-specific; you pass frames in and provide a write function. |
| Routing CAN IDs to links | Depends on your addressing plan; one link serves one pair of CAN IDs. |
| Clocks and timers | The library never reads a clock: every call takes `now_us`. |
| Threads, queues, interrupts, locking | Every system has its own; see the integration guide. |
| Diagnostic services (UDS, OBD) | Application layer; the examples show a small UDS-style ECU. |
| Extended/mixed addressing, CAN FD | Not implemented yet; see Extending the Library. |

## 1.3 Who it is for

- **ECU firmware.** A diagnostic server or bootloader on a microcontroller, bare metal or under an RTOS.

- **Testers and gateways.** A diagnostic client on a PC (SocketCAN) or a gateway that routes ISO-TP between buses.
- **Test benches.** Simulated nodes that need a protocol implementation with deterministic timing (the time is an argument, so tests control it).

## 1.4 Key properties

| Property | Value |
|---|---|
| Language | C11, no warnings with `-Wall -Wextra -Wpedantic -Wconversion` |
| Dependencies | `<stddef.h>`, `<stdint.h>`, `<stdbool.h>`, `<string.h>` only |
| Memory | No heap; caller-owned `struct isotp_link`; buffers sized by `ISOTP_TX_BUFFER_SIZE` / `ISOTP_RX_BUFFER_SIZE` |
| RAM per link | 104 bytes + both buffers (8 296 bytes with the default 4095-byte buffers, Cortex-M4) |
| Stack | 120 bytes worst case on Cortex-M4 `-Os` (details) |
| Code size | 1 784 bytes of flash on Cortex-M4 `-Os` (codec 516, link 1 268) |
| Coding standard | MISRA C:2012, 0 Cppcheck addon violations, 3 documented deviations |
| Tests | 82 unit tests, 100 % line and branch coverage; 62 integration tests against a Python twin |
| Thread safety | None built in, by design; see Threading and reentrancy |
| License | MIT License |

## 1.5 Terminology

**Link**
: One ISO-TP connection to one peer: the CAN ID this node sends with and the frames the application routes to it. `struct isotp_link` holds its state.

**Message (N_SDU)**
: The payload the application sends or receives, 1 to `ISOTP_TX/RX_BUFFER_SIZE` bytes.

**SF, FF, CF, FC**
: Single Frame, First Frame, Consecutive Frame and Flow Control frame — the four frame types (N_PDU) of ISO 15765-2.

**FF_DL, SF_DL**
: The message length announced in a First Frame or Single Frame.

**SN**
: Sequence number of a Consecutive Frame, 1, 2, … 15, 0, 1, …

**BS, STmin**
: Block size and minimum separation time: how many Consecutive Frames the sender may send before the next Flow Control frame, and how long it waits between them.

**N_Bs, N_Cr**

Sender timeout waiting for Flow Control; receiver timeout waiting for the next Consecutive Frame.

**N_WFTmax**

How many Flow Control frames with the status Wait in a row the sender accepts.

**Physical and functional addressing**

A request to one node, or a broadcast request (only as a Single Frame).

## 1.6 Conventions in this documentation

- Hexadecimal bytes without prefix inside frames (`10 14 00 01`), with `0x` in text and code.
- The 32-bit FF_DL is big-endian, as on the wire.
- Times are in microseconds (`_us`) unless a unit is given.
- `ISOTP_TX_BUFFER_SIZE` and `ISOTP_RX_BUFFER_SIZE` are shown with their default, 4095.

CHAPTER 2

# Getting Started

This chapter shows how to get ISOTP-C, add it to your project, and build and test it. The library itself is two C files and five headers; everything else in the repository is tests, examples, tooling and documentation.

## 2.1 Get the library

```
git clone https://github.com/AntonVynohradov/isotp-c.git
```

Each release (github.com/AntonVynohradov/isotp-c/releases) provides the source code, a prebuilt static library for Linux x86-64 with its headers, the MISRA report and the PDF user manual.

## 2.2 Add it to your project

**CMake.**

```
include(path/to/isotp-c/isotp.cmake)
target_link_libraries(app PRIVATE isotp)
```

**Any other build system.** Compile `src/*.c` and add the repository root and `inc/` to the include path.

**Prebuilt package.** Link `lib/libisotp.a` and add `include/` to the include path.

## 2.3 Prerequisites for development

- C11 compiler, CMake 3.20+
- Optional: GoogleTest and gcovr, Python 3 and pytest, Cppcheck, clang-format, MkDocs (requirements-dev.txt), Doxygen, WeasyPrint with Pango

## 2.4 Build with presets

| Preset | Purpose |
|---|---|
| `ISOTP[Debug]` | Static library with debug information |
| `ISOTP[Release]` | Optimized static library |
| `ISOTP[UnitTests]` | GoogleTest executable with coverage; target `coverage` |
| `ISOTP[IntegrationTests]` | pytest suite; target `integration_tests` |
| `ISOTP[Examples]` | Host examples and the bare-metal simulation test |

```
cmake --preset "ISOTP[Release]"
cmake --build "build/ISOTP[Release]"
```

## 2.5 Unit tests and coverage

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

82 GoogleTest tests in `tests/unit/` cover the codec (every frame type, every rejection, every capacity boundary), STmin, and the link (transmitter, receiver, timeouts, flow control, errors, callbacks). The unit-test build uses 5000-byte buffers so that the 32-bit FF_DL paths are reachable. The `coverage` target runs them and fails below `ISOTP_COVERAGE_MIN` (100 %) line or branch coverage of `src/`; the HTML report is in `build/ISOTP[UnitTests]/Coverage/`.

## 2.6 Integration tests

```
cmake --preset "ISOTP[IntegrationTests]"
cmake --build "build/ISOTP[IntegrationTests]" --target integration_tests
# or directly:
python -m pytest tests/integration -v
```

`tests/integration/isotp/protocol.py` is a second implementation of the frame codec in Python, written from the standard and using only the standard library (the *Python twin*). The suite

- checks the reference frames and the frames a receiver must ignore on the twin,
- compiles `src/isotp_protocol.c` with `gcc` and compares it with the twin on 300 random messages per encoder, 2 000 random frames for the decoder and every STmin value,
- compiles the whole library with `tests/integration/isotp_harness.c` and lets a peer built on the twin exchange messages with the C link on a simulated clock: message sizes from 1 to 4999 bytes, block sizes 1 to 255, four STmin values, timeouts, Wait frames, overflow, wrong sequence numbers, padding, functional addressing, a busy driver and line noise.

The tests that need `gcc` are skipped (and reported as skipped) when it is missing.

## 2.7 Static analysis

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

## 2.8 Formatting

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

## 2.9 Versioning and releases

The version comes from the latest `vX.Y.Z` Git tag (`cmake/version.cmake`) and is passed to the compiler as `ISOTP_VERSION_MAJOR/MINOR/PATCH` and `ISOTP_VERSION`; `isotp.h` has fallbacks for builds without CMake. The project follows Semantic Versioning and keeps a changelog (p. 53).

A release is made from `main` by pushing a tag `vX.Y.Z`. The *Release* workflow checks that the tag matches `isotp.h`, `cmake/version.cmake` and `CHANGELOG.md`, runs every quality gate, and publishes `isotp-c-X.Y.Z-linux-x86_64.tar.gz` (static library, headers, license, readme, changelog), the MISRA report and the PDF manual as a GitHub Release.

## 2.10 Documentation

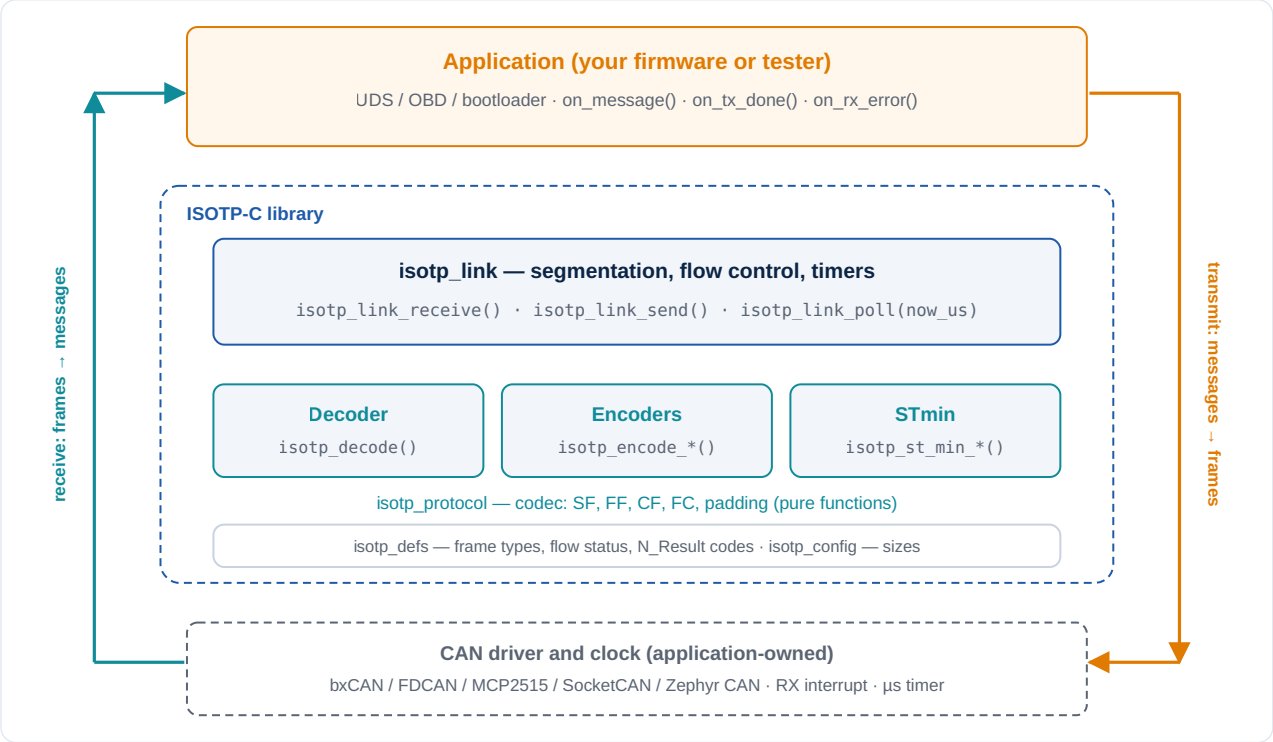
| Task | Command | VS Code task |
|---|---|---|
| Live preview | `mkdocs serve` | *Docs: Serve on localhost* |
| Build the website | `mkdocs build` | *Docs: Build site* |
| Build the PDF manual | `python tools/docs/build_manual.py` | *Docs: Build PDF manual* |
| Regenerate diagrams | `python tools/docs/diagrams.py` | *Docs: Generate diagrams* |
| Compile the C snippets of these pages | `python tools/docs/check_snippets.py` | *Docs: Check C snippets* |

CHAPTER 3

# How It Works

This chapter explains how ISOTP-C is built: its layers, how frames and messages flow through it, the transmitter and receiver state machines with their timers, how errors are reported, and why it is designed this way.

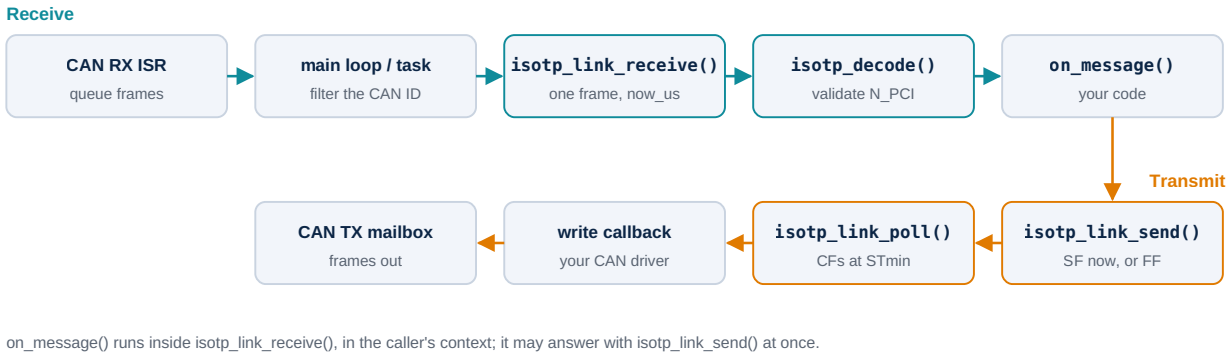
## 3.1 Layers




| Layer | Header | Responsibility |
|---|---|---|
| Definitions | `isotp_defs.h` | Frame types, flow status values, frame layout limits, return and N_Result codes, big-endian helpers. No state. |
| Configuration | `isotp_config.h` | Buffer sizes and protocol defaults, range-checked at compile time. |
| Codec | `isotp_protocol.h` | `isotp_decode()` validates one CAN frame and describes it in a `struct isotp_pdu`; `isotp_encode_*()` write one frame; `isotp_frame_pad()`; STmin conversion. Pure functions over caller buffers. |
| Link | `isotp_link.h` | One connection to one peer: copies outgoing messages, sends FF and CFs under the receiver's flow control, reassembles incoming messages, answers with Flow Control, runs the N_Bs and N_Cr timers, reports results. |

Each layer depends only on the ones below it. The codec can be used without the link, for example in a bus monitor that only decodes frames.

## 3.2 Data flow

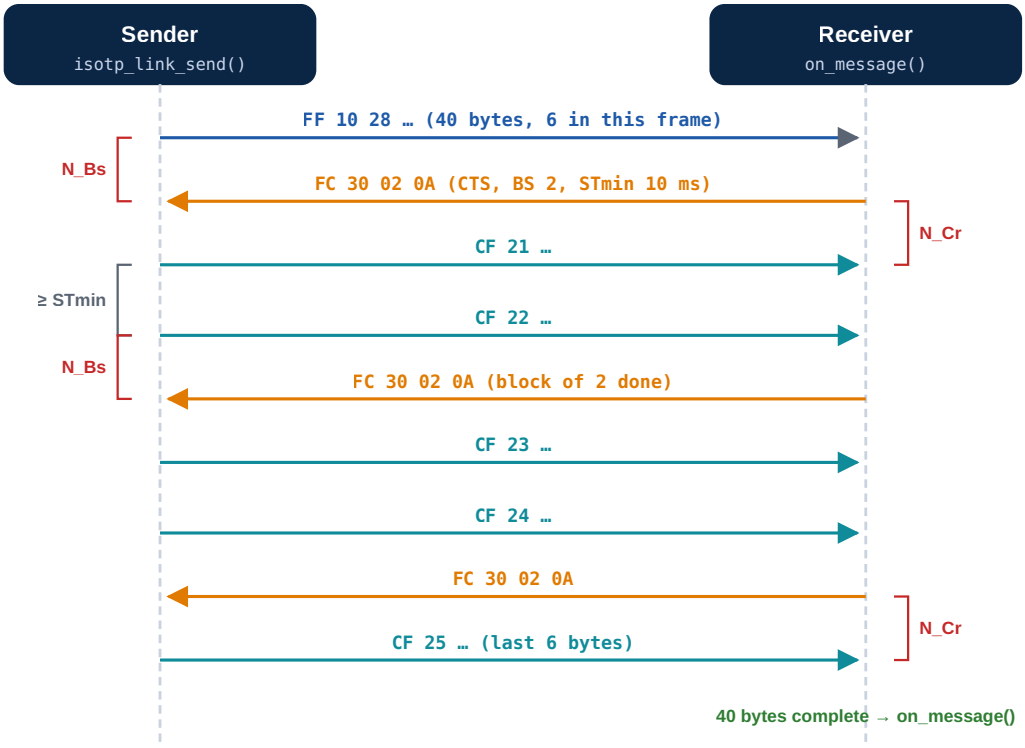



on_message() runs inside isotp_link_receive(), in the caller's context; it may answer with isotp_link_send() at once.

**Receive path.** Your CAN interrupt stores the frame; your task or main loop takes it out, checks that its CAN ID belongs to this link and calls `isotp_link_receive()` with the current time. The link decodes the frame and acts on it. A Single Frame is delivered to `on_message()` straight from the frame data; the frames of a segmented message are copied into the link's receive buffer, and `on_message()` is called when the last byte has arrived. The link writes its Flow Control frames itself, through your write function.

**Transmit path.** `isotp_link_send()` sends a message of up to 7 bytes at once as a Single Frame. A longer message is copied into the link's transmit buffer and its First Frame is written; then the link waits for the receiver's Flow Control frame. Each call of `isotp_link_poll()` sends at most one Consecutive Frame, when the separation time has passed and the block is not finished. When the last frame has been written, the link calls `on_tx_done(ISOTP_N_OK)`.

## 3.3 Segmentation and flow control

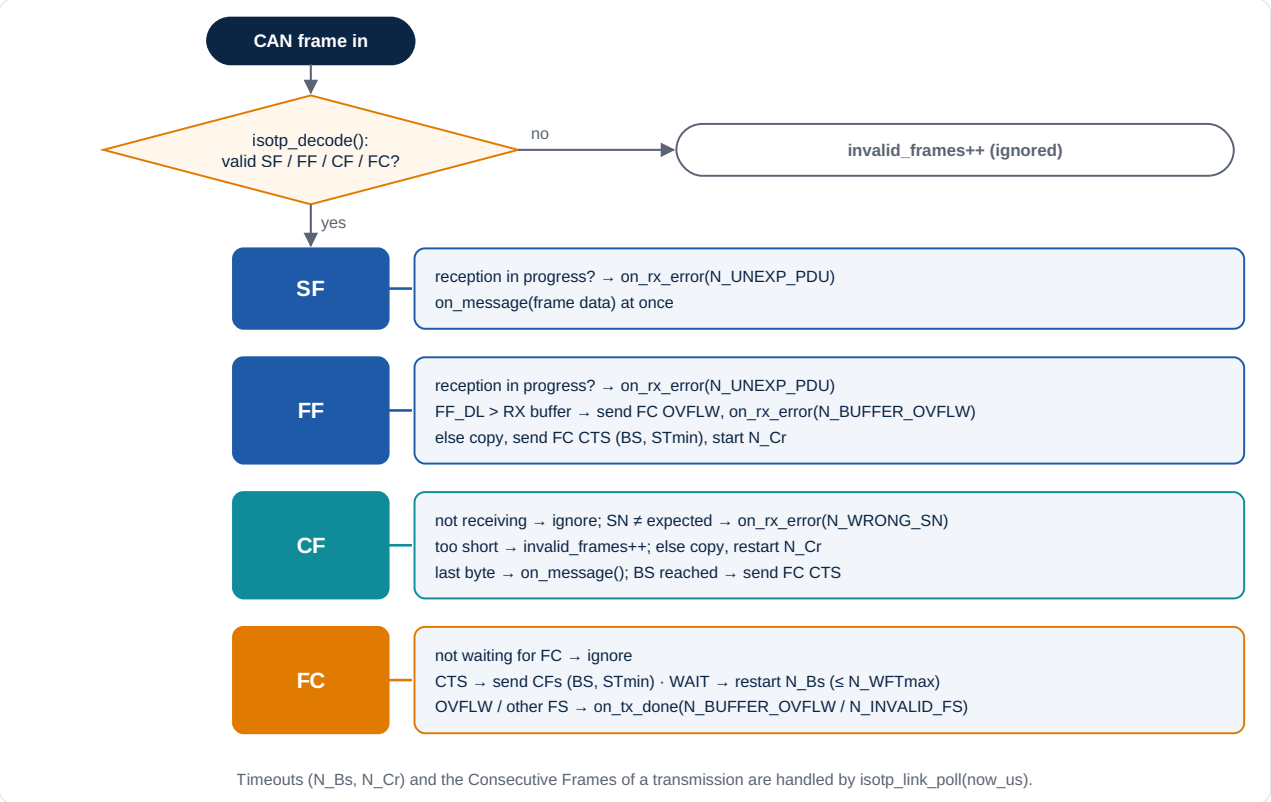


A message that does not fit a Single Frame is sent as a First Frame, which announces the total length and carries the first 6 bytes, followed by Consecutive Frames of up to 7 bytes with a 4-bit sequence number that starts at 1 and wraps from 15 to 0. After the First Frame and after every block of BS Consecutive Frames, the sender waits for a Flow Control frame from the receiver:

| Flow status | Sender |
|---|---|
| CTS (continue to send) | Sends the next block of BS frames (BS = 0: all remaining frames) with at least STmin between two frames |
| WAIT | Keeps waiting and restarts N_Bs; more than N_WFTmax Wait frames in a row end the transmission with `ISOTP_N_WFT_OVRN` |
| OVFLW (overflow) | Ends the transmission with `ISOTP_N_BUFFER_OVFLW` |
| other value | Ends the transmission with `ISOTP_N_INVALID_FS` |

The receiver answers a First Frame with CTS and its own BS and STmin (`isotp_link_config::block_size`, `::st_min_us`), and repeats the CTS after every BS Consecutive Frames. It answers with OVFLW when the announced length does not fit `ISOTP_RX_BUFFER_SIZE`. The library never sends WAIT: its receive buffer is always ready.

## 3.4 The receiver




Timeouts (N_Bs, N_Cr) and the Consecutive Frames of a transmission are handled by isotp_link_poll(now_us).

1. **Decode.** `isotp_decode()` rejects what ISO 15765-2 tells a receiver to ignore: a reserved frame type, SF_DL 0 or longer than the frame, a First Frame shorter than 8 bytes or announcing a message that would fit a Single Frame, an escape-sequence First Frame with a length up to 4095, a Consecutive Frame without data, a Flow Control frame shorter than 3 bytes. Rejected frames are counted (`isotp_link_invalid_frames()`).

2. **Single or First Frame.** If a reception is in progress, it ends with `on_rx_error(ISOTP_N_UNEXP_PDU)` and the new frame is processed: a Single Frame is delivered at once, a First Frame starts a new reception.
3. **Consecutive Frame.** Ignored when no reception is in progress. A wrong sequence number ends the reception with `ISOTP_N_WRONG_SN`. A frame that carries fewer bytes than are still expected is counted as invalid and ignored. Otherwise its bytes are copied, N_Cr restarts, and the message is delivered when it is complete.
4. **Flow Control frame.** Handled by the transmitter when it waits for one (see above); ignored otherwise.
5. **Timeout.** `isotp_link_poll()` ends a reception with `ISOTP_N_TIMEOUT_CR` when no Consecutive Frame arrived within N_Cr.

> **Recovery after errors**
>
> After any error the receiver is idle and accepts the next Single or First Frame. ISO-TP has no retransmission: the sender learns about a receiver error only by its own timeout (N_Bs) or by an Overflow, and the application (for example the UDS client) repeats the request.

## 3.5 The transmitter

| State | Entered when | Leaves on |
|---|---|---|
| `ISOTP_TX_IDLE` | init, end of a transmission | `isotp_link_send()` of more than 7 bytes |
| `ISOTP_TX_WAIT_FC` | First Frame written; block of BS frames written | FC CTS → `SEND_CF`; FC WAIT (≤ N_WFTmax) → stays; FC OVFLW, invalid FS, too many WAIT, N_Bs expired → `IDLE` with an error |
| `ISOTP_TX_SEND_CF` | FC CTS received | last frame written → `IDLE` (`ISOTP_N_OK`); block done → `WAIT_FC`; write error → `IDLE` (`ISOTP_N_ERROR`) |

When the write function returns `ISOTP_BUSY` for a Consecutive Frame, the link tries the same frame again on the next poll. For a Single or First Frame, `isotp_link_send()` returns the writer's result and nothing is started.

## 3.6 Time

The library never reads a clock. Every call that can start or check a timer takes `now_us`, a free-running microsecond counter that may wrap at 2^32. Deadlines are stored as `now_us + timeout` and compared modulo 2^32, which is correct as long as a timeout is shorter than 2^31 µs (about 35 minutes); `isotp_link_init()` enforces that. The resolution you get is the resolution of your counter and of your poll interval: to send at an STmin of 1 ms, poll at least every millisecond.

## 3.7 Memory model

| Structure | Contents | Size with the default configuration (Cortex-M4) |
|---|---|---|
| `struct isotp_link` | configuration copy, both message buffers, both state machines, counters | 8 296 bytes |
| `struct isotp_link_config` | CAN ID, callbacks, flow-control and timing parameters | 48 bytes |
| `struct isotp_pdu` | one decoded frame (on the stack inside the link) | 16 bytes |
| Frame buffer | 8 bytes on the stack while one frame is written | 8 bytes |

There is no global state. Every link is independent; an application with three peers declares three links.

## 3.8 Error handling

| Situation | Result |
|---|---|
| A `NULL` pointer, zero length or a value out of range | `ISOTP_ERROR`; nothing is written or changed |
| `isotp_link_send()` while a segmented transmission is in progress | `ISOTP_BUSY` |
| The write function fails on a Single or First Frame | Its return value is returned by `isotp_link_send()` |
| The write function fails on a Consecutive Frame | `ISOTP_BUSY`: retried on the next poll; other: `on_tx_done(ISOTP_N_ERROR)` |
| The write function fails on a Flow Control frame | `on_rx_error(ISOTP_N_ERROR)`; the reception ends |
| Malformed frame | Ignored, `isotp_link_invalid_frames()` + 1 |
| Protocol error or timeout | `on_tx_done()` / `on_rx_error()` with the N_Result code; also readable with `isotp_link_tx_result()` / `isotp_link_rx_result()` |

## 3.9 Threading and reentrancy

- **Codec functions** are reentrant: they only touch their arguments.
- **A link** is not thread-safe. Call `isotp_link_receive()`, `isotp_link_poll()` and `isotp_link_send()` of one link from one context — a CAN task or the main loop, not the CAN interrupt — or protect all three with one mutex.
- **Callbacks** run in the caller's context, inside `isotp_link_receive()`, `isotp_link_poll()` or `isotp_link_send()`. They may call `isotp_link_send()` on the same link: the link updates its state before it calls them.
- **The message data** passed to `on_message()` is valid only during the call.

## 3.10 Design decisions

| Decision | Reason |
|---|---|
| Time is an argument (`now_us`) | No dependency on an OS or timer driver; tests control time exactly; works with any clock source. |
| One Consecutive Frame per poll | Bounded execution time per call; the CAN driver's queue is never flooded; STmin below the poll interval is still met because frames are never sent early. |
| The link copies outgoing messages | The caller may reuse its buffer at once; no lifetime rules for the payload. |
| Buffers sized at build time | No heap; the RAM per link is known at link time and in the map file. |
| STmin rounded up when encoded | The sender is never asked to go faster than the receiver needs. |
| Reserved STmin values read as 127 ms | Required by ISO 15765-2 for senders. |
| Single Frame delivered from the frame | No copy for the most common case (short requests). |
| Separate write and application contexts | The driver and the application layer are usually different modules. |
| Functional addressing only for Single Frames | ISO 15765-2 does not allow segmented messages to functional addresses. |

CHAPTER 4

# Using the Library

A link is used the same way in an ECU (diagnostic server) and in a tester (diagnostic client): the ECU waits for requests and answers them, the tester sends a request and waits for the answer. This chapter shows both, how to use the codec on its own, what every function returns, and the mistakes that are easy to make.

## 4.1 Setting up a link

Start from the defaults, set the CAN ID this node sends with and the callbacks, and initialize the link. The link keeps a copy of the configuration.

```c
static struct isotp_link link;

static int can_write(void* context, uint32_t can_id, const uint8_t* data, size_t length)
{
    (void) context;
    return board_can_write(can_id, data, length);
}

static void on_message(const uint8_t* data, uint32_t length, void* context)
{
    (void) data;
    (void) length;
    (void) context;
}

int setup(void)
{
    struct isotp_link_config config;

    isotp_link_config_default(&config);
    config.tx_id = 0x7E8U;          /* this ECU answers on 0x7E8 */
    config.write = can_write;
    config.on_message = on_message;
    config.block_size = 8U;         /* ask the tester for Flow Control every 8 frames */
    config.st_min_us = 1000U;       /* and 1 ms between its Consecutive Frames */
    config.padding = true;          /* most vehicles expect 8-byte frames */
    return isotp_link_init(&link, &config);
}
```

The link does not filter CAN IDs. Give it only the frames of its peer: in an ECU the physical request ID (0x7E0 above) and, if used, the functional ID (0x7DF); in a tester the response ID of the ECU.

## 4.2 Writing an ECU

An ECU calls `isotp_link_receive()` for every frame of its request IDs and `isotp_link_poll()` regularly, both with the current time. Requests arrive in `on_message()`.

### Answering

Answer from `on_message()` or later from the application, with `isotp_link_send()`. The answer is copied, so a local buffer is fine:

```c
static struct isotp_link link;

static void on_message(const uint8_t* data, uint32_t length, void* context)
{
    (void) context;
    if ((length >= 1U) && (data[0] == 0x3EU))   /* UDS TesterPresent */
    {
        const uint8_t answer[2] = {0x7EU, 0x00U};
        (void) isotp_link_send(&link, answer, sizeof(answer), board_micros());
    }
}
```

`isotp_link_send()` returns `ISOTP_BUSY` while a previous long answer is still being sent. A request–response protocol such as UDS never has two answers in flight; if your application can, keep the second answer and send it from `on_tx_done()`.

### Rules a well-behaved ECU follows

- Poll the link at least as often as the shortest separation time a tester may ask for — usually every millisecond.
- Answer within the time the application protocol allows (UDS: P2server, 50 ms by default); a long answer only has to *start* in time.
- Do not answer functional requests that do not need an answer: this is decided by the application layer, not by ISO-TP.
- Keep the receive buffer as large as the longest request you accept; longer requests are refused with Flow Control *Overflow*.

## 4.3 Writing a tester

A tester sends a request and waits for the answer or for an error:

```c
static struct isotp_link tester;
static volatile bool done;

static void on_answer(const uint8_t* data, uint32_t length, void* context)
{
    (void) data;
    (void) length;
    (void) context;
    done = true;
}

static void on_error(int n_result, void* context)
{
    (void) context;
    if (n_result != ISOTP_N_OK)
    {
        done = true;    /* timeout, overflow, wrong SN, ... */
    }
}

int request_vin(void)
{
    static const uint8_t request[3] = {0x22U, 0xF1U, 0x90U};   /* ReadDataByIdentifier */
    uint8_t frame[8];
    uint32_t can_id;
    const uint32_t start = board_micros();
    int result = isotp_link_send(&tester, request, sizeof(request), start);

    done = false;
    while ((result == ISOTP_OK) && !done && ((board_micros() - start) < 2000000U))
```

```
    {
        size_t length = board_can_read(&can_id, frame);
        if ((length > 0U) && (can_id == 0x7E8U))
        {
            isotp_link_receive(&tester, frame, length, board_micros());
        }
        isotp_link_poll(&tester, board_micros());
    }
    return result;
}
```

with `on_answer` as `on_message` and `on_error` as both `on_tx_done` and `on_rx_error` in the tester's configuration (`tx_id` 0x7E0). A functional request goes to all ECUs at once and must fit a Single Frame:

```
static struct isotp_link tester;

int tester_present_to_all(void)
{
    static const uint8_t request[2] = {0x3EU, 0x80U};   /* suppress positive response */
    return isotp_link_send_functional(&tester, 0x7DFU, request, sizeof(request));
}
```

## 4.4 Using the codec without a link

The codec is useful on its own, for example in a bus monitor or a gateway that only looks at frames:

```
void show(const uint8_t* data, size_t length)
{
    struct isotp_pdu pdu;

    if (isotp_decode(data, length, &pdu) == ISOTP_OK)
    {
        if (pdu.type == ISOTP_PCI_FIRST_FRAME)
        {
            /* a message of pdu.message_length bytes starts here */
        }
        else if (pdu.type == ISOTP_PCI_FLOW_CONTROL)
        {
            uint32_t separation_us = isotp_st_min_decode(pdu.st_min);
            (void) separation_us;
        }
        else
        {
            /* Single or Consecutive Frame */
        }
    }
}
```

## 4.5 Return values

| Function | Success | Failure |
|---|---|---|
| `isotp_decode()` | `ISOTP_OK` | `ISOTP_ERROR`: `NULL`, length 0 or > 8, malformed frame |
| `isotp_encode_*()`, `isotp_frame_pad()` | frame length (1..8) | `ISOTP_ERROR`: `NULL`, value out of range, buffer too small |

| Function | Success | Failure |
|---|---|---|
| `isotp_link_init()` | `ISOTP_OK` | `ISOTP_ERROR`: `NULL`, no writer, value out of range |
| `isotp_link_send()` | `ISOTP_OK` | `ISOTP_BUSY`, `ISOTP_ERROR`, or the writer's error |
| `isotp_link_send_functional()` | `ISOTP_OK` | `ISOTP_ERROR` (longer than 7 bytes), or the writer's error |
| `isotp_link_tx_result()`, `isotp_link_rx_result()` | `ISOTP_N_OK` | an `ISOTP_N_*` code |

## 4.6 Diagnostics

| Function | Tells you |
|---|---|
| `isotp_link_tx_busy()` | A segmented transmission is in progress |
| `isotp_link_tx_result()` | The outcome of the last transmission |
| `isotp_link_rx_result()` | The outcome of the last reception (also `ISOTP_N_UNEXP_PDU` when a new request interrupted one) |
| `isotp_link_invalid_frames()` | How many malformed frames were ignored — a high count points to wrong CAN ID routing or a peer with another addressing mode |

## 4.7 Common mistakes

| Symptom | Likely cause |
|---|---|
| Single Frames work, long messages time out at the peer (N_Bs) | Frames of the peer's ID are not passed to `isotp_link_receive()`, or the Flow Control is sent with the wrong `tx_id` |
| Long messages stall after the First Frame | `isotp_link_poll()` is not called, or called with a time that does not advance |
| Every transmission ends with `ISOTP_N_TIMEOUT_BS` at once | `now_us` is in milliseconds or wraps at less than 2^32 |
| The peer reports wrong sequence numbers | Frames are dropped by a full CAN RX queue; enlarge it or ask for a larger STmin |
| The peer ignores the frames | It expects 8-byte frames: set `padding` |
| `isotp_link_invalid_frames()` grows | Another node's traffic is routed to the link, or the peer uses extended addressing |
| `isotp_link_send()` returns `ISOTP_BUSY` forever | The previous transmission waits for Flow Control that never comes; it ends after N_Bs |

CHAPTER 5

# Extending the Library

Almost everything an application needs is built on top of ISOTP-C: a different CAN driver, more peers, diagnostic services, gateways. This chapter shows where each kind of extension belongs, what would need a library change, and how to contribute one.

## 5.1 Where to extend

| You want to… | Do it in | Library change needed? |
|---|---|---|
| Use another CAN driver or controller | your write function and receive loop | No |
| Talk to several ECUs | one `struct isotp_link` per peer, route frames by CAN ID | No |
| Use 29-bit CAN IDs (for example normal fixed addressing 0x18DAxxyy) | `tx_id` and your routing; the writer sets the IDE flag | No |
| Implement UDS, OBD or a bootloader | `on_message()` and the application | No |
| Route ISO-TP between two buses (gateway) | two links and the application | No |
| Change buffer sizes or defaults | compiler definitions, see Configuration | No |
| Extended or mixed addressing (address byte before N_PCI) | the codec and the link | Yes |
| CAN FD (frames of up to 64 bytes) | the codec and the link | Yes |
| Several messages in flight per link | not planned: ISO-TP is one message per direction per link | — |

## 5.2 Adding a CAN driver

The write function receives the CAN ID and 1 to 8 data bytes and must return:

| Return | Meaning for the link |
|---|---|
| `0` | The frame is queued for transmission |
| `ISOTP_BUSY` | No free transmit mailbox: a Consecutive Frame is retried on the next poll; `isotp_link_send()` returns it for a Single or First Frame |
| any other value | Driver error: a transmission ends with `ISOTP_N_ERROR`, a reception with `ISOTP_N_ERROR` when a Flow Control frame cannot be sent |

The data is only valid during the call: copy it into the mailbox or your queue. Do not block for long — the link calls the writer from `isotp_link_receive()` to send Flow Control, and a slow writer delays the frames behind it.

## 5.3 Talking to several ECUs

Declare one link per peer and dispatch on the CAN ID:

```
static struct isotp_link engine;
static struct isotp_link brake;

void on_can_frame(uint32_t can_id, const uint8_t* data, size_t length, uint32_t now_us)
{
    if (can_id == 0x7E8U)
    {
        isotp_link_receive(&engine, data, length, now_us);
    }
    else if (can_id == 0x7E9U)
    {
        isotp_link_receive(&brake, data, length, now_us);
    }
    else
    {
        /* not ISO-TP traffic for us */
    }
}
```

Poll every link from the same loop.

## 5.4 Adding an addressing mode or CAN FD

Both change the frame layout: extended and mixed addressing put an address byte in front of the N_PCI, CAN FD allows frames of up to 64 bytes with an escape sequence for the Single Frame length. A contribution would add the layout to `struct isotp_link_config` (so one build serves all modes), extend `isotp_decode()` and the encoders with the offset and the maximum frame length, extend the Python twin the same way, and add the reference frames of the standard to the unit and integration tests.

## 5.5 Contributing to the library

1. **Code.** C11, no new dependencies, no heap, the MISRA rules used in the code (one `return` per function, parenthesized expressions, `(void)` casts, no pointer arithmetic).
2. **Tests.** A unit test for every behavior; 100 % line and branch coverage; update the Python twin and the integration tests when the wire format changes.
3. **Analysis.** `misra-analyze` reports 0 violations, or the deviation is in the register (p. 47).
4. **Format.** `clang-format` reports no changes.
5. **Documentation.** The affected pages, the stack table (p. 26), the diagrams (`tools/docs/diagrams.py`) and `CHANGELOG.md` `[Unreleased]`.
6. **Compatibility.** The public API is stable within a major version.

The local definition of done before a pull request:

```
cmake --preset "ISOTP[Debug]" && cmake --build "build/ISOTP[Debug]"
cmake --preset "ISOTP[UnitTests]" && cmake --build "build/ISOTP[UnitTests]" --target cover
age
python -m pytest tests/integration
cmake --build "build/ISOTP[Debug]" --target misra-analyze
cmake --preset "ISOTP[Examples]" && cmake --build "build/ISOTP[Examples]"
```

```
ctest --test-dir "build/ISOTP[Examples]" -L example
python tools/docs/check_snippets.py
```

CHAPTER 6

# Configuration

Build-time parameters live in `inc/isotp_config.h`. Each one is guarded by `#ifndef` and range-checked at compile time; override it with a compiler definition, for example `-DISOTP_RX_BUFFER_SIZE=256U`. The two buffer sizes are compiled into `struct isotp_link`; the other values are only the defaults that `isotp_link_config_default()` writes into a `struct isotp_link_config`, so every link can use its own values at run time.

| Parameter | Default | Range | Meaning |
|---|---|---|---|
| `ISOTP_TX_BUFFER_SIZE` | `4095U` | 8 … 2^31−1 | Largest message a link sends, in bytes |
| `ISOTP_RX_BUFFER_SIZE` | `4095U` | 8 … 2^31−1 | Largest message a link receives; longer ones are refused with FC OVFLW |
| `ISOTP_DEFAULT_BLOCK_SIZE` | `8U` | 0 … 255 | BS this receiver requests; 0 = no further Flow Control |
| `ISOTP_DEFAULT_ST_MIN_US` | `0U` | 0 … 127000 | STmin this receiver requests, µs |
| `ISOTP_DEFAULT_N_BS_TIMEOUT_US` | `1000000U` | 1 … 2^31−1 | N_Bs, µs (ISO 15765-2: 1000 ms) |
| `ISOTP_DEFAULT_N_CR_TIMEOUT_US` | `1000000U` | 1 … 2^31−1 | N_Cr, µs (ISO 15765-2: 1000 ms) |
| `ISOTP_DEFAULT_WFT_MAX` | `10U` | 0 … 255 | N_WFTmax: Wait frames accepted in a row |

Messages longer than 4095 bytes use the 32-bit FF_DL of ISO 15765-2:2016; they need a buffer size above 4095. Your peer must support the escape sequence too.

## 6.1 Run-time configuration

`struct isotp_link_config` also holds what has no build-time default:

| Member | Default | Meaning |
|---|---|---|
| `tx_id` | 0 | CAN ID of every frame this link sends (11- or 29-bit, passed to the writer) |
| `write`, `write_context` | none | Frame writer (required) and its context |
| `on_message`, `on_tx_done`, `on_rx_error`, `app_context` | none | Callbacks and their context |
| `padding`, `padding_value` | false, `0xCC` | Pad every sent frame to 8 bytes |

`isotp_link_init()` rejects a configuration without a writer, an STmin above 127 000 µs and timeouts of 0 or above 2^31−1 µs (deadlines are compared modulo 2^32).

## 6.2 Memory footprint

Measured with `sizeof` and `arm-none-eabi-size` (Cortex-M4, `-Os`):

| Object | Size |
|---|---|
| `struct isotp_link`, default 4095-byte buffers | 8 296 bytes |
| `struct isotp_link`, 256-byte buffers | 616 bytes |
| `struct isotp_link`, 64-byte buffers | 232 bytes |
| Per link without the buffers | 104 bytes |
| `struct isotp_link_config` | 48 bytes |
| Code: `isotp_protocol.o` + `isotp_link.o` | 516 + 1 268 = 1 784 bytes of flash, no data, no bss |

The library has no global variables: the RAM it uses is the `struct isotp_link` instances you declare (one per peer) and the stack.

## 6.3 Stack usage

Worst-case stack of each public function, including the library functions it calls, measured with `-fstack-usage -fcallgraph-info=su` and `tools/docs/stack_usage.py`, default configuration, in bytes:

| Function | Cortex-M4 `-Os` | Cortex-M4 `-O0` | x86-64 `-O2` |
|---|---|---|---|
| `isotp_decode()` | 12 | 80 | 8 |
| `isotp_encode_consecutive()` | 16 | 32 | 8 |
| `isotp_encode_first()` | 12 | 48 | 8 |
| `isotp_encode_flow_control()` | 0 | 32 | 8 |
| `isotp_encode_single()` | 16 | 32 | 8 |
| `isotp_frame_pad()` | 16 | 32 | 8 |
| `isotp_link_config_default()` | 0 | 16 | 8 |
| `isotp_link_init()` | 16 | 32 | 16 |
| `isotp_link_invalid_frames()` | 0 | 16 | 8 |
| `isotp_link_poll()` | 80 | 136 | 72 |
| `isotp_link_receive()` | 120 | 200 | 120 |
| `isotp_link_rx_result()` | 0 | 16 | 8 |
| `isotp_link_send()` | 72 | 120 | 88 |
| `isotp_link_send_functional()` | 64 | 112 | 56 |
| `isotp_link_tx_busy()` | 0 | 16 | 8 |
| `isotp_link_tx_result()` | 0 | 16 | 8 |
| `isotp_st_min_decode()` | 0 | 24 | 8 |

| Function | Cortex-M4 `-Os` | Cortex-M4 `-O0` | x86-64 `-O2` |
|---|---|---|---|
| `isotp_st_min_encode()` | 0 | 24 | 8 |

Toolchains: xPack GNU Arm Embedded GCC 14.2.1 (`-mcpu=cortex-m4 -mthumb`) and GCC 13.3.0 (x86-64 Linux).

The figures do not include what runs behind a function pointer: your write function and your callbacks run on top of the library frame that calls them.

| Path | Worst chain (Cortex-M4 `-Os`) | Stack |
|---|---|---|
| Receive | `isotp_link_receive()` → `isotp_link_send_fc()` → `isotp_link_write()` → `isotp_frame_pad()` | 120 B + write function or callback |
| Poll | `isotp_link_poll()` → `isotp_link_write()` → `isotp_frame_pad()` | 80 B + write function |
| Send | `isotp_link_send()` → `isotp_link_write()` → `isotp_frame_pad()` | 72 B + write function |

`on_message()` runs inside `isotp_link_receive()`; when it answers with `isotp_link_send()`, add that chain too (Cortex-M4 `-Os`: up to 120 + 72 bytes plus your own frames).

Re-measure after a change:

```
mkdir -p build/su && for f in src/*.c; do
  arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -Os -std=c11 -I. -Iinc \
      -fstack-usage -fcallgraph-info=su -c "$f" -o "build/su/$(basename "$f" .c).o"
done
python tools/docs/stack_usage.py build/su --markdown --prefix isotp_
```

## 6.4 CMake options

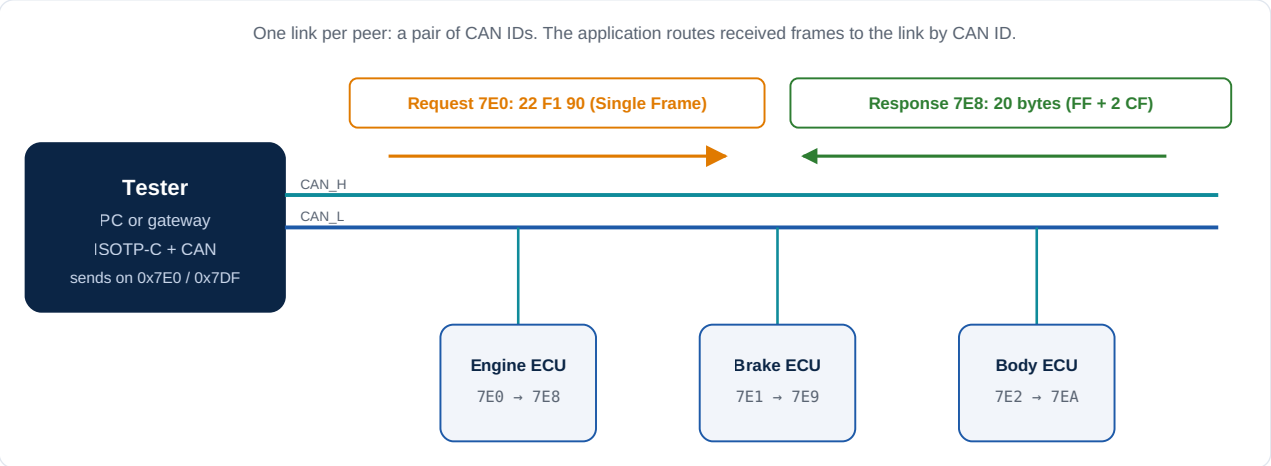
| Option | Default | Meaning |
|---|---|---|
| `ISOTP_UNIT_TESTS` | `OFF` | GoogleTest suite and `coverage` target (built with 5000-byte buffers) |
| `ISOTP_INTEGRATION_TESTS` | `OFF` | pytest suite |
| `ISOTP_EXAMPLES` | `OFF` | Host examples and the bare-metal simulation test |
| `ISOTP_COVERAGE_MIN` | `100` | Coverage gate |
| `ISOTP_MISRA_FAIL_ON_VIOLATION` | `ON` | MISRA gate |
| `ISOTP_MISRA_RULE_TEXTS` | empty | Licensed MISRA rule texts |

CHAPTER 7

# ISO 15765-2 (ISO-TP)

This chapter describes the ISO 15765-2 wire format for classical CAN with normal addressing and exactly what ISOTP-C does with each part of it. The normative reference is the standard (iso.org/standard/84211.html); ISOTP-C includes the 32-bit message length introduced by the 2016 edition.

## 7.1 The bus




ISO-TP runs on top of CAN. Each direction between two nodes uses its own CAN ID — in diagnostics, for example, the tester sends to an ECU on 0x7E0 and the ECU answers on 0x7E8 — and a functional ID (0x7DF) addresses all ECUs at once. One ISOTP-C link is one such pair of IDs: it sends with `tx_id`, and the application gives it the frames of the peer's ID. Both directions of a link work at the same time (full duplex).

## 7.2 Frames




The first byte, or the first two bytes, of every frame are the protocol control information (N_PCI). Its high nibble is the frame type:

| Type | Byte 0 | Further N_PCI | Data | Use |
|---|---|---|---|---|
| Single Frame (SF) | `0` \| SF_DL | — | 1..7 bytes | A whole message of up to 7 bytes |
| First Frame (FF) | `1` \| FF_DL bits 11..8 | byte 1: FF_DL bits 7..0 | 6 bytes | Start of a message of 8..4095 bytes |
| First Frame, escape | `10` | byte 1 `00`, bytes 2..5: FF_DL, 32 bit, big-endian | 2 bytes | Start of a message of more than 4095 bytes |
| Consecutive Frame (CF) | `2` \| SN | — | 1..7 bytes | Next part of the message |
| Flow Control (FC) | `3` \| FS | byte 1: BS, byte 2: STmin | — | The receiver's answer to an FF and after every block |

A First Frame is always 8 bytes long. The last Consecutive Frame carries the remaining bytes; with padding, the rest of the frame is filled with the padding byte.

## 7.3 Flow status, block size and separation time

| Field | Values | Meaning |
|---|---|---|
| FS | `0` CTS, `1` WAIT, `2` OVFLW | Continue to send / wait for the next FC / message too long for the receiver |
| BS | `0` … `255` | Consecutive Frames the sender may send before the next FC; `0`: all |
| STmin | `00` … `7F` | Minimum time between two Consecutive Frames, 0..127 ms |
| STmin | `F1` … `F9` | 100..900 µs |

| Field | Values | Meaning |
|---|---|---|
| STmin | other values | Reserved; a sender treats them as `7F` (127 ms) |

`isotp_st_min_encode()` rounds a requested time up to the next value that can be encoded, so the sender never goes faster than asked; `isotp_st_min_decode()` maps the reserved values to 127 ms.

## 7.4 Timers and N_Result

| Timer | Side | Runs from … to … | Default | On expiry |
|---|---|---|---|---|
| N_Bs | Sender | FF or last CF of a block written → FC received | 1000 ms | `ISOTP_N_TIMEOUT_BS` |
| N_Cr | Receiver | FC written or CF received → next CF received | 1000 ms | `ISOTP_N_TIMEOUT_CR` |

ISO 15765-2 also defines N_As/N_Ar (time for the CAN driver to send a frame) and the performance times N_Br/N_Cs. ISOTP-C writes frames synchronously through your write function, so it does not time them: `ISOTP_N_TIMEOUT_A` is never produced.

| N_Result | Reported when |
|---|---|
| `ISOTP_N_OK` | The message was sent or received completely |
| `ISOTP_N_TIMEOUT_BS` | No Flow Control frame within N_Bs |
| `ISOTP_N_TIMEOUT_CR` | No Consecutive Frame within N_Cr |
| `ISOTP_N_WRONG_SN` | A Consecutive Frame with an unexpected sequence number |
| `ISOTP_N_INVALID_FS` | A Flow Control frame with an unknown flow status |
| `ISOTP_N_UNEXP_PDU` | A Single or First Frame interrupted a reception |
| `ISOTP_N_WFT_OVRN` | More than N_WFTmax Flow Control frames with WAIT in a row |
| `ISOTP_N_BUFFER_OVFLW` | Sender: the receiver answered OVFLW. Receiver: the announced message is longer than `ISOTP_RX_BUFFER_SIZE` |
| `ISOTP_N_ERROR` | The write function failed while sending a frame |

## 7.5 Receiver behavior

What the receiver of ISOTP-C does with every frame, following ISO 15765-2:

| Frame | Condition | Action |
|---|---|---|
| any | Reserved type, SF_DL 0 or longer than the frame, FF not 8 bytes, FF_DL ≤ 7, escape FF_DL ≤ 4095, CF without data, FC shorter than 3 bytes | Ignored, counted as invalid |
| SF | — | Delivered; a reception in progress ends with N_UNEXP_PDU |
| FF | FF_DL fits the buffer | FC CTS sent, reception starts; a reception in progress ends with N_UNEXP_PDU |

| Frame | Condition | Action |
|---|---|---|
| FF | FF_DL too long | FC OVFLW sent, N_BUFFER_OVFLW |
| CF | No reception in progress | Ignored |
| CF | Wrong SN | Reception ends with N_WRONG_SN |
| CF | Fewer data bytes than still expected | Ignored, counted as invalid |
| FC | Not waiting for FC | Ignored |

Frames longer than needed (padding) are accepted in every case.

## 7.6 Scope of this implementation

| Feature of ISO 15765-2 | ISOTP-C |
|---|---|
| Normal addressing, 11- and 29-bit CAN IDs | Yes |
| Normal fixed addressing (29-bit IDs with source and target address) | Yes, through `tx_id` and your routing |
| Extended and mixed addressing | No |
| Classical CAN (8 bytes) | Yes |
| CAN FD (up to 64 bytes) | No |
| 32-bit FF_DL (messages > 4095 bytes) | Yes |
| Functional addressing | Single Frames, as required |
| Sending FC WAIT | No (the receive buffer is always ready) |

## 7.7 Reference frames used in the tests

The ISO standard is not freely available, so the test frames are built from its frame layout. They are checked in `tests/unit/gtest_isotp_decode.cpp`, `tests/unit/gtest_isotp_encode.cpp` and `tests/integration/test_protocol.py`.

| Frame | Bytes |
|---|---|
| SF, UDS DiagnosticSessionControl (extended session) | `02 10 03` |
| SF, 7 bytes | `07 00 01 02 03 04 05 06` |
| FF, 20-byte message | `10 14 00 01 02 03 04 05` |
| FF, 4095-byte message | `1F FF 00 01 02 03 04 05` |
| FF, 4096-byte message (escape) | `10 00 00 00 10 00 AA BB` |
| CF, SN 1 | `21 06 07 08` |
| FC, CTS, BS 8, STmin 20 ms | `30 08 14` |
| FC, OVFLW | `32 00 00` |

CHAPTER 8

# Integration Guide

The library does not touch hardware and does not read a clock. An integration has three parts:

1. **Receive:** move every CAN frame of the peer's CAN ID into `isotp_link_receive()`, together with the current time, from a task or the main loop — not from the CAN interrupt handler.
2. **Transmit:** a write function that queues one CAN frame, and regular calls of `isotp_link_poll()` with the current time.
3. **Application:** `on_message()` for complete messages, and optionally `on_tx_done()` and `on_rx_error()` for results.

Complete, buildable programs for each pattern are described in Examples (p. 34).

## 8.1 The time source

`now_us` is any free-running microsecond counter that wraps at 2^32: a 32-bit timer running at 1 MHz, `k_cyc_to_us_floor32(k_cycle_get_32())` on Zephyr, the DWT cycle counter divided by the core clock in MHz, or `clock_gettime(CLOCK_MONOTONIC)` on Linux. A millisecond tick multiplied by 1000 also works when your STmin and timeouts are whole milliseconds; the library only needs the value to grow monotonically and wrap at 2^32.

## 8.2 RTOS: interrupt, queue and CAN task

```
CAN RX interrupt ──► message queue ──► CAN task ──► isotp_link_receive() ──► on_message()
                                                └──► every 1 ms: isotp_link_poll() ──► write()
```

The interrupt copies the frame (ID, length, data) into a queue and returns. The CAN task waits on the queue with a timeout of one millisecond, passes each frame of the peer's ID to the link, and polls the link after every wake-up. Only this task touches the link, so no mutex is needed. When another task must send, either hand the message to the CAN task through a second queue, or protect `isotp_link_send()`, `isotp_link_receive()` and `isotp_link_poll()` of that link with one mutex.

## 8.3 Adding the library to a CMake build

```cmake
include(path/to/isotp-c/isotp.cmake)
target_link_libraries(app PRIVATE isotp)
target_link_libraries(isotp PUBLIC zephyr_interface)   # Zephyr applications only
```

Override the configuration on the target, for example `target_compile_definitions(isotp PUBLIC ISOTP_RX_BUFFER_SIZE=512U)`; `PUBLIC` makes the application see the same `struct isotp_link` size as the library.

## 8.4 Bare metal and host

**Bare metal.** The CAN interrupt fills a small ring buffer of frames; the main loop drains it, feeds the link and polls it, then sleeps until the next interrupt. A periodic timer interrupt (for example SysTick at 1 kHz) wakes the loop often enough for STmin and the timeouts. The bare-metal example (p. 34) shows the complete program.

**Linux.** Use SocketCAN: a raw `CAN_RAW` socket with a filter on the peer's ID, `poll()` with a timeout of one millisecond, and `clock_gettime(CLOCK_MONOTONIC)`. The kernel also has its own ISO-TP socket (`CAN_ISOTP`); ISOTP-C is useful on Linux when you need the same code as on the ECU, or full control over timing in a test bench.

## 8.5 Notes from hardware bring-up

- **Padding.** Many ECUs and testers expect every frame to have 8 data bytes. If the peer ignores your frames, set `padding = true`.
- **Acceptance filters.** Configure the CAN controller to accept the peer's ID (and the functional ID in an ECU); everything else costs interrupt time.
- **Transmit mailboxes.** With one mailbox and STmin 0, Consecutive Frames leave as fast as the bus allows; return `ISOTP_BUSY` when the mailbox is full and the link retries.
- **Frame order.** Use a single transmit FIFO or one mailbox for ISO-TP: priority-based mailboxes can send Consecutive Frames out of order, which the receiver reports as a wrong sequence number.
- **RX overruns.** A receiver that asks for STmin 0 and BS 0 receives frames back to back (about 4 000 frames per second at 500 kbit/s): make the RX queue deep enough or ask for a block size.

CHAPTER 9

# Examples

Three examples show the patterns of the Integration Guide (p. 32). The bare-metal example is also a test: CI runs it on a simulated board.

| Example | Role | Platform | Built and run in CI |
|---|---|---|---|
| Bare metal | ECU | Any MCU; runs on a PC against a simulated tester | Yes, as a self-checking test |
| Linux SocketCAN | Tester | Linux | Built |
| Zephyr | ECU | Any Zephyr board with a CAN controller | No, needs a Zephyr workspace |

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

## 9.1 Bare metal

A UDS-style ECU on request ID 0x7E0 (and functional ID 0x7DF) that answers on 0x7E8. It supports TesterPresent, and reading and writing a 17-byte VIN with Read/WriteDataByIdentifier 0xF190 — so the response to a read and the write request are both segmented — and answers other services with a negative response.

| File | Contents |
|---|---|
| `examples/bare_metal/main.c` | The ECU: CAN RX interrupt → ring buffer → main loop → link → services |
| `examples/bare_metal/board.h` | The four functions to port: `board_init()`, `board_can_write()`, `board_micros()`, `board_idle()` |
| `examples/bare_metal/board_host.c` | Simulated board for the PC: a tester with its own link on a virtual bus |

The ECU asks for a block size of 2 and an STmin of 1 ms; the simulated tester asks for a block size of 4 and an STmin of 2 ms, so both directions use several Flow Control frames. Before every request the tester injects a malformed frame on the request ID and a frame of another node. The program exits with 0 when every response matches. Real output:

```
Simulated CAN bus at 500000 bit/s: ECU 0x7E0/0x7E8, tester with BS 4, STmin 2 ms
  request:  3E 00
  response: 7E 00
TesterPresent (functional)           OK
  request:  22 F1 90
  response: 62 F1 90 53 46 53 49 53 4F 54 50 43 32 30 30 30 30 30 30 31
ReadDataByIdentifier VIN             OK
  request:  2E F1 90 53 46 53 49 53 4F 54 50 43 32 30 30 30 30 30 30 32
  response: 6E F1 90
```

```
WriteDataByIdentifier VIN            OK
  request:  22 F1 90
  response: 62 F1 90 53 46 53 49 53 4F 54 50 43 32 30 30 30 30 30 30 32
ReadDataByIdentifier VIN again       OK
  request:  10 01
  response: 7F 10 11
Unsupported service                  OK
  request:  3E 80
TesterPresent, response suppressed   OK
0 of 6 exchanges failed
```

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

| Function | On an STM32 with bxCAN, for example |
|---|---|
| `board_init()` | Clock and pins, bit timing, filters for 0x7E0 and 0x7DF, enable the FIFO 0 interrupt, SysTick at 1 kHz |
| `board_can_write()` | Put the frame into a free TX mailbox; `ISOTP_BUSY` when none is free |
| `board_micros()` | A 32-bit timer at 1 MHz, or the DWT cycle counter / (core clock in MHz) |
| `board_idle()` | `__WFI()`; the SysTick interrupt wakes it every millisecond |
| FIFO 0 interrupt | Read the frame and call `app_can_rx_isr()` |

## 9.2 Linux SocketCAN

`isotp_request` sends one request from a PC and prints the response:

```
isotp_request can0 7E0 7E8 22 F1 90
```

| File | Contents |
|---|---|
| `examples/linux_socketcan/main.c` | Raw CAN socket with a filter on the response ID, `poll()` every millisecond, `CLOCK_MONOTONIC` |

It pads frames to 8 bytes, waits up to 2 s for the response and prints the N_Result on an error. Without hardware, create a virtual bus with `sudo ip link add dev vcan0 type vcan && sudo ip link set up vcan0` and run an ECU on it.

## 9.3 Zephyr

The ECU on the Zephyr CAN API, for the CAN controller chosen as `zephyr,canbus`:

```
west build -b <board> examples/zephyr
```

| File | Contents |
|---|---|
| `examples/zephyr/src/main.c` | Filters for 0x7E0/0x7DF into a `k_msgq`, one thread that feeds and polls the link every millisecond, `can_send()` with `-EAGAIN` mapped to `ISOTP_BUSY` |
| `examples/zephyr/CMakeLists.txt` | Includes `isotp.cmake` and links the library to `zephyr_interface` |

| File | Contents |
|---|---|
| `examples/zephyr/prj.conf` | `CONFIG_CAN=y`, no heap |

It answers TesterPresent and replies *serviceNotSupported* to everything else; add your services in `on_request()`.

CHAPTER 10

# API Reference

This chapter documents every public function, type and macro of ISOTP-C. Include the umbrella header:

```
#include "isotp.h"
```

| Header | Contents |
|---|---|
| `isotp.h` | All headers; library version |
| `isotp_config.h` | Build-time configuration |
| `isotp_defs.h` | Frame types, flow status, layout limits, return and N_Result codes, big-endian helpers |
| `isotp_protocol.h` | Frame codec: decode, encode SF/FF/CF/FC, padding, STmin |
| `isotp_link.h` | Transport layer of one connection |

## 10.1 Version — `isotp.h`

| Macro | Meaning |
|---|---|
| `ISOTP_VERSION_MAJOR`, `ISOTP_VERSION_MINOR`, `ISOTP_VERSION_PATCH` | Version numbers from the Git tag (CMake) or the fall-backs in `isotp.h` |
| `ISOTP_VERSION` | Version string `"MAJOR.MINOR.PATCH"` |

## 10.2 Configuration — `isotp_config.h`

| Macro | Default | Meaning |
|---|---|---|
| `ISOTP_TX_BUFFER_SIZE` | `4095U` | Largest message a link sends |
| `ISOTP_RX_BUFFER_SIZE` | `4095U` | Largest message a link receives |
| `ISOTP_DEFAULT_BLOCK_SIZE` | `8U` | Default BS requested by the receiver |
| `ISOTP_DEFAULT_ST_MIN_US` | `0U` | Default STmin requested by the receiver |
| `ISOTP_DEFAULT_N_BS_TIMEOUT_US` | `1000000U` | Default N_Bs |
| `ISOTP_DEFAULT_N_CR_TIMEOUT_US` | `1000000U` | Default N_Cr |
| `ISOTP_DEFAULT_WFT_MAX` | `10U` | Default N_WFTmax |

Ranges and memory: see Configuration (p. 25).

## 10.3 Definitions — `isotp_defs.h`

| Macro | Value | Meaning |
|---|---|---|
| `ISOTP_OK` | `0` | Success |
| `ISOTP_ERROR` | `-1` | Invalid argument, malformed frame or insufficient capacity |
| `ISOTP_BUSY` | `-2` | Try again later: transmission in progress or CAN driver busy |
| `ISOTP_PCI_SINGLE_FRAME` … `ISOTP_PCI_FLOW_CONTROL` | `0x0` … `0x3` | Frame types |
| `ISOTP_FS_CONTINUE`, `ISOTP_FS_WAIT`, `ISOTP_FS_OVERFLOW` | `0x0`, `0x1`, `0x2` | Flow status |
| `ISOTP_CAN_DL` | `8U` | Data bytes of a classical CAN frame |
| `ISOTP_SF_MAX_DL` | `7U` | Largest Single Frame payload |
| `ISOTP_FF_DL_12BIT_MAX` | `4095U` | Largest 12-bit FF_DL |
| `ISOTP_FF_SHORT_DATA`, `ISOTP_FF_LONG_DATA` | `6U`, `2U` | Payload bytes in a First Frame (12-bit / 32-bit FF_DL) |
| `ISOTP_CF_MAX_DATA` | `7U` | Payload bytes in a full Consecutive Frame |
| `ISOTP_FC_DL` | `3U` | Flow Control frame length without padding |
| `ISOTP_SN_MASK`, `ISOTP_FIRST_SN` | `0x0FU`, `1U` | Sequence numbers |
| `ISOTP_ST_MIN_MAX_US` | `127000U` | Largest STmin |
| `ISOTP_N_OK` … `ISOTP_N_ERROR` | `0` … `-9` | N_Result codes, see Protocol |

```
static inline uint32_t isotp_get_u32_be(const uint8_t* data)
```

Reads a big-endian 32-bit value from four bytes.

```
static inline void isotp_put_u32_be(uint8_t* data, uint32_t value)
```

Writes a 32-bit value as four big-endian bytes.

## 10.4 Codec — `isotp_protocol.h`

```
struct isotp_pdu
{
    uint8_t type;
    uint32_t message_length;
    uint8_t data_offset;
    uint8_t data_length;
    uint8_t sequence_number;
    uint8_t flow_status;
    uint8_t block_size;
    uint8_t st_min;
};
```

One decoded frame. Only the members of the decoded type are meaningful; the others are zero. The payload is not copied: it is `frame[data_offset]` … `frame[data_offset + data_length - 1]` of the decoded frame.

| Member | SF | FF | CF | FC |
|---|---|---|---|---|
| `type` | `ISOTP_PCI_SINGLE_FRAME` | `ISOTP_PCI_FIRST_FRAME` | `ISOTP_PCI_CONSECUTIVE_FRAME` | `ISOTP_PCI_FLOW_C` |
| `message_length` | SF_DL | FF_DL | — | — |
| `data_offset`, `data_length` | 1, SF_DL | 2, 6 (or 6, 2) | 1, frame length − 1 | — |
| `sequence_number` | — | — | SN | — |
| `flow_status`, `block_size`, `st_min` | — | — | — | FS, BS, raw STmin |

## isotp_decode

```
int isotp_decode(const uint8_t* frame, size_t length, struct isotp_pdu* pdu);
```

Validates one received CAN frame and describes it. Rejects the frames ISO 15765-2 tells a receiver to ignore (see Receiver behavior (p. 30)). The flow status of a Flow Control frame is not range-checked: an unknown value is an N_INVALID_FS condition for the sender.

| Parameter | Description |
|---|---|
| `frame` | CAN frame data |
| `length` | CAN data length, 1..8 |
| `pdu` | Receives the description |

| Return | Meaning |
|---|---|
| `ISOTP_OK` | `pdu` describes a valid frame |
| `ISOTP_ERROR` | `NULL` argument, invalid length or malformed frame |

## isotp_encode_single

```
int isotp_encode_single(uint8_t* frame, size_t capacity, const uint8_t* payload, uint32_t
length);
```

Encodes a Single Frame.

| Parameter | Description |
|---|---|
| `frame` | Output buffer |
| `capacity` | Size of `frame`, at least `length` + 1 |
| `payload` | Message bytes |

| Parameter | Description |
|---|---|
| `length` | Message length, 1..7 |

| Return | Meaning |
|---|---|
| `length` + 1 | Frame length |
| `ISOTP_ERROR` | `NULL` argument, length out of range or buffer too small |

## isotp_encode_first

```
int isotp_encode_first(uint8_t* frame, size_t capacity, const uint8_t* payload,
                       uint32_t message_length);
```

Encodes a First Frame and its first payload bytes: a 12-bit FF_DL and 6 bytes for messages up to 4095 bytes, the escape sequence with a 32-bit FF_DL and 2 bytes above.

| Parameter | Description |
|---|---|
| `frame` | Output buffer |
| `capacity` | Size of `frame`, at least 8 |
| `payload` | Start of the message; 6 (or 2) bytes are read |
| `message_length` | Length of the whole message, at least 8 |

| Return | Meaning |
|---|---|
| `8` | Frame length |
| `ISOTP_ERROR` | `NULL` argument, message shorter than 8 bytes or buffer too small |

## isotp_encode_consecutive

```
int isotp_encode_consecutive(uint8_t* frame, size_t capacity, uint8_t sequence_number,
                             const uint8_t* data, uint32_t length);
```

Encodes a Consecutive Frame.

| Parameter | Description |
|---|---|
| `frame` | Output buffer |
| `capacity` | Size of `frame`, at least `length` + 1 |
| `sequence_number` | SN, 0..15 |
| `data` | Segment bytes |
| `length` | Segment length, 1..7 |

| Return | Meaning |
|---|---|
| `length` + 1 | Frame length |
| `ISOTP_ERROR` | `NULL` argument, value out of range or buffer too small |

## isotp_encode_flow_control

```
int isotp_encode_flow_control(uint8_t* frame, size_t capacity, uint8_t flow_status,
                              uint8_t block_size, uint8_t st_min);
```

Encodes a Flow Control frame.

| Parameter | Description |
|---|---|
| `frame` | Output buffer |
| `capacity` | Size of `frame`, at least 3 |
| `flow_status` | `ISOTP_FS_CONTINUE`, `ISOTP_FS_WAIT` or `ISOTP_FS_OVERFLOW` |
| `block_size` | BS, 0 = no further Flow Control |
| `st_min` | Raw STmin byte, see `isotp_st_min_encode()` |

| Return | Meaning |
|---|---|
| `3` | Frame length |
| `ISOTP_ERROR` | `NULL` argument, unknown flow status or buffer too small |

## isotp_frame_pad

```
int isotp_frame_pad(uint8_t* frame, size_t capacity, size_t length, uint8_t value);
```

Fills a frame with a padding byte up to 8 data bytes.

| Parameter | Description |
|---|---|
| `frame` | Frame buffer |
| `capacity` | Size of `frame`, at least 8 |
| `length` | Current frame length, 1..8 |
| `value` | Padding byte |

| Return | Meaning |
|---|---|
| `8` | New frame length |
| `ISOTP_ERROR` | `NULL` argument, length out of range or buffer too small |

### isotp_st_min_encode

```
uint8_t isotp_st_min_encode(uint32_t us);
```

Encodes a separation time for a Flow Control frame, rounded up: 0 µs gives `0x00`, 1..900 µs give `0xF1`..`0xF9`, 901..127 000 µs give `0x01`..`0x7F`, larger values `0x7F`.

| Parameter | Description |
|---|---|
| `us` | Separation time in microseconds |

| Return | Meaning |
|---|---|
| `0x00`..`0x7F`, `0xF1`..`0xF9` | Raw STmin byte |

### isotp_st_min_decode

```
uint32_t isotp_st_min_decode(uint8_t st_min);
```

Decodes a received STmin byte. Reserved values give 127 000 µs, as ISO 15765-2 requires.

| Parameter | Description |
|---|---|
| `st_min` | Raw STmin byte |

| Return | Meaning |
|---|---|
| 0..127 000 | Separation time in microseconds |

## 10.5 Link — `isotp_link.h`

```
typedef int (*isotp_can_write_fn)(void* context, uint32_t can_id, const uint8_t* data,
                                  size_t length);
```

Writes one CAN frame. Returns `0` when the frame is queued, `ISOTP_BUSY` when the driver has no free mailbox (a Consecutive Frame is retried on the next poll), any other value on a driver error. `data` is valid only during the call.

```
typedef void (*isotp_message_handler_t)(const uint8_t* data, uint32_t length, void*
context);
```

Receives one complete message. `data` is valid only during the call. The handler may call `isotp_link_send()` on the same link.

```
typedef void (*isotp_result_fn)(int n_result, void* context);
```

Receives the N_Result of a finished transmission (`on_tx_done`) or of a failed reception (`on_rx_error`).

```
struct isotp_link_config
{
    uint32_t tx_id;
    isotp_can_write_fn write;
    void* write_context;
    isotp_message_handler_t on_message;
    isotp_result_fn on_tx_done;
    isotp_result_fn on_rx_error;
    void* app_context;
    uint8_t block_size;
    uint32_t st_min_us;
    uint32_t n_bs_timeout_us;
    uint32_t n_cr_timeout_us;
    uint8_t wft_max;
    bool padding;
    uint8_t padding_value;
};
```

| Member | Meaning |
|---|---|
| `tx_id` | CAN ID of every frame this link sends |
| `write`, `write_context` | Frame writer (required) and the context passed to it |
| `on_message` | Complete received message; `NULL` discards it |
| `on_tx_done` | End of every transmission started by `isotp_link_send()`; may be `NULL` |
| `on_rx_error` | Failed or interrupted reception; may be `NULL` |
| `app_context` | Context passed to the three callbacks above |
| `block_size`, `st_min_us` | BS and STmin requested from the sender (STmin 0..127 000 µs) |
| `n_bs_timeout_us`, `n_cr_timeout_us` | N_Bs and N_Cr, 1..2^31−1 µs |
| `wft_max` | N_WFTmax: Wait frames accepted in a row |
| `padding`, `padding_value` | Pad every sent frame to 8 bytes with this byte |

`struct isotp_link` holds the state of one link: a copy of the configuration, both message buffers, both state machines and the invalid-frame counter. Declare it with static storage (it holds the buffers) and treat its members as private.

## isotp_link_config_default

```
void isotp_link_config_default(struct isotp_link_config* config);
```

Fills a configuration with CAN ID 0, no writer, no callbacks, no padding (padding byte `0xCC`) and the `ISOTP_DEFAULT_*` flow-control and timing values. Nothing happens for `NULL`.

## isotp_link_init

```
int isotp_link_init(struct isotp_link* link, const struct isotp_link_config* config);
```

Copies the configuration and clears both directions. Can be called again to reset a link.

| Parameter | Description |
|---|---|
| `link` | Link state to initialize |
| `config` | Configuration copied into the link |

| Return | Meaning |
|---|---|
| `ISOTP_OK` | Link ready |
| `ISOTP_ERROR` | `NULL` argument, no writer, STmin above 127 000 µs, or a timeout of 0 or above 2^31−1 µs |

## isotp_link_receive

```
void isotp_link_receive(struct isotp_link* link, const uint8_t* data, size_t length,
                        uint32_t now_us);
```

Processes one received CAN frame of the peer. Delivers complete messages to `on_message()`, answers First Frames and finished blocks with Flow Control, hands Flow Control frames to the transmitter, ignores and counts malformed frames. See The receiver (p. 14).

| Parameter | Description |
|---|---|
| `link` | Link state; nothing happens for `NULL` |
| `data` | CAN frame data |
| `length` | CAN data length, 1..8 |
| `now_us` | Current time in microseconds (free-running, wraps at 2^32) |

## isotp_link_send

```
int isotp_link_send(struct isotp_link* link, const uint8_t* payload, uint32_t length,
                    uint32_t now_us);
```

Starts sending one message. Up to 7 bytes are sent at once as a Single Frame and reported to `on_tx_done()` before the call returns. A longer message is copied into the link, its First Frame is written, and the rest is sent from `isotp_link_poll()`.

| Parameter | Description |
|---|---|
| `link` | Link state |
| `payload` | Message bytes; may be reused when the call returns |
| `length` | Message length, 1..`ISOTP_TX_BUFFER_SIZE` |
| `now_us` | Current time in microseconds |

| Return | Meaning |
|---|---|
| `ISOTP_OK` | Message accepted |

| Return | Meaning |
|---|---|
| `ISOTP_BUSY` | A segmented transmission is in progress, or the writer is busy |
| `ISOTP_ERROR` | `NULL` argument, length 0 or message too long |
| other | Error returned by the writer; nothing was started |

### isotp_link_send_functional

```
int isotp_link_send_functional(const struct isotp_link* link, uint32_t can_id,
                               const uint8_t* payload, uint32_t length);
```

Sends a Single Frame with a functional CAN ID. Independent of the link's transmission state; does not call `on_tx_done()`.

| Parameter | Description |
|---|---|
| `link` | Link state (its writer and padding are used) |
| `can_id` | Functional CAN ID |
| `payload` | Message bytes |
| `length` | Message length, 1..7 |

| Return | Meaning |
|---|---|
| `ISOTP_OK` | Frame written |
| `ISOTP_ERROR` | `NULL` argument or length out of range |
| other | Error returned by the writer |

### isotp_link_poll

```
void isotp_link_poll(struct isotp_link* link, uint32_t now_us);
```

Sends the next Consecutive Frame when the separation time has passed (at most one per call), and ends a transmission or reception whose timer N_Bs or N_Cr has expired.

| Parameter | Description |
|---|---|
| `link` | Link state; nothing happens for `NULL` |
| `now_us` | Current time in microseconds |

## Diagnostics

```
bool isotp_link_tx_busy(const struct isotp_link* link);
```

```
int isotp_link_tx_result(const struct isotp_link* link);
```

```c
int isotp_link_rx_result(const struct isotp_link* link);
```

```c
uint32_t isotp_link_invalid_frames(const struct isotp_link* link);
```

| Function | Returns | For `NULL` |
|---|---|---|
| `isotp_link_tx_busy()` | `true` while a segmented transmission is in progress | `false` |
| `isotp_link_tx_result()` | N_Result of the last finished transmission | `ISOTP_N_ERROR` |
| `isotp_link_rx_result()` | N_Result of the last finished or failed reception | `ISOTP_N_ERROR` |
| `isotp_link_invalid_frames()` | Malformed frames ignored since `isotp_link_init()` | `0` |

## 10.6 Return codes summary

| Value | Meaning |
|---|---|
| `ISOTP_OK` (`0`) | Success |
| `ISOTP_ERROR` (`-1`) | Invalid argument, malformed frame or insufficient capacity |
| `ISOTP_BUSY` (`-2`) | Try again later |
| `1`..`8` | Frame length (encoders, `isotp_frame_pad()`) |
| other | Passed through from your write function |

CHAPTER 11

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

## 11.1 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 `-DISOTP_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 gate was verified by adding an early `return` (Rule 15.5): the target reported the violation and failed.

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.

## 11.2 Current status

| Item | Result |
|---|---|
| Cppcheck MISRA addon (2.13, the version in CI) | 0 violations |
| Cppcheck `--enable=all` | 0 findings |
| Unit test coverage of `src/` | 100 % lines (335), 100 % branches (213) |
| Deviations | 3, all advisory, listed below |

## 11.3 Manual review notes

| Guideline | How the library complies |
|---|---|
| Dir 4.1 (run-time failures) | Every copy into a message buffer is bounded by an invariant written next to it: a First Frame is only accepted when its FF_DL fits `ISOTP_RX_BUFFER_SIZE`, and every Consecutive Frame copies at most the bytes still missing. `isotp_link_send()` rejects messages longer than `ISOTP_TX_BUFFER_SIZE`. `inc/isotp_config.h` rejects sizes and defaults out of range at compile time. Time arithmetic is unsigned and wraps by design; deadlines are compared modulo 2^32. |
| Dir 4.11 (validity of arguments) | Every public function checks its pointer arguments, lengths and ranges and returns `ISOTP_ERROR` (or ignores the call, for the `void` functions; the accessors return a neutral value for `NULL`). The inline big-endian helpers in `isotp_defs.h` take a pointer to four readable bytes, like `memcpy()`. |
| Dir 4.12, Rule 21.3 (dynamic memory) | No heap. All state is in the caller-owned `struct isotp_link`. |
| 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. A callback that calls back into the same link (for example `isotp_link_send()` from `on_message()`) is a bounded, one-level re-entry. |
| Rule 21.6 (standard I/O) | No `<stdio.h>`. The library includes only `<stddef.h>`, `<stdint.h>`, `<stdbool.h>` and `<string.h>`. |

## 11.4 Deviation register

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

- **Location:** return type of `isotp_decode()`, the `isotp_encode_*()` functions, `isotp_frame_pad()`, `isotp_link_init()`, `isotp_link_send()`, `isotp_link_send_functional()`, `isotp_link_tx_result()`, `isotp_link_rx_result()`, of the `isotp_can_write_fn` callback and the N_Result argument of `isotp_result_fn`.
- **Rationale:** the value is a status code (`ISOTP_OK`, `ISOTP_ERROR`, `ISOTP_BUSY`, an N_Result code, a frame length of at most 8, or the code the write callback returned), not a quantity whose size matters. `int` lets an integrator return the error code of an existing CAN driver from the write callback without a conversion. `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 compares them with the named codes or uses a frame length (1..8) that the encoders produced.

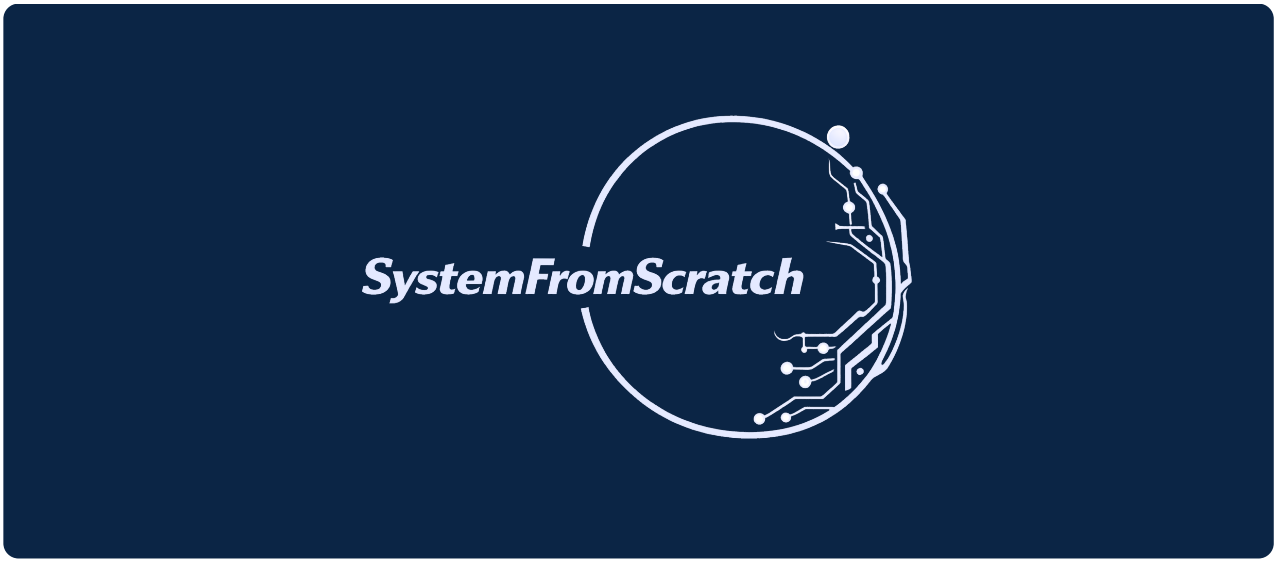
### DEV-02: macros not used inside the library (Rule 2.5, advisory)

- **Location:** `ISOTP_N_TIMEOUT_A` in `inc/isotp_defs.h`, `ISOTP_VERSION_*` in `isotp.h`.
- **Rationale:** these are the public names of protocol values and configuration. They are used by applications; `ISOTP_N_TIMEOUT_A` is kept so that the N_Result set of ISO 15765-2 is complete even though the library writes synchronously and never produces it.

### 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 `isotp_link_tx_busy()`, `isotp_link_invalid_frames()` and the codec functions that only the link calls.
- **Rationale:** they are the library interface and are called by applications. Giving them internal linkage would remove them from the API.

# Author & Company




ISOTP-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** · ORCID 0009-0005-6965-0911 (orcid.org/0009-0005-6965-0911)

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

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

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

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

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

For ISOTP-C, Anton rewrote the library that started as a fork of isotp-c by Li Shen and Simon Cahill: the layered codec and link of version 2, the unit and integration tests with the Python twin, the examples 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 |

ISOTP-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) |

| Channel | Address |
|---|---|
| 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/isotp-c/issues). Please include the library version, your compiler and target, and — for communication problems — a capture of the bytes on the bus.
- **Contributions:** see Contributing to the library (p. 23).
- **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 ISOTP-C; the MIT License grants no rights to use them, and they may not be used to endorse or promote products derived from ISOTP-C without prior written permission.

ISOTP-C implements the transport protocol published by ISO in ISO 15765-2 (iso.org/standard/84211.html). The test vectors are built from the frame layout of that standard. ISOTP-C is an independent implementation and is not affiliated with or endorsed by ISO.

Version 1 of this library was derived from isotp-c (github.com/SimonCahill/isotp-c) by Li Shen and Simon Cahill, released under the MIT License; their copyright notices are kept in the license (p. 55).

APPENDIX A

# Changelog

All notable changes to this project are documented in this file. The format follows Keep a Changelog (keepachangelog.com/en/1.1.0/), and the project adheres to Semantic Versioning (semver.org/spec/v2.0.0.html).

## A.1 Unreleased

## A.2 2.0.1 - 2026-10-26

Version 2.0.0 was tagged but not published as a GitHub Release, and its tag cannot be reused. 2.0.1 is the first published release of version 2: it contains every change listed under 2.0.0 (github.com/AntonVynohradov/isotp-c/compare/v1.6.0...v2.0.0) and the fixes below. The library code is unchanged.

### Fixed

- The documentation is deployed after the Release workflow of a `vX.Y.Z` tag, from the tagged commit, instead of after every build on `main`.
- The documentation is pushed to `gh-pages` with the deploy key again; the `gh-pages` ruleset rejects pushes made with `GITHUB_TOKEN`.

## A.3 2.0.0 - 2026-10-26

Version 2 is a rewrite of the library to the SystemFromScratch® protocol-library standard. The public API changed; the wire format is ISO 15765-2 as before.

### Added

- Layered library: `isotp_defs.h` (constants, N_Result codes), `isotp_protocol.h` (frame codec: `isotp_decode()`, `isotp_encode_*()`, `isotp_frame_pad()`, STmin) and `isotp_link.h` (transport layer), each usable on its own.
- Explicit time input: every link call takes `now_us`; the library no longer calls `isotp_user_get_us()` or any other platform function.
- `struct isotp_link_config` with run-time block size, STmin, N_Bs, N_Cr, N_WFTmax and padding per link; `isotp_link_config_default()`.
- Callbacks with a context: `on_message()`, `on_tx_done()` (every transmission, with its N_Result) and `on_rx_error()`.
- `isotp_link_send_functional()` for functional (broadcast) Single Frames.
- Diagnostics: `isotp_link_tx_busy()`, `isotp_link_tx_result()`, `isotp_link_rx_result()`, `isotp_link_invalid_frames()`.
- Examples: bare-metal UDS-style ECU with a simulated tester (CTest), Linux SocketCAN request tool, Zephyr application.
- Python twin of the codec with a C/Python cross-check and end-to-end tests of the C link against the twin.

- Documentation website in the SystemFromScratch® style, MISRA deviation register, measured RAM and stack figures, `CITATION.cff`, release workflow.

### Changed

- **Breaking:** `IsoTpLink` is now `struct isotp_link`; `isotp_init_link()`, `isotp_on_can_message()`, `isotp_send()`, `isotp_poll()` are replaced by `isotp_link_init()`, `isotp_link_receive()`, `isotp_link_send()`, `isotp_link_poll()`.
- **Breaking:** the message buffers are part of `struct isotp_link`, sized by `ISOTP_TX_BUFFER_SIZE` and `ISOTP_RX_BUFFER_SIZE`, instead of caller pointers.
- **Breaking:** received messages are delivered to `on_message()`; the polled `isotp_receive()` and the weak `isotp_user_*()` functions are removed.
- **Breaking:** configuration macros renamed from `ISO_TP_*` to `ISOTP_*`; the compile-time switches for callbacks, padding and the send argument are now run-time configuration.
- Return codes reduced to `ISOTP_OK`, `ISOTP_ERROR`, `ISOTP_BUSY`; protocol outcomes use the `ISOTP_N_*` N_Result codes.
- One Consecutive Frame per `isotp_link_poll()` call, with wrap-safe timers.

### Fixed

- The 32-bit FF_DL is encoded and decoded big-endian on every host (it was byte-swapped only on little-endian hosts).
- Reserved STmin values are treated as 127 ms, as ISO 15765-2 requires (they were 0).
- The requested STmin is rounded up when encoded, never down.
- Flow Control frames are only accepted while the sender waits for one; an escape First Frame announcing 4095 bytes or less is ignored.
- Segmented transmissions are no longer allowed to functional addresses.
- The N_Cr timer works when the deadline is 0 after a clock wrap.
- No `<stdio.h>`, `snprintf()` or `assert()` in the library.

### Quality

- 82 unit tests, 100 % line and branch coverage of `src/` (gate in CI).
- 62 integration tests (pytest) against the Python twin.
- Cppcheck MISRA C:2012 addon: 0 violations (gate in CI); 3 documented advisory deviations.

## A.4 1.6.0 and earlier

Versions 1.x of the fork kept the API of isotp-c by Li Shen and Simon Cahill. See the Git history (github.com/AntonVynohradov/isotp-c/commits/main) and the releases (github.com/AntonVynohradov/isotp-c/releases).

APPENDIX B

# License

ISOTP-C is free and open-source software, licensed under the **MIT License** (opensource.org/license/mit).

| Item | Value |
|---|---|
| Copyright | © 2019-2024 Li Shen & Co-Operators; © 2024 Simon Cahill & Contributors; © 2026 Anton Vynohradov |
| License | MIT License |
| SPDX identifier | `MIT` |
| Applies to | The library (`isotp.h`, `inc/`, `src/`), tests, examples, tools and this documentation |

The library keeps the license of the project it was forked from. The documentation tooling shared by the SystemFromScratch® protocol libraries is published under the same license in this repository.

## 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, **merge** it into other software, **distribute** and **sublicense** it, in source or binary form, alone or as part of a product, and **sell** copies.

## B.2 What it requires

- Include the copyright notices and the permission notice in all copies or substantial portions of the software.

Firmware that links the library in binary form meets this by making the license text and the copyright notices 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 liability of the authors for any claim or damage.
- No trademark rights: the license does not let you use the names of the authors or of SystemFromScratch® to endorse your product.

## B.4 Trademarks

SystemFromScratch® is a registered trademark; the MIT License grants no rights to use the name or logo, and they may not be used to endorse or promote products derived from ISOTP-C without prior written permission. ISO 15765-2 is a standard of the International Organization for Standardization (ISO); the name is used in this documentation only to identify the protocol that ISOTP-C implements. ISOTP-C is an independent implementation and is not affiliated with or endorsed by ISO.

## B.5 Full license text


```
MIT License

Copyright (c) 2019-2024 Li Shen & Co-Operators
Copyright (c) 2024 Simon Cahill & Contributors.
Copyright (c) 2026 Anton Vynohradov

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```