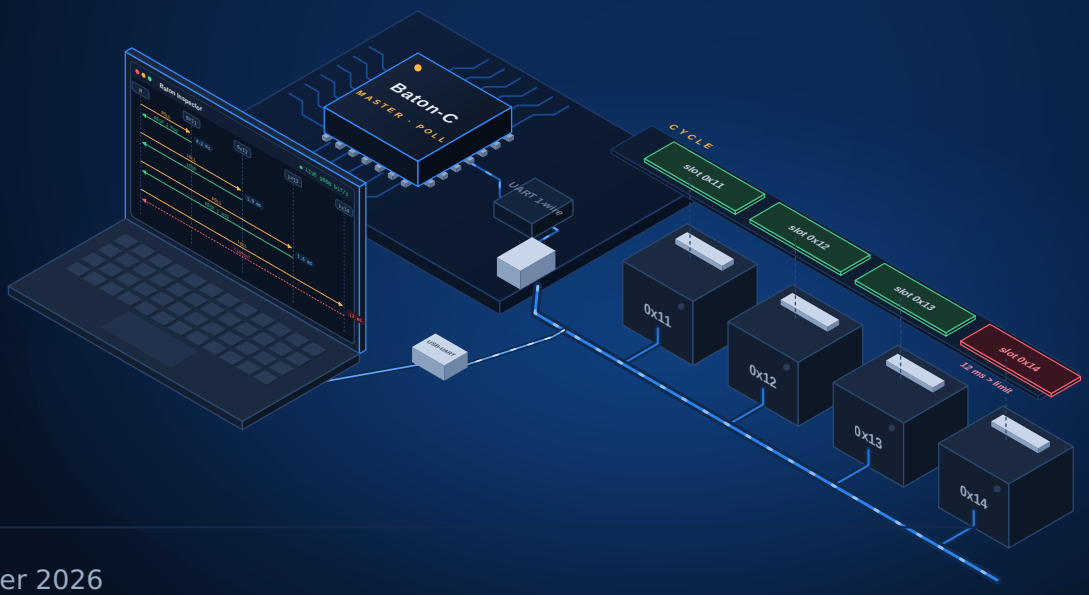
SystemFromScratch

OPEN-SOURCE EMBEDDED LIBRARY

Baton-C

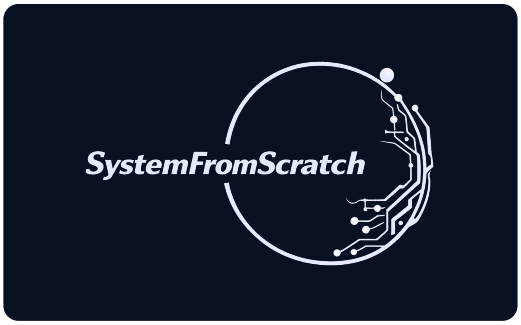
Deterministic Master/Slave Messaging over a Single-Wire UART

USER MANUAL



Version 1.0.0 · October 2026
Anton Vynohradov
systemfromscratch.com

Imprint



Baton-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/Baton-C/
Source code: https://github.com/AntonVynohradov/Baton-C

Copyright © 2026 Anton Vynohradov.

The Baton-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 Baton-C without prior written permission.

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

Contents

1 Introduction 6

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

2 Getting Started 9

2.1 Get the library 9
2.2 Add it to your project 9
2.3 Prerequisites for development 9
2.4 Build with presets 10
2.5 Unit tests and coverage 10
2.6 Integration tests 11
2.7 Static analysis 11
2.8 Formatting 11
2.9 Versioning and releases 11
2.10 Documentation 12

3 How It Works 13

3.1 Layers 13
3.2 Data flow 14
3.3 The parser 14
3.4 The encoder 14
3.5 Event queues 15
3.6 State mailboxes 15
3.7 Packing a frame 15
3.8 State machines 16
3.9 Memory model 16
3.10 Error handling 16
3.11 Threading and reentrancy 17
3.12 Design decisions 17

4 Using the Library 19

4.1 Define the messages 19
4.2 Set up a node 19
4.3 The write function 20
4.4 Run the node 21
4.5 Write a slave 21
4.6 Write a master 22
4.7 Events and states 22
4.8 Return values 22
4.9 Diagnostics 23
4.10 The codec without the node 23
4.11 Common mistakes 23

5 Extending the Library 25

5.1 Where to extend ... 25
5.2 Adding a message ... 25
5.3 Adding a transport ... 25
5.4 Data models ... 26
5.5 Several buses ... 26
5.6 Bridging buses ... 26
5.7 Contributing to the library ... 26

6 Configuration ... 28
6.1 Build-time parameters ... 28
6.2 Derived sizes ... 28
6.3 Memory footprint ... 29
6.4 Code size ... 29
6.5 Stack usage ... 29
6.6 CMake options ... 30

7 Baton bus protocol ... 31
7.1 The bus ... 31
7.2 Frame ... 32
7.3 Parser behavior ... 34
7.4 The slot ... 35
7.5 Delivery rules ... 35
7.6 Unreachable nodes ... 36
7.7 Time budget ... 36
7.8 Reference frames ... 37

8 Integration Guide ... 38
8.1 The physical layer ... 38
8.2 Bare metal ... 38
8.3 RTOS ... 39
8.4 Adding the library to a CMake build ... 39
8.5 Host ... 39
8.6 Hardware bring-up ... 40

9 Examples ... 41
9.1 Bare metal ... 41
9.2 Host serial master ... 42
9.3 Zephyr ... 43

10 Baton Inspector ... 44
10.1 Modes ... 44
10.2 Installation ... 45
10.3 The window ... 45
10.4 Bus definition ... 48
10.5 Timing rules ... 49
10.6 Python API ... 49
10.7 Session log (JSON Lines) ... 51
10.8 Tests of the tool ... 51

11 API Reference ... 52
11.1 Headers ... 52
11.2 Version ... 52
11.3 Configuration macros ... 52

11.4 Protocol constants (baton_defs.h) 52
11.5 Node API 53
11.6 Codec (baton_protocol.h) 57
11.7 Link (baton_link.h) 57
11.8 Event queue (baton_queue.h) 58

12 MISRA C:2012 compliance 59
12.1 Checking 59
12.2 Current status 59
12.3 Manual review notes 60
12.4 Deviation register 60

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

Appendix A Changelog 65
A.1 Unreleased 65
A.2 1.0.0 - 2026-10-25 65

Appendix B License 67
B.1 What the license allows 67
B.2 What it requires 67
B.3 What it does not give 67
B.4 Trademarks 68
B.5 Full license text 68

CHAPTER 1

Introduction

Baton-C is a C11 library for deterministic master/slave messaging between a few microcontrollers on one shared UART line (single-wire half duplex, open-drain or RS-485, 9600 bit/s by default). The master passes a token — the *baton* — to each slave in turn; only the node that holds it transmits. On top of this collision-free schedule the library delivers application messages of two kinds: **events**, delivered exactly once and in order, and **states**, of which the latest value always arrives. It is written for embedded systems that must be predictable and reviewable: no heap, no RTOS, no clock reads, MISRA C:2012 without violations and 100 % line and branch test coverage.

1.1 What the library does

- **Schedules the bus.** The master polls the slaves in a fixed order. A polled slave may send DATA frames to other slaves and must end its slot with a RESP within `slot_time_ms`. The worst-case cycle depends only on the configuration, not on the traffic.
- **Delivers events reliably.** Events are queued (HIGH and NORMAL priority), packed several per frame, acknowledged, retransmitted with the same sequence number, filtered for duplicates, and dropped after `event_ttl_ms` with a report of the nodes that did not acknowledge them.
- **Delivers states.** A state is a mailbox: publishing overwrites it and marks it for every other node. It cannot overflow; intermediate values may be skipped.
- **Handles an unreliable bus.** CRC-16 per frame, inter-byte timeout, echo check of the own transmission, retries, and a schedule that keeps unreachable nodes from eating the time of the healthy ones.
- **Encodes and parses frames.** The codec can be used on its own, for example in a bus monitor.

1.2 What the library does not do

Concern	Why it is not in the library
UART driver, pin setup, transceiver direction	Hardware-specific; you pass received bytes in and provide a non-blocking write function.
Threads, interrupt priorities, locking	Every system has its own; the library offers two optional critical-section hooks. See the integration guide.
A clock	Every call that needs the time takes `now_ms`; the library never reads a timer.
Message meaning, data layouts, units	Application data; the library transports byte strings up to their `MAX_LEN`.
Arbitration between several masters	One master per bus is the design.
Broadcast frames	A broadcast event is sent to each node in its own frame and acknowledged by each one (see design decisions).

1.3 Who it is for

- **Small multi-MCU products** — a control board and a few sensor, actuator or display boards connected by one wire, where CAN is too expensive and a plain UART protocol has no delivery guarantee.
- **Safety- or certification-minded firmware** that needs bounded execution time, static memory and a documented MISRA C:2012 status.
- **Host tools** — a PC or a Linux board can be the master or a bus monitor through a USB-UART adapter (see the examples (p. 41)).

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, no global variables; all state in a caller-owned `struct baton_node`
RAM per node	1696 B with the configuration template (8 nodes, 64-byte payload, queues 256 + 512 B), details
Code size	6620 B on Cortex-M4 with `-Os` (GCC 15.2), without `memcpy` / `memset` / `memmove`
Stack	280 B worst case in `baton_node_process()` on Cortex-M4 `-Os`, details
Execution time	Bounded by the configured sizes; no function blocks or waits
Coding standard	MISRA C:2012, 0 Cppcheck addon violations, 4 deviations
Tests	100 % line and branch coverage; bus simulation with noise and dead nodes; C/Python cross-check
Thread safety	One processing context plus interrupts; see Threading
License	Apache License 2.0

1.5 Terminology

Master
: The node that polls the slaves; node index 0. Exactly one per bus.

Slave
: A node that transmits only while it holds the baton; node index 1..n in the order of `slave_addresses`, which must be identical on every node.

Slot
: The time between a POLL to a slave and its RESP, at most `slot_time_ms`.

Cycle
: One round of POLLs over all slaves.

Frame
: One unit on the wire: `SYNC DST SRC CTRL LEN PAYLOAD CRC`. Types: POLL, RESP, DATA, ACK.

Message

One application record (id, length, data) inside a frame payload (TLV). Several messages share one frame and one acknowledgement.

Event

A message class: queued, acknowledged, delivered exactly once, in order.

State

A message class: a mailbox whose latest value is delivered to every other node.

Channel

The stop-and-wait state between two nodes in one direction: a 4-bit sequence number, the FIRST flag and the list of messages in the unacknowledged frame.

Node mask

A 32-bit set of node indexes; bit 0 is the master, bit i is `slave_addresses[i - 1]`.

1.6 Conventions in this documentation

- Hexadecimal bytes without prefix inside frames (`A5 11 01`), with `0x` in text and code.
- The CRC is big endian on the wire; node masks inside a RESP are little endian.
- Sizes are shown with the values of the configuration template (`config/baton_user_config_template.h`), timing with the defaults of `baton_node_default_config()` at 9600 bit/s.

CHAPTER 2

Getting Started

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

2.1 Get the library

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

The usual way to use it is as a Git submodule of the firmware project:

```
git submodule add https://github.com/AntonVynohradov/Baton-C.git lib/Baton-C
```

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

2.2 Add it to your project

1. Configuration. Copy `config/baton_user_config_template.h` into your project as `baton_user_config.h`, define your message lists and, if needed, the sizes. Every node of a bus must use the same message lists. Do not edit the template inside the submodule.

2a. CMake.

```
set(BATON_CONFIG_DIR ${CMAKE_CURRENT_SOURCE_DIR}/config)   # folder with
baton_user_config.h
include(lib/Baton-C/baton.cmake)
target_link_libraries(app PRIVATE baton)
```

2b. Any other build system (vendor IDE, Make). Compile `src/*.c` with C11 and add the repository root, `inc/` and the folder of your `baton_user_config.h` to the include path of the library and of the application.

3. Code. Include `baton.h`, fill a `struct baton_node_config` and follow Using the Library (p. 19).

The prebuilt package of a release is compiled with the configuration template; it is useful for host tools with the same message lists. Firmware normally builds the sources with its own configuration.

2.3 Prerequisites for development

- C11 compiler (GCC or Clang), CMake 3.20+, Make
- Unit tests: GoogleTest and gcovr
- Integration tests: Python 3 with pytest, and `gcc` on the path for the cross-check
- Analysis: Cppcheck (with its `misra.py` addon) and optionally the CodeQL CLI
- Formatting: clang-format

- Documentation: the packages of `requirements-dev.txt` (MkDocs Material, mkdoxy, WeasyPrint), Doxygen, and Pango for WeasyPrint

2.4 Build with presets

Preset	Purpose
`BATON[Debug]`	Static library with debug information (configuration template)
`BATON[Release]`	Optimized static library
`BATON[UnitTests]`	GoogleTest executable with coverage, bus simulation runner; target `coverage`
`BATON[IntegrationTests]`	pytest suite; target `integration_tests`
`BATON[Examples]`	Host examples and the bare-metal simulation test
`BATON[Sanitize]`	Like `UnitTests`, the simulation runner built with ASan and UBSan (Linux)

```
cmake --preset "BATON[Debug]" && cmake --build "build/BATON[Debug]"
```

CMake option	Default	Effect
`BATON_CONFIG_DIR`	—	Folder with `baton_user_config.h` (required when included by an application)
`BATON_UNIT_TESTS`	`OFF`	GoogleTest executable, `coverage` target, `baton_sim`
`BATON_INTEGRATION_TESTS`	`OFF`	pytest through CTest, `integration_tests` target
`BATON_EXAMPLES`	`OFF`	Host examples
`BATON_SANITIZERS`	`OFF`	ASan/UBSan for `baton_sim`
`BATON_COVERAGE_MIN`	`100`	Minimum line and branch coverage of `coverage`
`BATON_MISRA_FAIL_ON_VIOLATION`	`ON`	`misra-analyze` fails on a violation
`BATON_MISRA_RULE_TEXTS`	—	Optional MISRA rule texts file for readable messages

2.5 Unit tests and coverage

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

The 132 GoogleTest tests cover the codec, the link, the queue and both state machines, plus the bus simulation: several nodes on a simulated single-wire bus with the real byte time of 9600 bit/s, bit errors, a dead node and an RS-485 variant without echo. Each scenario checks exactly-once in-order delivery, final state delivery, no collisions and the slot length. The `coverage` target fails below `BATON_COVERAGE_MIN` (100 %) line or branch coverage of `src/`; the HTML report is in `build/BATON[UnitTests]/Coverage/`.

The same simulation runs optimized for many seeds:

```
cmake --build "build/BATON[UnitTests]" --target baton_sim
tests/run_seeds.sh "build/BATON[UnitTests]" 100            # Windows: tests\run_seeds.ps1
```

2.6 Integration tests

```
cmake --preset "BATON[IntegrationTests]"
cmake --build "build/BATON[IntegrationTests]" --target integration_tests
```

`tests/integration/baton/protocol.py` is an independent Python implementation of the codec using only the standard library (the *Python twin*). The tests check the reference frames of the protocol chapter (p. 37), and compile `src/baton_protocol.c` with `gcc` to compare 300 random frames between C and Python — encoding, parsing, and the reaction to a corrupted byte. Without `gcc` the cross-check is skipped and pytest says so. When CMake finds a Python without pytest, pass `-DPython3_EXECUTABLE=<python with pytest>`.

2.7 Static analysis

Target	Tool	Gate
`misra-analyze`	Cppcheck MISRA C:2012 addon (`cmake/misra_check.cmake`)	Fails on any violation
`cppcheck-analyze`	Cppcheck, all checks, `cppcheck-excludes.txt`	Report in `cppcheck.txt`
`codeql-analyze`	CodeQL `cpp-queries`	SARIF report

```
cmake --build "build/BATON[Debug]" --target misra-analyze
```

The analysis uses `tests/config/baton_user_config.h`. See MISRA C:2012 (p. 59) for the status and the deviation register.

2.8 Formatting

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

`.clang-format` is LLVM-based: 4 spaces, Allman braces, 100 columns, pointer to the type. The *Format Check* workflow fails a pull request that changes under clang-format.

2.9 Versioning and releases

The version comes from the latest `vX.Y.Z` Git tag (`cmake/version.cmake`); without a tag, `BATON_DEFAULT_VERSION` is used. `baton.h` has the same fallbacks for builds without CMake, and a unit test checks that `BATON_VERSION` matches the numbers. A release:

1. Set `BATON_DEFAULT_VERSION` and the fallbacks in `baton.h` to X.Y.Z.
2. Move the `[Unreleased]` entries of `CHANGELOG.md` to `## [X.Y.Z] - YYYY-MM-DD`.
3. Merge to `main`; the maintainer tags and pushes `vX.Y.Z`.

The *Release* workflow checks that the tag, `baton.h`, `cmake/version.cmake` and the changelog agree, runs every gate (coverage, integration, MISRA, examples), builds the package, the MISRA report and the PDF manual, and publishes the GitHub release.

2.10 Documentation

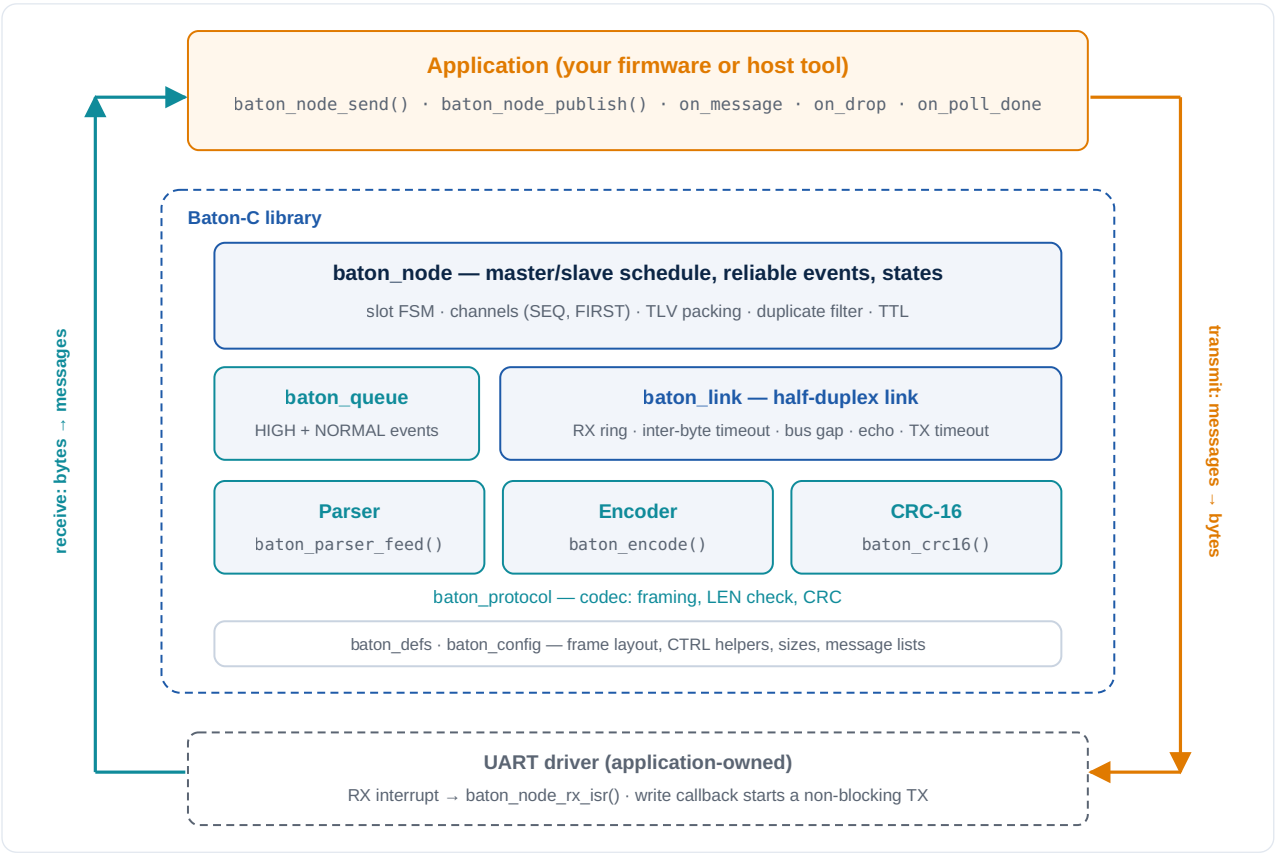
Task	Command	VS Code task
Live preview	`mkdocs serve`	*Docs: Serve on localhost*
Build the website	`mkdocs build`	*Docs: Build site*
Build the PDF manual	`python tools/docs/build_manual.py`	*Docs: Build PDF manual*
Regenerate diagrams	`python tools/docs/diagrams.py`	*Docs: Generate diagrams*
Re-measure the stack	`python tools/docs/stack_usage.py <dir>`	—

CHAPTER 3

How It Works

This chapter explains how Baton-C is built: the layers and what each one owns, how bytes travel through them, how events and states are stored and packed, the state machines, the memory model, error handling, threading, and the design decisions behind them.

3.1 Layers

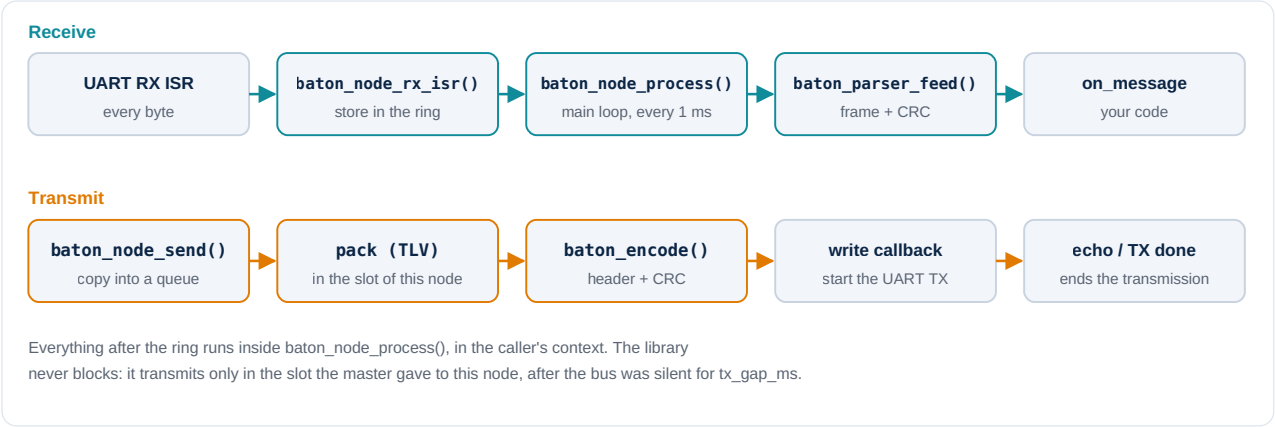


Layer	Files	Contents
Configuration	`inc/baton_config.h`, application `baton_user_config.h`	Sizes (each `#ifndef`-guarded and range-checked with `#error`) and the two message lists
Definitions	`inc/baton_defs.h`	Frame layout, frame types, ACKB bits, `static inline` CTRL and byte-order helpers. No state
Codec	`inc/baton_protocol.h`, `src/baton_protocol.c`	CRC-16, the incremental parser, the encoder. Pure functions over caller buffers
Link	`inc/baton_link.h`, `src/baton_link.c`	Receive ring (interrupt producer), parser with inter-byte timeout, transmitter: bus gap, echo check, timeout. Reports events to the node
Event queue	`inc/baton_queue.h`, `src/baton_queue.c`	Ring of variable-size records with a serialized 16-byte header

Layer	Files	Contents
Node	`inc/baton_node.h`, `src/baton_node.c`	Master and slave state machines, channels, TLV packing, duplicate filter, TTL, the application API
Umbrella	`baton.h`	Includes everything; version fallbacks

Each layer depends only on the ones below it. The application normally uses only the eight `baton_node_*` functions; the codec is public so that tools (a bus monitor, a test generator) can use it alone.

3.2 Data flow



Receive. The UART interrupt calls `baton_node_rx_isr()`, which only writes the byte into the ring of the link (single producer, single consumer, no lock). In `baton_node_process()` the link drains the ring: bytes of the node's own transmission are compared with what was sent (echo), every other byte goes to the parser. A complete valid frame is reported to the node, which dispatches it by type, filters retransmissions and calls `on_message` once per message.

Transmit. `baton_node_send()` copies an event into a queue and returns. When the node may transmit — a slave in its slot, the master at the start of a slot — it packs messages for one peer into the payload area of its transmit buffer, `baton_encode()` adds the header and the CRC in place, and the link calls the write function once the bus has been silent for `tx_gap_ms`. The transmission ends with the last echo byte (`rx_echo`) or `baton_node_tx_done_isr()`.

3.3 The parser

The parser is described byte by byte in Protocol → Parser behavior (p. 34). It decodes into its own `struct baton_frame`, so no second frame buffer exists, and it computes the CRC while the bytes arrive. The link adds the inter-byte timeout: when more than `inter_byte_timeout_ms` pass inside a frame, the parser is reset and `rx_timeouts` counts it.

3.4 The encoder

`baton_encode()` writes SYNC, the header, the payload and the CRC into a caller buffer after checking the capacity. When the payload already sits at `&output[BATON_HEADER_SIZE]` — the node packs it there — nothing is copied.

3.5 Event queues

There are two queues, HIGH and NORMAL. Each is a byte array inside `struct baton_node` holding variable-size records: a 16-byte header — pending mask, inflight mask, time stamp, id, length, destination, flags, serialized byte by byte (no pointer casts) — and the data, padded to 4 bytes.

- **Copy in.** `baton_node_send()` copies the data and returns at once; the caller keeps no buffer.
- **No split records.** A record that does not fit at the end of the array goes to the start; a SKIP marker is left at the end if a header fits there. Data is always contiguous.
- **pending** is the node mask still to acknowledge the event; a broadcast sets every other node. **inflight** is the node mask with an unacknowledged frame containing it; it is always a subset of pending.
- **Release from the head only**, when pending is empty. References held by channels therefore stay valid.
- **Peek and commit.** Packing does not remove a record; the acknowledgement does, and a retransmission needs no copy.
- **TTL.** Every 10 ms records older than `event_ttl_ms` are removed and reported through `on_drop`.

An unreachable destination can hold a record at the head of a queue for at most `event_ttl_ms`; size the queues for the traffic of that time.

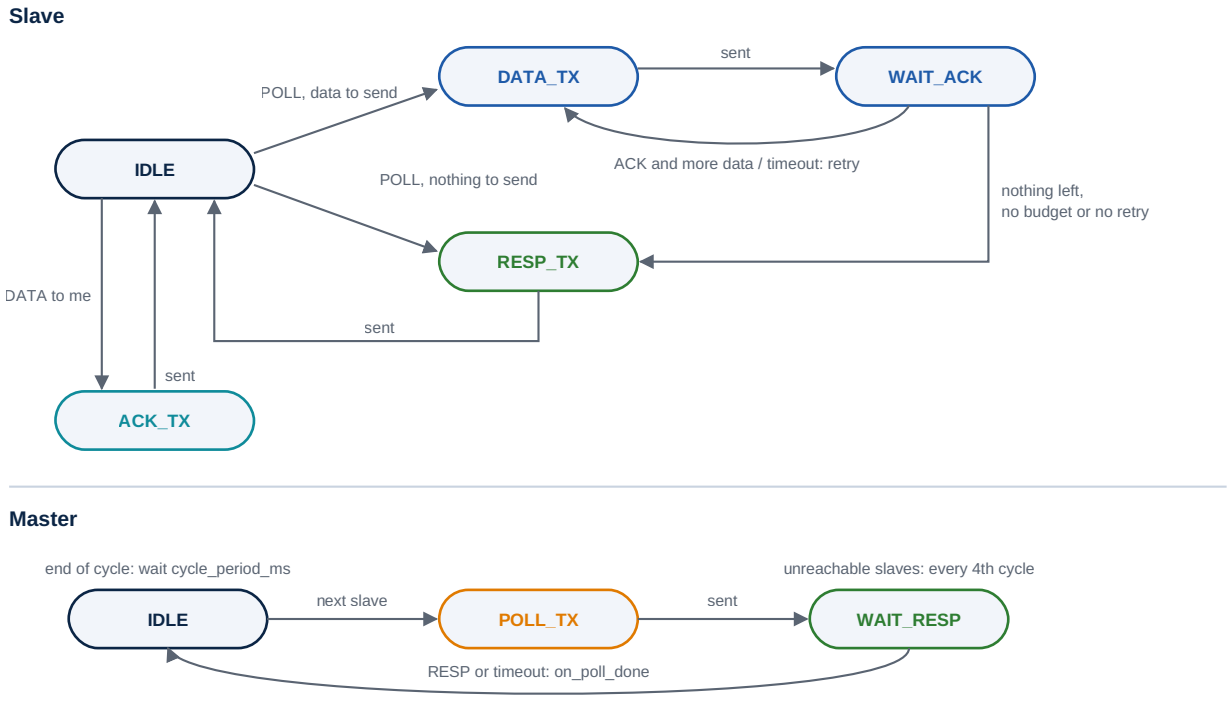
3.6 State mailboxes

Each state has a buffer of its `MAX_LEN`, a current length and a dirty node mask. `baton_node_publish()` overwrites the value and marks every other node — it never fails for lack of space. Packing copies the *current* value and clears the destination's bit; if the frame is not acknowledged, the retransmission carries the latest value again. Intermediate values may be skipped; the last one always arrives.

3.7 Packing a frame

1. If the channel to the peer has an unacknowledged frame, the same messages are packed again with the same SEQ. A state that grew and no longer fits is removed from it and sent later.
2. Otherwise the frame is filled with the peer's HIGH events in FIFO order up to the first that does not fit, then NORMAL events the same way, then its dirty states.
3. The limits are the remaining slot budget (Protocol → Time budget (p. 36)), the room of the frame type and `BATON_MAX_MSGS_PER_FRAME`.

3.8 State machines



The slave serves its peers in two passes per slot: first the reachable ones in round-robin order, then the ones with three or more failures in a row, once each and without retry. A peer that failed in the slot is in the NACK mask and is not served again in that slot.

The transmitter of the link has three states: idle, waiting for a silent bus, and busy. Busy ends with the full echo (`rx_echo`) or the TX-done interrupt; a timeout of `F(length) + tx_gap_ms + 5` ms is the safety net (`tx_timeouts`). An echo mismatch is counted in `collisions`.

3.9 Memory model

All state lives in one caller-owned `struct baton_node`; the library has no global variables (only `static const` tables). Its size is fixed at build time by `baton_user_config.h` — see Configuration → Memory footprint (p. 29). No function allocates memory, recurses or uses variable-length arrays, so the stack is fixed: see Configuration → Stack usage (p. 29).

3.10 Error handling

Situation	What happens	Visible as
Invalid argument	The function returns an error code and changes nothing	`BATON_ERR_*`, `BATON_PARSE_ERROR`
Inconsistent configuration	`baton_node_init()` fails; the node stays uninitialized	`BATON_ERR_PARAM`
Event queue full	`baton_node_send()` fails; nothing is queued	`BATON_ERR_FULL`, `queue_full`

Situation	What happens	Visible as
CRC mismatch, LEN too large	Frame dropped	`crc_errors`
Gap inside a frame	Frame dropped, parser reset	`rx_timeouts`
Receive ring full	Byte dropped	`rx_overflows`
Echo differs from the sent bytes	Transmission completes, counted	`collisions`
Write fails or no end of transmission	Treated as sent, counted	`tx_timeouts`
No ACK	Retry, then NACK in the RESP	`ack_timeouts`, `retries`
No RESP	Next slave	`resp_timeouts`, `on_poll_done(responded = false)`
Retransmitted content	Acknowledged, not delivered	`duplicates`
Unknown or too long message	Skipped	`bad_messages`
Event older than `event_ttl_ms`	Removed	`events_dropped`, `on_drop`

3.11 Threading and reentrancy

Function	Context
`baton_node_process()`, `baton_node_init()`, `baton_node_get_stats()`	One context: the main loop or one RTOS task
`baton_node_rx_isr()`	UART receive interrupt; writes only the ring
`baton_node_tx_done_isr()`	UART TX-complete interrupt; sets a flag
`baton_node_send()`, `baton_node_publish()`	Any context **if** `enter_critical`/`exit_critical` are set; otherwise the context of `baton_node_process()`
`on_message`, `on_drop`, `on_poll_done`	Always inside `baton_node_process()`; they may call `baton_node_send()` and `baton_node_publish()`

Critical sections are a few instructions long: reserving queue space, moving head and tail, copying a state value. Event data is copied outside them. The codec functions are reentrant on different structures. Several buses need several `struct baton_node` instances; they do not share anything.

3.12 Design decisions

Decision	Reason
POLL as a token (the baton)	Determinism without arbitration: no collisions, a bounded cycle computed from the configuration
Two message classes	Events need exactly-once, in-order delivery; states need "latest value wins" and must never overflow

Decision	Reason
Several messages per frame (TLV), one ACK per frame	At 9600 bit/s the 7-byte overhead and the turnaround dominate; packing removes most of it
Traffic with the master has no separate ACK	Slave → master content is acknowledged by the ACKB of the next POLL, master → slave content by the ACKB of the RESP in the same slot
Stop-and-wait per channel, 4-bit SEQ and FIRST	Simple, exactly-once with one outstanding frame; FIRST survives a restart of the sender
Retransmission rebuilt from queue references	No per-frame copy in RAM
Only RESP frames with content are acknowledged	An empty RESP shared the SEQ of the next one and confirmed it in advance; the bus simulation found this, the fix is permanent
Unreachable nodes served last, polled every 4th cycle	A dead node cannot eat the slot or the cycle of the healthy ones
`on_drop` reports the nodes that did not acknowledge	No silent loss: the application knows what may be missing
Explicit `now_ms`, no clock reads	Portable, testable with simulated time
No broadcast frame with slotted ACKs	It would be more efficient with many receivers but much more complex; kept for a future major version

CHAPTER 4

Using the Library

This chapter shows how to set up a node, write a slave and a master, send events and publish states, read the diagnostics, use the codec on its own, and avoid the common mistakes. The complete programs are in the examples (p. 41).

4.1 Define the messages

Every node of a bus is built with the same `baton_user_config.h`. Each message is `X(NAME, ID, MAX_LEN)`; the library generates `BATON_MSG_<NAME>` (the id) and `BATON_MSG_<NAME>_MAX_LEN`:

```
#define
BATON_STATE_LIST(X)
\
    X(NODE_STATUS, 0x01,
4)                                                          \
    X(TEMPERATURE, 0x02, 2)

#define
BATON_EVENT_LIST(X)
\
    X(CMD_START, 0x10,
8)                                                            \
    X(CMD_STOP, 0x11,
1)                                                             \
    X(ALARM, 0x12, 16)
```

Rule	Checked
IDs 0..255, unique across both lists	At compile time (duplicate `case` value, `_Static_assert`)
`MAX_LEN` 1.. `BATON_MAX_MSG_LEN`	At compile time
Each event fits into both queues	At compile time
Same lists on every node	Not checkable — a mismatch shows as `bad_messages`

Use a **state** for a value whose latest version matters (a measurement, a mode, a heartbeat); use an **event** for something that must happen exactly once (a command, an alarm, a button press).

4.2 Set up a node

```
static const uint8_t slaves[] = {0x11U, 0x12U, 0x13U}; /* polling order, same on every
node */
static struct baton_node node;

int start_node(void)
{
    struct baton_node_config config;

    baton_node_default_config(&config);
    config.role = BATON_ROLE_SLAVE;
    config.own_address = 0x12U;
```

```
    config.master_address = 0x01U;
    config.slave_addresses = slaves;
    config.slave_count = 3U;
    config.write = uart_write;
    config.on_message = on_message;
    return baton_node_init(&node, &config, board_millis());
}
```

Field	Default	Meaning
`role`	`BATON_ROLE_SLAVE`	`BATON_ROLE_MASTER` or `BATON_ROLE_SLAVE`
`own_address`, `master_address`	—	The master uses `own_address == master_address`
`slave_addresses`, `slave_count`	—	Slaves in polling order, 1.. `BATON_MAX_NODES` − 1; the array must stay valid
`baudrate`	9600	Used for frame durations and the slot budget
`slot_time_ms`	250	Longest slot; at least the minimum checked by init
`ack_timeout_ms`	20	Wait for an ACK after DATA
`max_retries`	1	DATA retransmissions per slot
`tx_gap_ms`	2	Silence before transmitting
`inter_byte_timeout_ms`	5	Gap that aborts a frame
`cycle_period_ms`	0	Master: shortest time between cycle starts
`event_ttl_ms`	5000	Undelivered events are dropped after this; 0 = never
`rx_echo`	`true`	The UART receives its own bytes (single wire); `false` for RS-485 without echo
`write`	—	Starts a non-blocking transmission (required)
`enter_critical`, `exit_critical`	`NULL`	Both or neither; needed to call send/publish from other contexts
`on_message`, `on_drop`, `on_poll_done`	`NULL`	Callbacks, called from `baton_node_process()`
`context`	`NULL`	Passed to every hook and callback

`baton_node_init()` returns `BATON_ERR_PARAM` for an inconsistent configuration: a missing write function, a broadcast or duplicate address, `own_address` not on the bus or not matching the role, a slot shorter than one maximum exchange, an ACK timeout not longer than an ACK frame plus the gap. It can be called again to restart the node.

4.3 The write function

```
static int uart_write(void* context, const uint8_t* data, size_t length)
{
    (void) context;
    return board_uart_write(data, length); /* 0 = started, negative = failed */
}
```

The function starts the transmission and returns; the bytes go out from the TX interrupt or a DMA channel. The buffer stays valid until the transmission ends: with `rx_echo` the library sees its last byte come back; otherwise call `baton_node_tx_done_isr()` from the TX-complete interrupt. A blocking write also works (see the host example (p. 42)), but it delays the processing of received bytes by the duration of the frame.

4.4 Run the node

```c
void uart_rx_irq_handler(uint8_t byte)
{
    baton_node_rx_isr(&node, byte);
}

void main_loop(void)
{
    for (;;)
    {
        baton_node_process(&node, board_millis());
        board_wait_for_tick(); /* about 1 ms */
    }
}
```

Call `baton_node_process()` at least every millisecond; it never blocks. All timing — gaps, timeouts, the slot budget, TTL — is measured with the `now_ms` you pass, a monotonic millisecond counter that may wrap.

4.5 Write a slave

A slave reacts to the messages it receives and publishes its own:

```c
static void on_message(void* context, uint8_t src, uint8_t msg_id, const uint8_t* data,
                       uint8_t length)
{
    (void) context;
    (void) src;
    if ((msg_id == BATON_MSG_CMD_STOP) && (length == 1U))
    {
        motor_stop(data[0]);
    }
}

void report_temperature(int16_t centi_degrees)
{
    uint8_t value[2];

    value[0] = (uint8_t) ((uint16_t) centi_degrees & 0xFFU);
    value[1] = (uint8_t) ((uint16_t) centi_degrees >> 8);
    (void) baton_node_publish(&node, BATON_MSG_TEMPERATURE, value, 2U);
}
```

Rules of a well-behaved slave:

- Keep `on_message` short; it runs inside `baton_node_process()`. Queue longer work.
- Encode data with an explicit byte order (as above), not by copying structs: every node must agree on the layout.
- Expect a message more than once only if it is a state (a new value) — events arrive exactly once.

4.6 Write a master

The master runs the schedule by itself; its application sends and receives like a slave and may watch the slots:

```c
static void on_poll_done(void* context, uint8_t slave, bool responded, uint32_t
nack_nodes)
{
    (void) context;
    if (!responded)
    {
        mark_offline(slave); /* no RESP in this slot */
    }
    if (nack_nodes != 0U)
    {
        log_unreachable(slave, nack_nodes); /* the slave could not reach these nodes */
    }
}

int command_start(uint8_t slave, uint16_t speed)
{
    uint8_t data[3];

    data[0] = (uint8_t) (speed & 0xFFU);
    data[1] = (uint8_t) (speed >> 8);
    data[2] = 1U; /* mode */
    return baton_node_send(&node, slave, BATON_MSG_CMD_START, data, 3U, BATON_PRIO_HIGH);
}
```

Set `config.role = BATON_ROLE_MASTER`, `own_address = master_address`, and `on_poll_done`. `cycle_period_ms` spaces the cycles when the bus should not be busy all the time.

4.7 Events and states

	`baton_node_send()` (event)	`baton_node_publish()` (state)
Destination	One node or `BATON_ADDR_BROADCAST`	Every other node
Storage	Copied into the HIGH or NORMAL queue	Overwrites the mailbox
Can fail for lack of space	Yes: `BATON_ERR_FULL`	Never
Delivery	Exactly once, in order per source, destination and priority	The latest value; intermediate ones may be skipped
Undeliverable	Dropped after `event_ttl_ms`, `on_drop`	Sent again until acknowledged

When `baton_node_send()` returns `BATON_ERR_FULL`, keep the event and try again later, or drop it deliberately — the library never drops silently.

4.8 Return values

Code	Value	Returned when
`BATON_OK`	0	Success
`BATON_ERR_PARAM`	−1	NULL data with a non-zero length, invalid configuration

Code	Value	Returned when
`BATON_ERR_NOT_INIT`	−2	The node is NULL or not initialized
`BATON_ERR_MSG_ID`	−3	Unknown id, or a state id passed to send / an event id to publish
`BATON_ERR_LEN`	−4	Length above the `MAX_LEN` of the message
`BATON_ERR_ADDR`	−5	Unknown destination, or the node itself
`BATON_ERR_FULL`	−6	The event queue has no room

4.9 Diagnostics

`baton_node_get_stats()` copies the counters (see Error handling (p. 16)):

```c
void print_link_health(void)
{
    struct baton_stats stats;

    baton_node_get_stats(&node, &stats);
    log_value("crc_errors", stats.crc_errors);     /* noise on the line */
    log_value("ack_timeouts", stats.ack_timeouts); /* a peer does not answer */
    log_value("queue_used", stats.queue_normal_used);
}
```

Rising `crc_errors` and `rx_timeouts` point to the physical layer; `collisions` should stay 0 — a non-zero value means two nodes transmitted at once (wrong topology on one node, two masters, or a node with another slave list).

4.10 The codec without the node

The codec works on its own, for example in a bus monitor that prints every frame:

```c
static struct baton_parser monitor;

void monitor_byte(uint8_t byte)
{
    if (baton_parser_feed(&monitor, byte) == 1)
    {
        const struct baton_frame* f = &monitor.frame;

        print_frame(f->src, f->dst, baton_ctrl_type(f->ctrl), baton_ctrl_seq(f->ctrl),
                    f->payload, f->length);
    }
}
```

Call `baton_parser_init(&monitor)` once before the first byte.

4.11 Common mistakes

Symptom	Cause
`baton_node_init()` returns `BATON_ERR_PARAM`	A field of the configuration, see Set up a node; often `slot_time_ms` too short for the baud rate and payload

Symptom	Cause
The master polls, slaves never answer	Different `slave_addresses` order or `master_address` on some node; the slave UART does not receive (wiring, idle level)
`tx_timeouts` grows	The write function does not start the UART, or `rx_echo` is `true` on a transceiver without echo
`collisions` grows	Two masters, or a node with another topology; on RS-485 the transceiver is switched to receive too late
`bad_messages` grows	The nodes are built with different message lists
`rx_overflows` grows	`baton_node_process()` is called too rarely for `BATON_RX_BUF_SIZE`
Events dropped (`on_drop`)	Destination offline, or `event_ttl_ms` shorter than a few cycles
`BATON_ERR_FULL`	Queues too small for the traffic during `event_ttl_ms`, or a destination offline

CHAPTER 5

Extending the Library

Almost everything an application needs is done on top of the library: new messages are configuration, new transports are a write function and an interrupt handler, data models are application code. This chapter says where each extension belongs and how to contribute changes to the library itself.

5.1 Where to extend

You want to…	Do it in	Library change needed?
Add a message	`BATON_STATE_LIST` / `BATON_EVENT_LIST` of `baton_user_config.h` on every node	No
Larger frames, more nodes, bigger queues	The sizes in `baton_user_config.h`	No
Support another UART, RS-485 or a USB adapter	The write function and the receive interrupt	No
Give message data a structure	Encode and decode functions in the application	No
Monitor a bus	The codec alone (`baton_parser_feed()`)	No
Change timing (slot, timeouts, baud rate)	`struct baton_node_config`	No
Change the wire format or the delivery rules	`src/`, the Python twin, the protocol chapter	Yes — a new major version

5.2 Adding a message

1. Add `X(NAME, ID, MAX_LEN)` to the state or event list with a new ID.
2. Rebuild **every node** of the bus with the new list. A node with an old list counts the new message in `bad_messages` and ignores it.
3. Write an encode and a decode function for its data with an explicit byte order.

Changing the `MAX_LEN` of a message is compatible as long as the senders never use more than the smallest `MAX_LEN` among the nodes.

5.3 Adding a transport

The write function contract:

Requirement	Why
Start the transmission and return 0, or return a negative value	The node never blocks; a failure is counted in `tx_timeouts` and treated as sent

Requirement	Why
Keep sending from the given buffer	The library keeps it unchanged until the end of the transmission
End: last echo byte (`rx_echo = true`) or `baton_node_tx_done_isr()`	The link uses it to start the next timer and to free the bus
Called only from `baton_node_process()`	It may use the same resources as the processing context

Received bytes of every node — including the own echo — go to `baton_node_rx_isr()`.

5.4 Data models

Keep the application data out of the library: define, per message, a C structure and two functions that convert it to and from bytes. The examples encode the uptime with `baton_put_u32le()` and decode it with `baton_get_u32le()` from `baton_defs.h`.

5.5 Several buses

Use one `struct baton_node` per bus. Nodes share nothing, but they share the message lists of the build; buses with different message lists need separate builds of the library (for example two CMake targets with different `BATON_CONFIG_DIR`).

5.6 Bridging buses

A gateway node is a member of two buses (two `struct baton_node`) and forwards in its `on_message` callbacks: an event received on bus A is sent with `baton_node_send()` on bus B. Delivery is then exactly once per hop; end-to-end confirmation, if needed, is an application message (like `OUTPUT_REPORT` in the examples).

5.7 Contributing to the library

1. **Code.** C11, no new dependencies, no heap, the MISRA rules used in the code (single exit, parenthesized expressions, no pointer arithmetic, `(void)` on ignored results).
2. **Tests.** Every behavior change needs a test; `coverage` must stay at 100 % lines and branches; the bus simulation must pass on 100 seeds (`tests/run_seeds.sh`). Update the Python twin when the codec changes.
3. **Analysis.** `misra-analyze` reports 0 violations, or the deviation is recorded in MISRA C:2012 (p. 59).
4. **Format.** `clang-format` reports no changes.
5. **Documentation.** The affected pages, the stack and size tables (Configuration (p. 28)), the diagrams (`tools/docs/diagrams.py`), and `CHANGELOG.md` under `[Unreleased]`.
6. **Compatibility.** The public API and the wire format stay stable within a major version.

Local definition of done:

```
cmake --preset "BATON[Debug]" && cmake --build "build/BATON[Debug]"
cmake --preset "BATON[UnitTests]" && cmake --build "build/BATON[UnitTests]" --target cover
```

```
age
cmake --build "build/BATON[UnitTests]" --target baton_sim && tests/run_seeds.sh "build/
BATON[UnitTests]" 100
cmake --preset "BATON[IntegrationTests]" && cmake --build "build/BATON[IntegrationTests]"
--target integration_tests
cmake --build "build/BATON[Debug]" --target misra-analyze
cmake --preset "BATON[Examples]" && cmake --build "build/BATON[Examples]" && ctest --test-
dir "build/BATON[Examples]" -L example
```

CHAPTER 6

Configuration

This chapter lists the build-time parameters with their ranges, the derived sizes, the measured memory footprint, code size and stack usage, how to measure them again, and the CMake options. The runtime parameters are described in Using the Library → Set up a node (p. 19).

6.1 Build-time parameters

The application defines them in `baton_user_config.h` (or with `-D` compiler options); `inc/baton_config.h` provides the defaults and rejects values out of range with `#error`.

Macro	Default	Range	Meaning
`BATON_MAX_NODES`	8	2..32	Nodes on the bus including the master
`BATON_MAX_PAYLOAD`	64	16..250	Largest frame payload in bytes
`BATON_MAX_MSGS_PER_FRAME`	16	1..64	Messages packed into one frame
`BATON_QUEUE_HIGH_SIZE`	256	64..65532	HIGH event queue in bytes
`BATON_QUEUE_NORMAL_SIZE`	512	64..65532	NORMAL event queue in bytes
`BATON_RX_BUF_SIZE`	128	16..4096, power of two	Receive ring filled by the interrupt
`BATON_STATE_LIST(X)`	—	required	STATE messages `X(NAME, ID, MAX_LEN)`
`BATON_EVENT_LIST(X)`	—	required	EVENT messages `X(NAME, ID, MAX_LEN)`

6.2 Derived sizes

Macro	Value	Meaning
`BATON_FRAME_MAX`	`7 + BATON_MAX_PAYLOAD`	Largest frame on the wire
`BATON_MASK_BYTES`	`(BATON_MAX_NODES + 7) / 8`	NACK mask in a RESP
`BATON_RESP_PREFIX`	`1 + BATON_MASK_BYTES`	ACKB and NACK
`BATON_MAX_MSG_LEN`	`BATON_MAX_PAYLOAD − BATON_RESP_PREFIX − 2`	Largest `MAX_LEN` of a message
`BATON_STATE_COUNT`, `BATON_EVENT_COUNT`	from the lists	Number of messages
`BATON_STATE_DATA_SIZE`	`1 + Σ MAX_LEN` of the states	State mailboxes

Sizing hints: an event occupies `16 + MAX(len rounded up to 4)` bytes of its queue until every destination acknowledged it, at most for `event_ttl_ms`. The receive ring must hold the bytes that arrive between two calls of `baton_node_process()` plus the echo of a maximum frame (`BATON_FRAME_MAX`).

6.3 Memory footprint

All memory of a node is the caller's `struct baton_node`; the library has no global or static variables except constant tables. Measured with `sizeof` for Arm Cortex-M4 (`arm-none-eabi-gcc` 15.2.1):

Configuration	`struct baton_node`	of which queues	of which link (ring, parser, counters)
Template (8 nodes, payload 64, 16 msgs, queues 256 + 512, ring 128)	1696 B	768 B	280 B
Unit tests (as template, ring 256)	1824 B	768 B	408 B
Examples (4 nodes, payload 64, 8 msgs, queues 128 + 256, ring 128)	1068 B	384 B	280 B

Approximately: `BATON_RX_BUF_SIZE + BATON_QUEUE_HIGH_SIZE + BATON_QUEUE_NORMAL_SIZE + 2 × BATON_FRAME_MAX + 7 + BATON_MAX_NODES × (2 × BATON_MAX_MSGS_PER_FRAME + 10) + state mailboxes + about 300 B` of configuration copy, counters and state variables.

6.4 Code size

Library code (`text`) for Cortex-M4 Thumb-2, compiled with the configuration template, `-ffunction-sections`, without the C library functions `memcpy`, `memset` and `memmove` that the library calls:

File	`-Os`	`-O0`
`baton_protocol.c`	932 B	1070 B
`baton_link.c`	602 B	1376 B
`baton_queue.c`	456 B	1194 B
`baton_node.c`	4630 B	9094 B
Total	**6620 B**	**12734 B**

The library has no initialized or zero-initialized data of its own (`data` and `bss` are 0).

6.5 Stack usage

Worst-case stack of the entry points, from GCC `-fstack-usage -fcallgraph-info=su`, following every direct call. Not included: the application functions the library calls through pointers (`write`, `enter_critical`, `exit_critical`, `on_message`, `on_drop`, `on_poll_done`) — add their own stack to the figure of the function that calls them — and `memcpy`/`memset`/`memmove`.

Function	Cortex-M4 `-Os`	Cortex-M4 `-O0`	x86-64 `-O2`
`baton_node_process()`	280 B	400 B	752 B
`baton_node_init()`	80 B	160 B	216 B
`baton_node_send()`	80 B	112 B	136 B
`baton_node_publish()`	24 B	64 B	80 B

Function	Cortex-M4 `-Os`	Cortex-M4 `-O0`	x86-64 `-O2`
`baton_node_get_stats()`	24 B	56 B	72 B
`baton_node_rx_isr()`	8 B	40 B	16 B
`baton_node_tx_done_isr()`	0 B	32 B	16 B
`baton_parser_feed()`	24 B	72 B	16 B
`baton_encode()`	48 B	80 B	80 B

Toolchains: Arm GNU Toolchain 15.2.Rel1 (`arm-none-eabi-gcc` 15.2.1, `-mcpu=cortex-m4 -mthumb`), MinGW-w64 GCC 14.2.0 (x86-64 Windows ABI, which reserves 32 bytes of shadow space per call).

Worst-case chains:

- `baton_node_process()` (`-Os`): process → slave slot step → DATA frame → pack → state copy → TLV emit, on the reception of a POLL.
- `baton_node_process()` (`-O0`): process → frame dispatch → POLL handler → slot step → RESP → pack → retransmission → state copy → TLV emit.

An interrupt that calls `baton_node_rx_isr()` adds 8 B (Cortex-M4 `-Os`) to the stack of whatever it interrupts.

How to measure again

```
mkdir -p build/stack
for f in src/*.c; do
  arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb -Os -std=c11 -fstack-usage -fcallgraph-
info=su \
    -I. -Iinc -I<folder of baton_user_config.h> -c "$f" -o "build/stack/$(basename
"$f" .c).o" \
    -dumpdir build/stack/
done
python tools/docs/stack_usage.py build/stack
arm-none-eabi-size build/stack/*.o
```

Measure again after any change in `src/` that may change a frame, and update this page.

6.6 CMake options

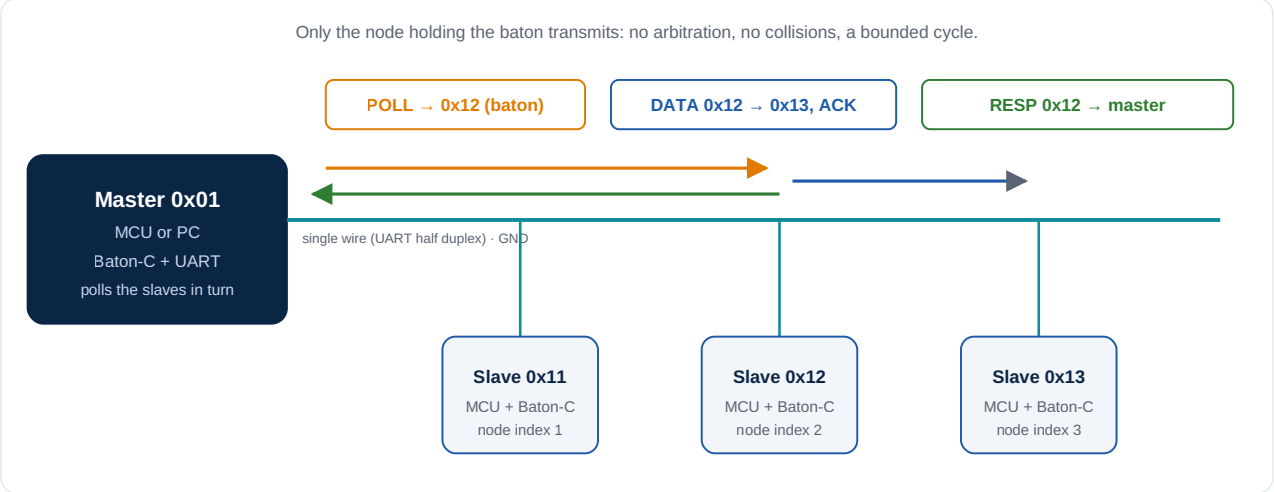
See Getting Started → Build with presets (p. 10).

CHAPTER 7

Baton bus protocol

This chapter specifies the Baton bus protocol — the bytes on the wire, the slot schedule and the delivery rules — and says exactly what Baton-C does with each part. It is the specification of the protocol: other implementations (for example the Python twin in `tests/integration/baton/protocol.py`) are checked against it.

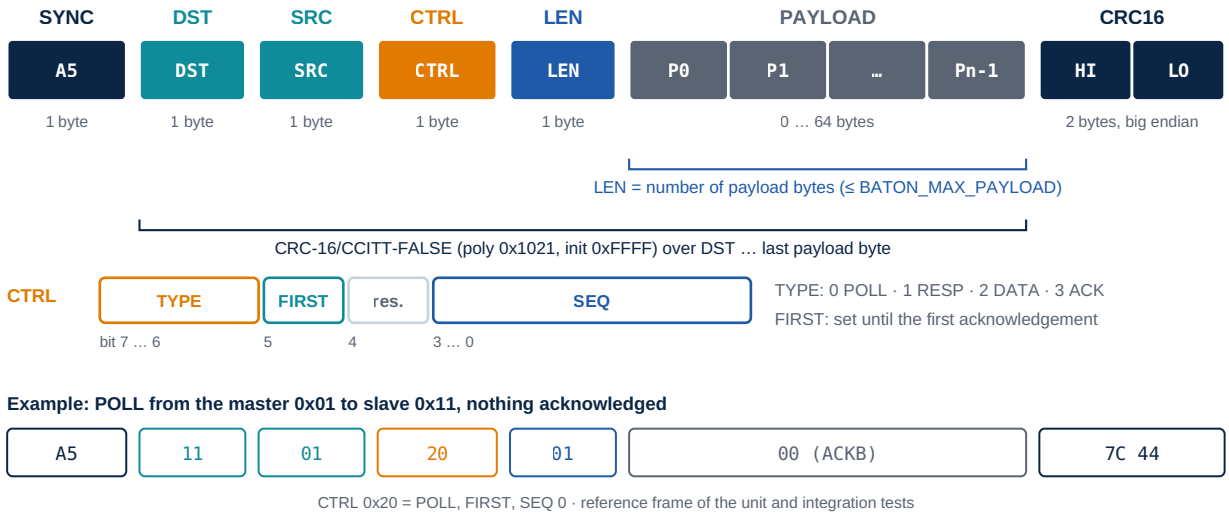
7.1 The bus



Property	Value
Medium	One shared UART line: single wire (TX and RX on one pin), open-drain, or RS-485
Character format	8N1: 10 bits per byte, 1.04 ms per byte at 9600 bit/s
Nodes	One master and 1..31 slaves (`BATON_MAX_NODES` 2..32), addresses `0x00`..`0xFE`
Topology	Identical on every node: master address and the slave list in polling order
Access	Only the polled slave transmits, for at most `slot_time_ms`; other slaves send only ACKs to DATA addressed to them
Turnaround	Before transmitting, a node waits for `tx_gap_ms` of silence on the line

The address `0xFF` (`BATON_ADDR_BROADCAST`) is reserved: it selects "every other node" in `baton_node_send()` and never appears in a frame.

7.2 Frame



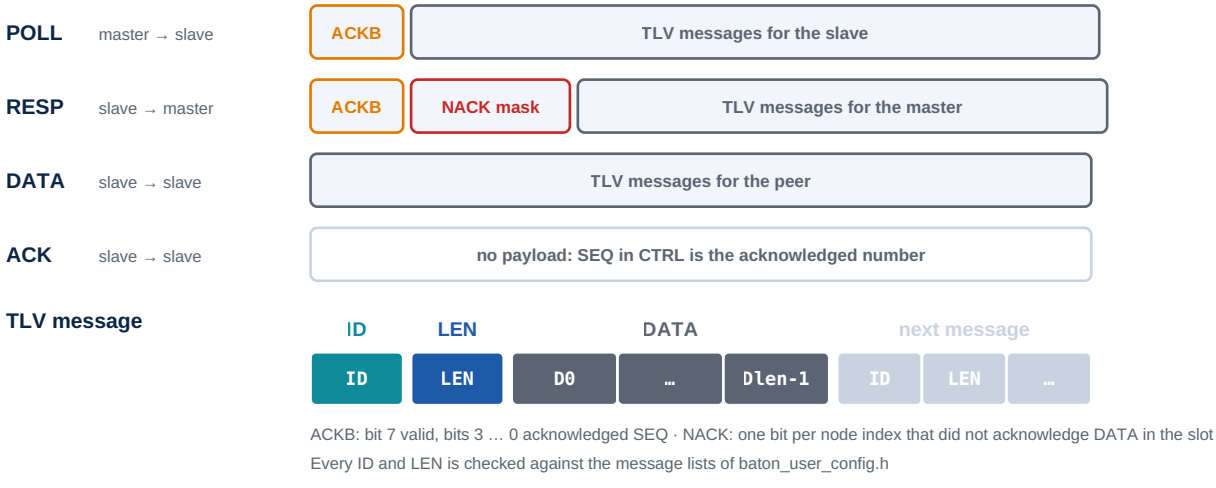
Offset	Field	Size	Description
0	SYNC	1	`0xA5`, start of a frame
1	DST	1	Destination address
2	SRC	1	Source address
3	CTRL	1	`TYPE[7:6]`, `FIRST[5]`, reserved `[4]` (0), `SEQ[3:0]`
4	LEN	1	Number of payload bytes, 0.. `BATON_MAX_PAYLOAD` (at most 250)
5	PAYLOAD	LEN	Depends on the frame type, see below
5 + LEN	CRC16	2	CRC-16/CCITT-FALSE over DST..PAYLOAD, high byte first

The overhead is 7 bytes. An ACK (no payload) takes 7 bytes, 8 ms at 9600 bit/s; a maximum frame with the default 64-byte payload takes 71 bytes, 74 ms.

Checksum

CRC-16/CCITT-FALSE: polynomial `0x1021`, initial value `0xFFFF`, no reflection, no final XOR. It covers DST, SRC, CTRL, LEN and the payload — not SYNC. The check value of the catalogue (`"123456789"`) is `0x29B1`; it is a unit test. Baton-C computes it bit by bit without a table (no 512-byte table in flash; at 9600 bit/s the cost does not matter).

Frame types and payloads



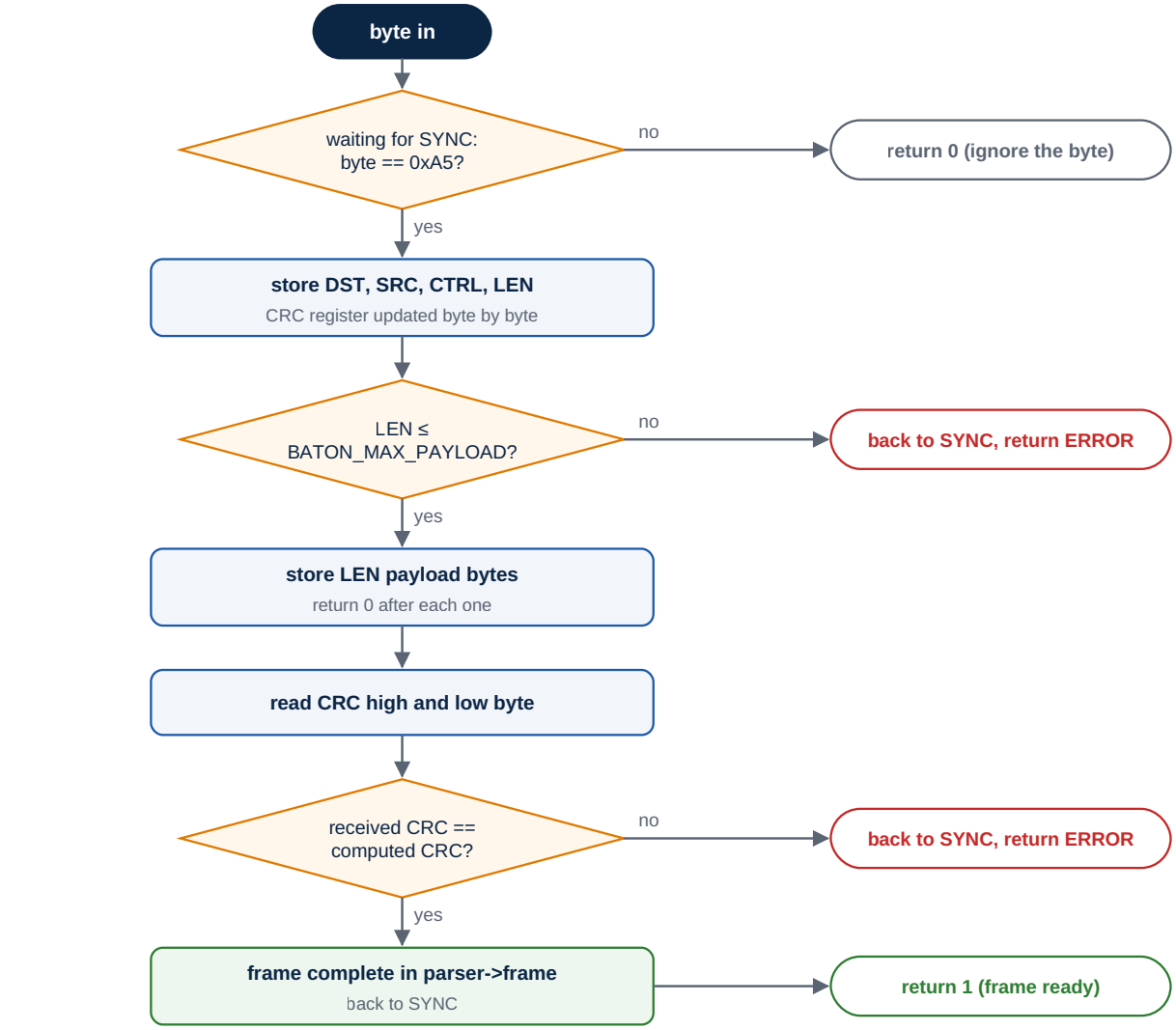
TYPE	Name	Direction	Payload
0	POLL	master → slave	`ACKB`, then messages for this slave
1	RESP	slave → master	`ACKB`, `NACK` mask, then messages for the master
2	DATA	slave → slave	Messages for the destination
3	ACK	slave → slave	None; `SEQ` in CTRL is the acknowledged sequence number

- **ACKB** (1 byte): bit 7 = valid, bits 3..0 = acknowledged sequence number. In a POLL it acknowledges the last RESP *with content* from this slave; in a RESP it acknowledges the POLL that started the slot.
- **NACK** (`(BATON_MAX_NODES + 7) / 8` bytes, little endian): the node mask of the slaves that did not acknowledge a DATA frame of this slave in this slot.

Messages (TLV)

A payload carries messages back to back, each as `ID`, `LEN`, `DATA[LEN]`. One frame carries up to `BATON_MAX_MSGS_PER_FRAME` messages. The receiver checks every `ID` and `LEN` against the message lists of `baton_user_config.h`; an unknown id or a length above the `MAX_LEN` of the message is skipped and counted in `bad_messages`, a truncated record ends the payload. The largest `MAX_LEN` is `BATON_MAX_MSG_LEN = BATON_MAX_PAYLOAD − 3 − BATON_MASK_BYTES`, so that one message fits into every frame type.

7.3 Parser behavior



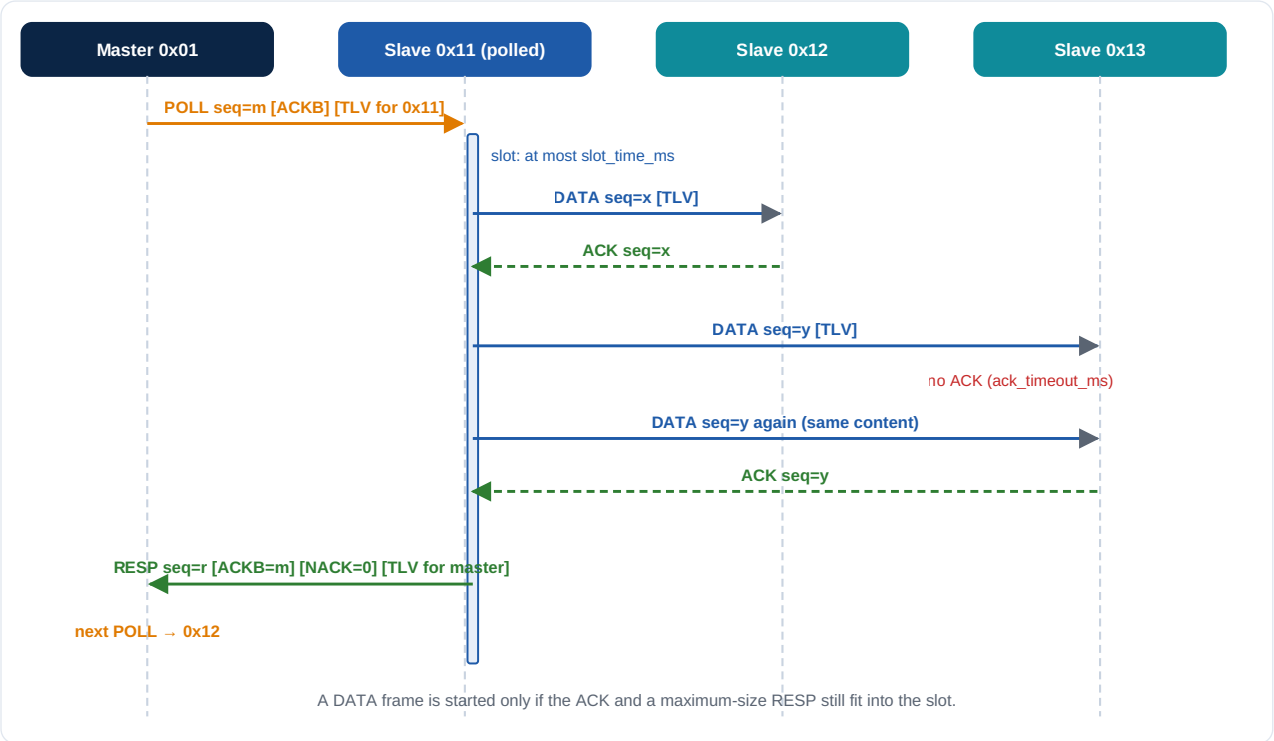
The link resets the parser when more than inter_byte_timeout_ms pass between two bytes of a frame.

`baton_parser_feed()` takes one byte per call:

1. Between frames it waits for `0xA5`; other bytes are ignored.
2. It stores DST, SRC, CTRL and LEN, updating the CRC register with each byte.
3. A LEN above `BATON_MAX_PAYLOAD` is rejected as soon as it is read: return `BATON_PARSE_ERROR`, wait for SYNC.
4. It stores LEN payload bytes, then the two CRC bytes.
5. A matching CRC returns `1` (the frame is in `parser->frame`); a mismatch returns `BATON_PARSE_ERROR`. Either way the parser waits for the next SYNC.

Recovery: the bytes of a rejected frame are not scanned again for a SYNC byte. The link resets the parser when more than `inter_byte_timeout_ms` pass between two bytes of a frame, so a frame cut off by noise costs at most that timeout. A corrupted parser structure (state or index out of range) is detected and reset instead of writing out of bounds; this is a unit test.

7.4 The slot



1. The master sends a POLL to the next slave in `slave_addresses`, carrying the master's messages for it.
2. The polled slave owns the slot. For every other slave with pending messages it:
   - packs them into a DATA frame that fits the remaining time budget,
   - waits up to `ack_timeout_ms` for the ACK and retries up to `max_retries` times,
   - after an ACK, serves the same peer again if it has more.
3. Finally the slave sends a RESP: it acknowledges the POLL, reports the NACK mask and carries messages for the master.
4. The master moves on after the RESP or after a timeout.

Each slot starts with a different peer (round robin), so no slave is always served last.

7.5 Delivery rules

Stop-and-wait per channel

Every ordered pair of nodes has an independent channel with a 4-bit sequence number:

- A sender has at most one unacknowledged frame with content per channel.
- A retransmission has **the same SEQ and the same messages**. It is rebuilt from references into the queue, so no copy of the frame is kept.
- SEQ advances only after an acknowledgement: an ACK, the ACKB of a RESP, or the ACKB of a POLL.
- The receiver remembers the last accepted SEQ of each sender. A frame with the same number is acknowledged again but not delivered, and counted in `duplicates`. Events are therefore delivered exactly once.

- Frames **without content** (POLL or RESP without messages) do not take part in sequencing: the receiver does not filter them, and the master does not acknowledge an empty RESP. An empty frame therefore never acknowledges the next frame with content in advance.

FIRST flag

After `baton_node_init()` a sender sets FIRST in each channel until its first acknowledgement. A receiver accepts a FIRST frame even when its SEQ equals the last accepted one, unless that one was also a FIRST frame with the same SEQ (a retransmission). A sender that restarts is therefore not mistaken for a duplicate.

Order

Events from one sender to one destination with the same priority arrive in the order they were sent: packing never skips a message of a queue. HIGH events may overtake NORMAL ones.

Abandoned content

When every event of an unacknowledged frame expired (TTL), the channel advances SEQ, so new content is not mistaken for a duplicate.

7.6 Unreachable nodes

Situation	Behavior
A slave does not acknowledge DATA	Its bit goes into the NACK mask of the RESP; the failure streak of the channel grows
3 failures in a row (`BATON_DEAD_STREAK`)	The peer is served last in the slot, once, without retry
A slave does not answer 3 POLLs in a row	The master polls it only every 4th cycle (`BATON_DEAD_POLL_DIV`)
An event is not delivered within `event_ttl_ms`	Removed; `on_drop(msg_id, dst, missed_nodes)`

The first acknowledgement resets the failure streak. `missed_nodes` lists the nodes that did *not acknowledge* the event: a node may still have received it when only its ACK was lost. The reverse never happens — every loss is reported.

7.7 Time budget

`F(n)` is the duration of an `n`-byte frame, `[n · 10000 / baudrate]` ms; `Fmax = F(7 + BATON_MAX_PAYLOAD)`.

Slave. A DATA frame is started only if an ACK and a maximum RESP still fit:

```
available_ms = slot_time_ms − elapsed − (Fmax + 2·tx_gap_ms + 2) − tx_gap_ms −
ack_timeout_ms
max_payload  = min(available_ms · baudrate / 10000 − 7, BATON_MAX_PAYLOAD)
```

The RESP therefore always ends within `slot_time_ms` after the POLL, whatever the number of peers and retries.

Master. It waits for the RESP `slot_time_ms + tx_gap_ms + 10` ms after the end of the POLL. If a frame is being received at that moment it waits for its end, but at most twice as long.

Checked by `baton_node_init()`:

```
slot_time_ms   ≥ Fmax + 3·tx_gap_ms + 2 + ack_timeout_ms + F(7 + 2 + BATON_MAX_MSG_LEN)
ack_timeout_ms > F(7) + tx_gap_ms
```

For the defaults (9600 bit/s, 64-byte payload, gap 2 ms, ACK timeout 20 ms, slot 250 ms):

Quantity	Value
Maximum frame (71 bytes)	74 ms
ACK (7 bytes)	8 ms
Minimum slot	174 ms
Empty slot (POLL 8 bytes + RESP 9 bytes + gaps)	about 25 ms
Longest slot, POLL included	at most 74 + 2 + 250 + 12 ≈ 340 ms
Longest cycle with N slaves	about N × 340 ms

The longest cycle depends only on the configuration. Heavy traffic makes messages wait longer in the queues; it does not make a slot longer.

7.8 Reference frames

Computed independently by the Python twin and used as unit and integration test vectors:

Frame	Bytes
POLL 0x01 → 0x11, FIRST, SEQ 0, ACKB not valid	`A5 11 01 20 01 00 7C 44`
ACK 0x11 → 0x12, SEQ 3	`A5 12 11 C3 00 45 5B`
RESP 0x11 → 0x01, FIRST, SEQ 0, ACKB valid SEQ 0, NACK 0	`A5 01 11 60 02 80 00 C9 91`
DATA 0x11 → 0x12, FIRST, SEQ 0, message 0x10 = `AB CD`	`A5 12 11 A0 04 10 02 AB CD D8 BA`

CHAPTER 8

Integration Guide

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

1. **Receive:** the UART receive interrupt passes every byte to `baton_node_rx_isr()`. It only writes the ring of the node, so it is safe in an interrupt.
2. **Transmit:** a write function that *starts* sending a complete frame and returns, plus — without echo — a call of `baton_node_tx_done_isr()` when the last byte left.
3. **Processing and application:** one context calls `baton_node_process()` every millisecond with the current time; the callbacks run there.

Complete, buildable programs for each pattern are described in Examples (p. 41).

8.1 The physical layer

Wiring	`rx_echo`	Notes
Single wire: TX and RX on one pin (STM32 half-duplex mode, or open-drain TX tied to RX)	`true`	The receiver hears every transmitted byte; the library compares the echo and ends the transmission with the last echo byte
Open-drain bus with pull-up, separate TX and RX pins tied together	`true`	Same as above
RS-485 with automatic or GPIO direction control, receiver disabled while sending	`false`	Call `baton_node_tx_done_isr()` from the TX-complete (shift register empty) interrupt, after switching the transceiver back to receive
RS-485 with the receiver always enabled	`true`	The echo also detects a transceiver that drives too late or too long

Keep the line idle-high, 8N1, the same baud rate on every node. The turnaround time of a transceiver must be shorter than `tx_gap_ms`.

8.2 Bare metal

```
void USART1_IRQHandler(void) /* example names for an STM32 */
{
    if (usart_rx_not_empty())
    {
        baton_node_rx_isr(&node, usart_read_byte());
    }
    if (usart_tx_empty() && (tx_position < tx_length))
    {
        usart_write_byte(tx_buffer[tx_position]);
        tx_position++;
    }
}
```

The write function stores the pointer and the length and enables the TX-empty interrupt; the library keeps the buffer valid until the end of the transmission. The main loop calls `baton_node_process(&node, millis)` every millisecond (SysTick). When `baton_node_send()` is also called from another interrupt, set `enter_critical` and `exit_critical` to disable and restore interrupts (`__disable_irq()` / restore the previous PRIMASK).

8.3 RTOS

Part	Where
`baton_node_rx_isr()`, `baton_node_tx_done_isr()`	UART interrupt handler; no RTOS call needed, the ring is lock-free
`baton_node_process()`	One task with a 1 ms period (`vTaskDelayUntil`, `k_sleep(K_MSEC(1))`) at a priority above the application tasks that wait for bus data
`baton_node_send()`, `baton_node_publish()`	Any task, with `enter_critical`/`exit_critical` set to a short interrupt lock (`irq_lock()`, `taskENTER_CRITICAL()`)
Callbacks	Run in the protocol task; post to a queue of the application task for longer work

The critical section must exclude the UART interrupt and every other context that calls send or publish. A mutex is not enough when an interrupt is one of them.

8.4 Adding the library to a CMake build

```
set(BATON_CONFIG_DIR ${CMAKE_CURRENT_SOURCE_DIR}/config)   # folder with
baton_user_config.h
include(path/to/Baton-C/baton.cmake)
target_link_libraries(app PRIVATE baton)
target_link_libraries(baton PUBLIC zephyr_interface)       # Zephyr applications only
```

`baton.cmake` defines the static library target `baton` with its include directories (the repository root, `inc/` and `BATON_CONFIG_DIR`) and the version definitions. For a Zephyr application, linking `zephyr_interface` compiles the library with the board's architecture flags.

Without CMake: compile `src/baton_protocol.c`, `src/baton_link.c`, `src/baton_queue.c` and `src/baton_node.c` with C11 and the three include folders.

8.5 Host

On a PC or a Linux board the master or a monitor reads the serial port with a short timeout and passes the bytes to `baton_node_rx_isr()` from the same thread that calls `baton_node_process()`. A blocking write is acceptable there; call `baton_node_tx_done_isr()` after it returns when the adapter has no echo. USB-UART adapters add latency (often 1–16 ms); keep `tx_gap_ms` and `ack_timeout_ms` above it.

8.6 Hardware bring-up

Baton-C has been verified on a simulated bus (unit tests, the bus simulation with noise and a dead node, the bare-metal example on a simulated board); it has **not yet been tested on hardware**. When bringing it up:

1. Check the line with a logic analyzer: idle high, 8N1, the configured baud rate, and the first POLL `A5 11 01 20 01 00 7C 44` (master 0x01 polling 0x11).
2. Start with the master and one slave; `on_poll_done` must report `responded`.
3. Check `collisions` (must stay 0) and `tx_timeouts` on every node — they reveal a wrong `rx_echo` setting or a transceiver that switches too late.
4. Add the other slaves and confirm the cycle time against Protocol → Time budget (p. 36).
5. Pull the cable of one slave: the others must keep their slots, and `on_drop` must report the events to it after `event_ttl_ms`.

CHAPTER 9

Examples

Three examples share one bus definition, `examples/config/baton_user_config.h` (the Baton Inspector (p. 44) bus definition `tools/baton_inspector/examples/example_bus.json` describes the same bus): master 0x01, slaves 0x11 and 0x12, the state `NODE_STATUS` (uptime in seconds) and the events `CMD_SET_OUTPUT` and `OUTPUT_REPORT`.

Example	Role	Platform	Built and run in CI
Bare metal	Slave 0x11	Any MCU; runs on a PC against a simulated board	Yes, as a self-checking test
Host serial master	Master 0x01	Linux, macOS, Windows	Built
Zephyr	Slave 0x12	Any Zephyr board	No — needs a Zephyr workspace; not compiled in this repository

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

9.1 Bare metal

File	Content
`examples/bare_metal/board.h`	Four board functions: `board_init()`, `board_millis()`, `board_uart_write()`, `board_idle()`; the application provides `app_uart_rx_isr()`
`examples/bare_metal/main.c`	The slave: configuration, main loop, uptime state, output commands
`examples/bare_metal/board_host.c`	`board.h` on a PC: a simulated single-wire bus with the master and slave 0x12 as further Baton nodes; checks the results

Structure. The UART receive interrupt calls `app_uart_rx_isr()`, which passes the byte to `baton_node_rx_isr()`. The main loop calls `baton_node_process()` every millisecond, publishes `NODE_STATUS` once per second, and in `on_message` applies `CMD_SET_OUTPUT` and answers with an `OUTPUT_REPORT` event to the sender. Everything runs in the main loop, so no critical section is configured.

On a PC. `board_idle()` advances a simulated clock by 1 ms and moves one byte per transmitter over the bus; every byte reaches every node, the sender included (echo). Before each of the three requests a burst of line noise — a broken header `A5 11 01 00` with an invalid length, and a stray byte — is put on the idle bus. Real output:

```
bare-metal example: slave 0x11 at 9600 bit/s on a simulated single-wire bus
  1502 ms  master -> 0x11  CMD_SET_OUTPUT output 0 = 0x01 (after line noise)
  1563 ms  master <- 0x11  OUTPUT_REPORT output 0 = 0x01
  2202 ms  0x12   -> 0x11  CMD_SET_OUTPUT output 1 = 0x55 (after line noise)
```

```
   2290 ms  0x12   <- 0x11  OUTPUT_REPORT output 1 = 0x55
   3106 ms  master -> 0x11  CMD_SET_OUTPUT output 2 = 0x00 (after line noise)
   3147 ms  master <- 0x11  OUTPUT_REPORT output 2 = 0x00
master: 274 polls (0 without answer), 3 CRC errors, 0 frames aborted by timeout
ok    master got 2 OUTPUT_REPORT events
ok    last report to the master: output 2 = 0x00
ok    slave 0x12 got 1 OUTPUT_REPORT event
ok    report to 0x12: output 1 = 0x55
ok    master sees the uptime state of 0x11
ok    slave 0x12 sees the uptime state of 0x11
ok    no collisions on the bus
EXAMPLE PASSED
```

The three CRC errors are the three noise bursts (their length byte is invalid).

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

Function	Implementation
`board_init()`	Clock the UART, configure single-wire half duplex 8N1 at the baud rate, enable the RX interrupt (and the TX interrupt in the driver)
`board_millis()`	A 1 ms SysTick counter
`board_uart_write()`	Store the buffer and start an interrupt or DMA transmission; return 0
`board_idle()`	`__WFI()`; the SysTick interrupt wakes the loop every millisecond

9.2 Host serial master

File	Content
`examples/host_serial/main.c`	Command-line master `baton_master`
`examples/host_serial/serial_port.h`	Five functions: open, write, read with timeout, close, millisecond clock
`examples/host_serial/serial_port_posix.c`	termios implementation (Linux, macOS)
`examples/host_serial/serial_port_win32.c`	Win32 implementation

```
baton_master <port> [--baud N] [--echo] [--seconds N] [--set ADDR OUTPUT VALUE]...
baton_master /dev/ttyUSB0 --set 0x11 0 1
baton_master COM3 --echo --seconds 30
```

The master polls 0x11 and 0x12, prints every message it receives and, once per second, the number of polls, the answered ones and the CRC errors. Each `--set` queues a `CMD_SET_OUTPUT` event. Use `--echo` with a single-wire adapter that receives its own bytes; without it, the end of `serial_write()` is taken as the end of the transmission (`baton_node_tx_done_isr()`). The example has not been run against hardware yet; without an adapter it reports `cannot open <port>`.

9.3 Zephyr

File	Content
`examples/zephyr/src/main.c`	Slave 0x12: UART interrupt driver, one protocol thread
`examples/zephyr/CMakeLists.txt`	Includes `baton.cmake`, links `zephyr_interface`
`examples/zephyr/prj.conf`	Serial, interrupt-driven UART, no heap
`examples/zephyr/app.overlay.example`	The chosen node `baton,uart`

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

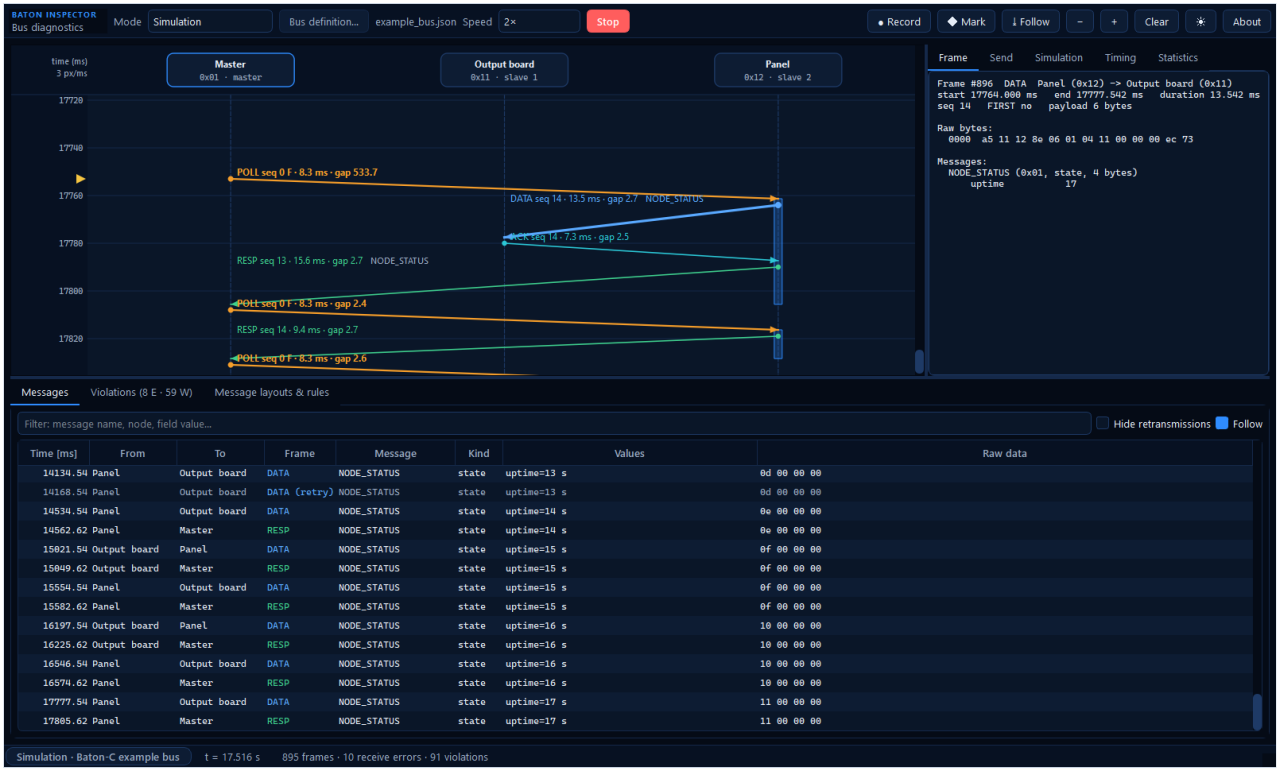
The UART interrupt passes received bytes to `baton_node_rx_isr()` and feeds the transmit FIFO from the buffer of the write function; the transmission ends with the echo of the last byte (`rx_echo`). A thread calls `baton_node_process()` every millisecond and publishes the uptime. `enter_critical` and `exit_critical` use `irq_lock()` so that other threads may send events. Disable the console and logging on the bus UART.

This example is not compiled in this repository (no Zephyr workspace in CI); treat it as a starting point.

CHAPTER 10

Baton Inspector

Baton Inspector is the diagnostic tool of Baton-C: a desktop GUI and a Python API for automated tests. It can be the master of a real bus, listen to it passively, or simulate a whole bus on the PC; it shows the communication as a live UML sequence diagram with the timing of every frame, measures slots, response times and cycles, reports timing and bus-usage violations, decodes message data with configurable field layouts, and records the parsed session in JSON Lines for analysis after a test run. It lives in `tools/baton_inspector`.



10.1 Modes

Mode	What runs	Hardware	Typical use
Simulation	The master and every slave of the bus definition on a simulated single-wire bus	None	Developing system tests, reproducing faults (noise, unplugged slave, late answers)
Master	The PC is the master of the real bus	USB-UART or USB-RS-485 adapter	Testing slaves without the real master; driving a system test
Monitor	The PC only listens	USB-UART or USB-RS-485 adapter	Measuring the timing of the real master and slaves
Attach	The GUI shows the live feed of a session started by a test	—	Watching an automated test run

How it uses the library. Every node the tool runs — the master and the simulated slaves — is a `struct baton_node` of the real Baton-C library, and every byte is framed by `baton_parser_feed()`. The library is compiled into a native bridge (`tools/baton_inspector/native/inspector_bridge.c`) with the `baton_user_config.h` of the system under test, so the tool knows the same messages as the firmware. Python only adds time stamps, decoding of the data fields, the timing analysis and the user interface.

Timing accuracy. In the simulation the times are exact (one byte every 10 bit-times). On a serial port they are the times the PC reads the bytes: USB adapters deliver bytes in chunks, which limits the accuracy to about 1–2 ms with a 1 ms latency timer (FTDI) and more with others. That is enough for slots, response times and cycles, not for the transmit gap (`tx_gap_ms`) or gaps between bytes. Rules are therefore checked with a tolerance (`tolerance_ms`, default 3 ms).

10.2 Installation

Requirements: Python 3 (tested with 3.12 and 3.14), CMake 3.20+, a C11 compiler (GCC/MinGW-w64, Clang or MSVC — the bitness of Python), and for the GUI and serial modes the packages of `tools/baton_inspector/requirements.txt` (PySide6, pyserial). Windows and Linux are supported; on Linux add the user to the `dialout` group for USB adapters.

```
python -m pip install -r tools/baton_inspector/requirements.txt
python tools/baton_inspector/baton_inspector.py --config tools/baton_inspector/examples/
example_bus.json --mode simulation --start
```

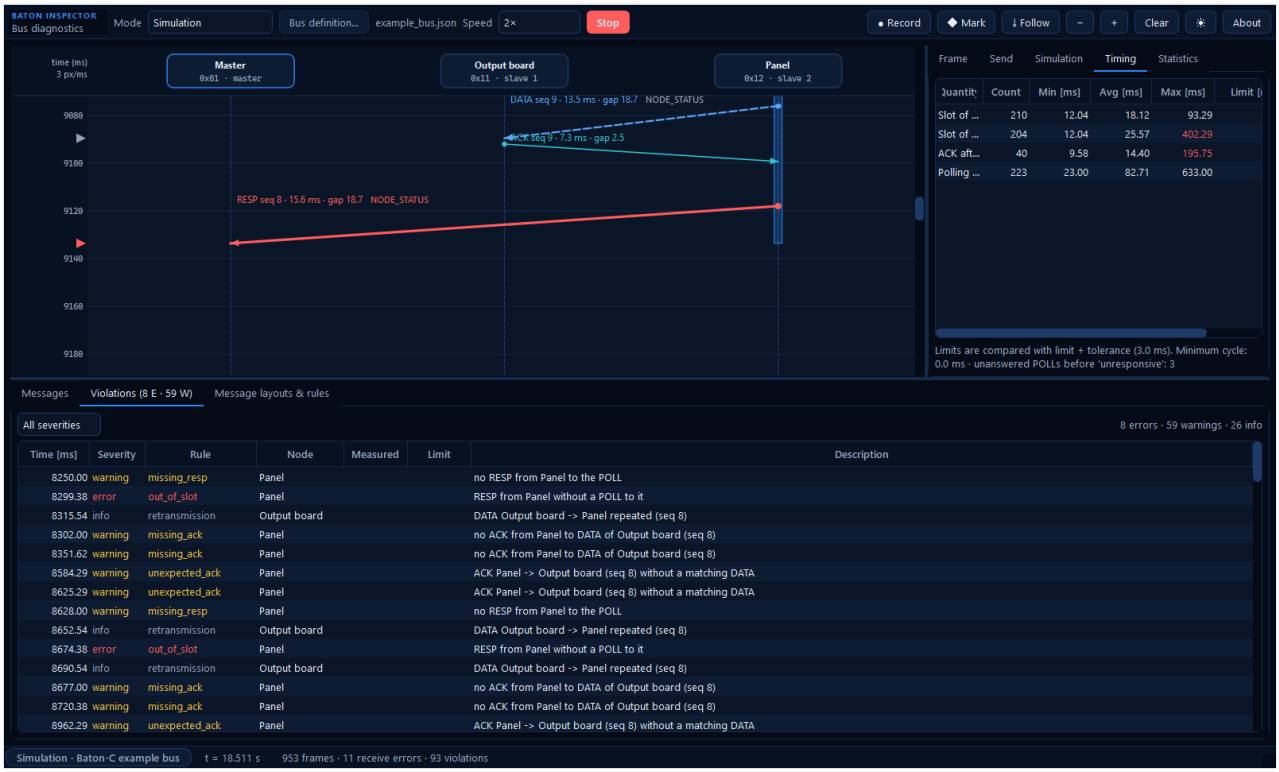
Option	Meaning
`--config FILE`	Bus definition (JSON)
`--mode simulation|master|monitor`	Mode to select
`--port PORT`	Serial port (`COM3`, `/dev/ttyUSB0`) or pyserial URL (`socket://host:port`)
`--start`	Start the session at once
`--attach PORT`	Show the live feed of a test session on `127.0.0.1:PORT`
`--theme dark|light`	Color theme

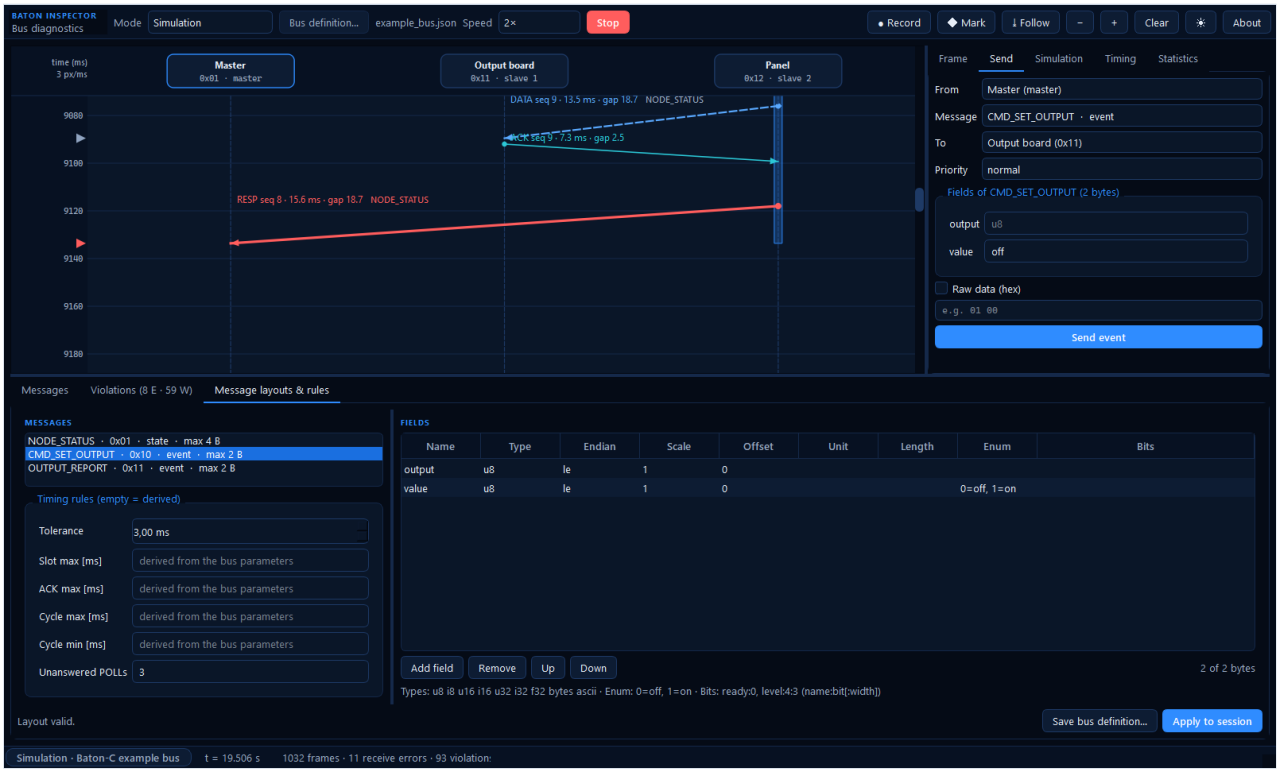
The first session builds the native bridge with CMake into `tools/baton_inspector/build/<hash>/` (2 s measured on Linux with GCC 13); later sessions with the same configuration file and library sources reuse it. In VS Code the tasks *Inspector: Run (Simulation)*, *Inspector: Run* and *Inspector: Tests* do the same.

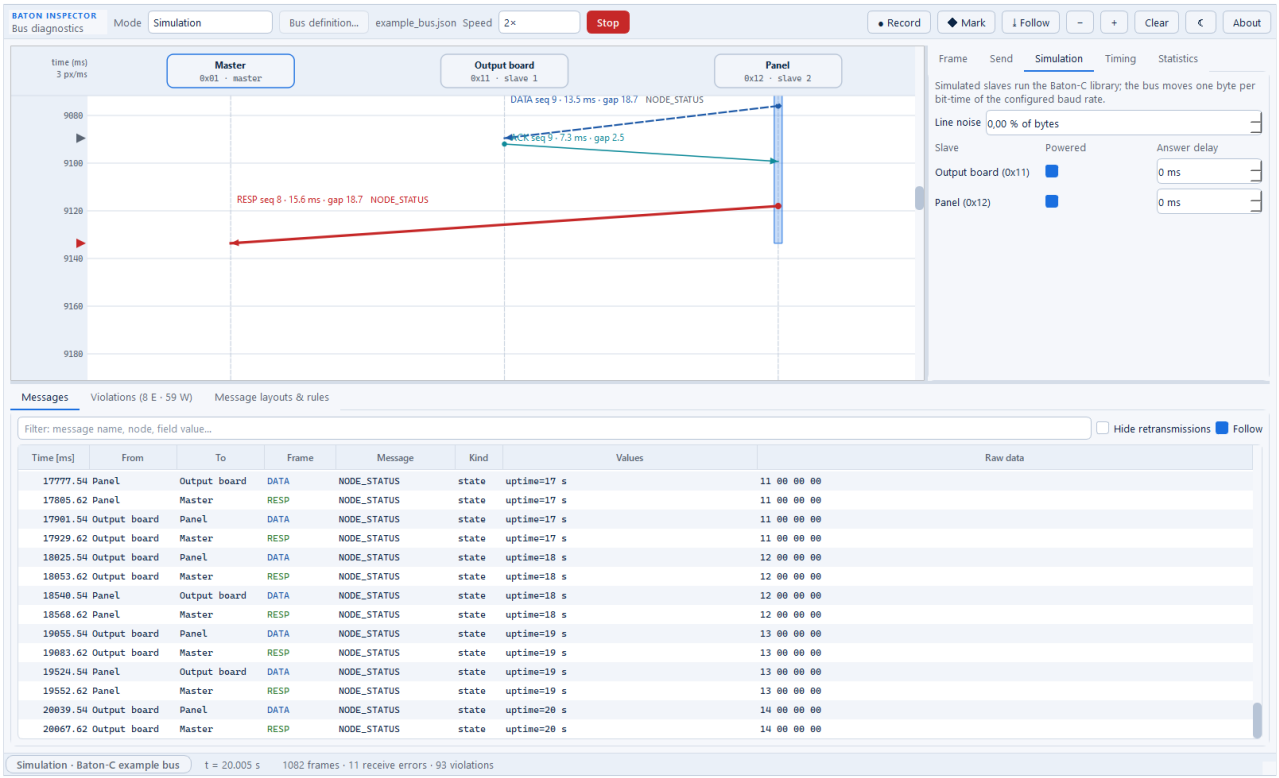
10.3 The window

Area	Content
Toolbar	Mode, bus definition, serial port and echo (master, monitor), speed (simulation), live port (attach), Start/Stop (F5), Record, Mark (Ctrl+M), Follow (End), zoom, Clear, theme

Area	Content
Sequence diagram	One lifeline per node. Each frame is an arrow from the sender at its start time to the receiver at its end time, labeled with type, sequence number, duration and the gap since the previous frame, followed by the names of its messages. The slot of the polled slave is an activation bar. Retransmissions are dashed, frames sent by the tool are bold, frames involved in a violation take the color of its severity. Triangles on the time axis mark violations, crosses receive errors, dashed lines test marks. Ctrl + wheel zooms, hovering shows the full frame
Frame	Times, sequence, ACKB/NACK, raw bytes and decoded messages of the selected frame, and its violations
Send	Send an event or publish a state from the master — in the simulation also from a simulated slave — with a form built from the field layout, or raw bytes
Simulation	Line noise, and per slave: powered or not, answer delay
Timing	Slots per slave, ACK times and cycles (count, min, avg, max) against their limits
Statistics	`baton_stats` of the nodes run by the tool
Messages	Every decoded message with time, sender, receiver, frame type, values and raw data; text filter, hide retransmissions
Violations	Every rule that was not met; click to jump to the frame
Message layouts & rules	The field layouts and timing rules of the bus definition: edit, apply to the running session, save







The screenshots are taken by `tools/baton_inspector/screenshots.py`, which drives the real window in simulation mode.

10.4 Bus definition

One JSON file describes the system under test (example: `tools/baton_inspector/examples/example_bus.json`). Relative paths are relative to the file.

```
{
  "name": "Baton-C example bus",
  "native": {"user_config": "../../../examples/config/baton_user_config.h"},
  "bus": {
    "baudrate": 9600, "master": "0x01", "slaves": ["0x11", "0x12"],
    "node_names": {"0x01": "Master", "0x11": "Output board", "0x12": "Panel"},
    "slot_time_ms": 250, "ack_timeout_ms": 20, "max_retries": 1, "tx_gap_ms": 2,
    "inter_byte_timeout_ms": 5, "cycle_period_ms": 0, "event_ttl_ms": 5000, "rx_echo": tru
e
  },
  "timing": {"tolerance_ms": 3, "rules": {"slot_max_ms": null, "ack_max_ms": null,
             "cycle_max_ms": null, "cycle_min_ms": null, "unanswered_polls": 3}},
  "messages": {
    "CMD_SET_OUTPUT": {"fields": [
      {"name": "output", "type": "u8"},
      {"name": "value", "type": "u8", "enum": {"0": "off", "1": "on"}}]}
  },
  "simulation": {
    "speed": 1.0, "noise": 0.0, "dead": [], "tx_delay_ms": {},
    "slaves": {"0x11": {
      "auto_reply": [{"on": "CMD_SET_OUTPUT", "send": "OUTPUT_REPORT", "copy": true}],
      "publish": [{"state": "NODE_STATUS", "period_ms": 1000, "fields": {"uptime": "$uptim
e_s"}}]}}
  }
}
```

Section	Content
`native.user_config`	The `baton_user_config.h` of the system: message IDs, lengths and sizes. Required
`bus`	Topology and the runtime parameters of `struct baton_node_config`; identical on every node of the real system
`timing`	Tolerance and the timing rules; `null` means derived from `bus` (see below)
`messages`	Field layouts by message name (names from `baton_user_config.h`; an optional `id` is checked)
`simulation`	Speed, noise, unpowered slaves, answer delays, and the behavior of the simulated slaves

Field types. `u8`, `i8`, `u16`, `i16`, `u32`, `i32` with `endian` (`le` default, `be`), `scale`, `offset` (value = raw × scale + offset) and `unit`; `f32`; `enum` (value → name); `bits` (named bits or bit ranges of an integer: `{"name": "level", "bit": 4, "width": 3}`); `bytes` and `ascii` with a `length`. Bytes after the last field are shown as `_rest`; data shorter than the layout as `_error`. The tool checks that the fields fit the `MAX_LEN` of the message.

Simulated slaves. `auto_reply` sends a message when one arrives: `copy` repeats the data, `fields` sets values (`"$output"` copies a field of the received message), `data` gives hex bytes, `to` is `"sender"` (default) or an address. `publish` publishes a state every `period_ms` (with an optional `offset_ms`); the values `$uptime_s` and `$uptime_ms` are the simulated time.

10.5 Timing rules

Rule	Sever-ity	Meaning	Limit (derived default)
`slot_overrun`	error	End of the POLL → end of the slave's RESP longer than the limit	`slot_max_ms` (`slot_time_ms`)
`out_of_slot`	error	A slave sends DATA or a RESP while another one holds the baton	—
`overlap`	error	A frame starts before the previous one ended	—
`invalid_sender`	error	POLL not from the master, RESP not to the master, DATA with the master	—
`cycle_too_long`	error	First POLL of a round → first POLL of the next round	`cycle_max_ms` (slaves × (maximum frame + 2 × `tx_gap_ms` + `slot_time_ms` + 10))
`missing_resp`	warn-ing	A POLL without RESP before the next POLL	—
`slave_unresponsive`	warn-ing	`unanswered_polls` POLLs in a row without RESP	3
`late_ack`	warn-ing	End of a DATA → end of its ACK longer than the limit	`ack_max_ms` (`ack_timeout_ms`)
`missing_ack`	warn-ing	A DATA without ACK before the next DATA to the same node or the end of the slot	—
`unexpected_resp`, `unexpected_ack`	warn-ing	A RESP or ACK that answers noth-ing	—
`unknown_node`	warn-ing	An address that is not on the bus	—
`crc_error`, `frame_timeout`	warn-ing	Bytes rejected by the parser; a frame cut off	—
`cycle_too_short`	warn-ing	A cycle shorter than `cycle_min_ms`	`cycle_min_ms` (`cycle_period_ms`)
`retransmission`	info	A frame repeated with the same sequence number and content	—

Every limit is compared with `limit + tolerance_ms`. A number in `timing.rules` overrides the de-rived value.

10.6 Python API

The API needs neither Qt nor a display; a test imports it from `tools/baton_inspector`.

```
from batoninspector import Session

with Session("bus.json", mode="simulation", speed=0, record="run.jsonl") as bus:
```

```
    bus.send("Output board", "CMD_SET_OUTPUT", {"output": 2, "value": "on"}, priority="hig
h")
    report = bus.wait_for("OUTPUT_REPORT", src="Output board", timeout_ms=1000)
    assert report.values == {"output": 2, "value": "on"}
    bus.assert_no_violations()
```

Call	Description
`Session(config, mode, port=None, rx_echo=None, speed=None, record=None, live_port=None)`	`mode`: `simulation`, `master`, `monitor`. `speed=0` (simulation): time advances only while the test waits — fast and repeatable; `1.0`: real time
`start()`, `stop()`, `with Session(...)`	Run the session
`send(dst, name, fields=None, data=None, priority="normal")`	Master: queue an event; `dst` is an address, a node name or `"broadcast"`
`publish(name, fields=None, data=None)`	Master: publish a state
`wait_for(name, src, dst, where, timeout_ms, since_ms)`	Wait for a message seen on the bus; `where` is a dict of field values or a predicate. By default only messages after the last `send`, `publish` or `mark` count. Raises `WaitTimeout`
`expect_none(name, src, dst, where, within_ms)`	Fail if a matching message appears
`find(...)`, `latest_state(name, src)`	Messages seen so far; the last value of a state
`run_for(ms)`, `mark(text)`, `now_ms`	Let time pass; insert a marker; session time
`get_violations(min_severity, rule, since_ms, until_ms)`, `assert_no_violations(min_severity="warning", ignore=())`	Timing and bus-usage violations
`timing_summary()`, `stats()`	Measured slots, ACK times, cycles and limits; `baton_stats` of the tool's nodes
`record(path)`, `stop_recording()`	JSON Lines log
`serve_live(port)`	Publish the session for the GUI (*Attach*) on `127.0.0.1`
`slave(address).send(...)`, `.publish(...)`, `.received`, `.set_deaf()`, `set_tx_delay(ms)`, `set_noise(p)`	Simulation only: drive a slave and inject faults
`apply_config(BusConfig)`	Use edited field layouts and timing rules

Records: `MessageRecord` (time, sender, receiver, name, kind, data, `values`, `text`, `retransmission`), `FrameRecord`, `ErrorRecord`, `Violation` (time, rule, severity, text, node, measured, limit, frames).

Example suite. `tools/baton_inspector/examples/test_example_system.py` runs the same tests against the simulation (default) or the real bus (`BATON_MODE=master BATON_PORT=COM3`), writes `test-results/<test>.jsonl` for every test, and publishes the session for the GUI when `BATON_LIVE_PORT` is set:

```
BATON_LIVE_PORT=47555 python -m pytest tools/baton_inspector/examples -v
python tools/baton_inspector/baton_inspector.py --attach 47555
```

10.7 Session log (JSON Lines)

One JSON object per line; times are session milliseconds, `wall` is the UTC time.

type	Content
`session`	Format version, start time, mode, bus parameters, effective limits, library version, message table, the bus definition
`frame`	`t_start_ms`, `t_end_ms`, `wall`, frame type, addresses, `first`, `seq`, prefix (ACKB, NACK), raw bytes, `retransmission`, `own`, and the messages with their decoded `values`
`error`	Rejected bytes: `crc` (CRC or LEN) or `timeout` (frame cut off)
`violation`	Time, rule, severity, text, node, measured value, limit, frame indexes
`mark`	A test marker
`stats`	`baton_stats` of the tool's nodes when the recording stopped
`end`	Timing summary and the number of violations

```python
from batoninspector import load_session, messages_of

log = load_session("test-results/test_output_command_is_confirmed.jsonl")
errors = [v for v in log.get("violation", []) if v["severity"] == "error"]
reports = [m for m in messages_of("run.jsonl") if m["name"] == "OUTPUT_REPORT"]
```

10.8 Tests of the tool

`tools/baton_inspector/tests` (field layouts, bus definition, timing rules on synthetic frames, sessions in simulation, a master on a serial loopback, recording, live feed) and the example suite run in the *Baton Inspector* CI workflow, together with a GUI start in offscreen mode.

CHAPTER 11

API Reference

This chapter lists every public header, macro, type and function. Applications normally include `baton.h` and use the node API (p. 53); the codec, the link and the queue are public for tools and tests. The generated Doxygen reference is in the *Doxygen* section of the website.

11.1 Headers

Header	Contents
`baton.h`	Umbrella: includes everything below; version macros
`inc/baton_config.h`	Sizes and range checks; includes the application's `baton_user_config.h`
`inc/baton_defs.h`	Frame layout, frame types, ACKB, CTRL and byte-order helpers
`inc/baton_protocol.h`	CRC-16, parser, encoder
`inc/baton_link.h`	Half-duplex link
`inc/baton_queue.h`	Event queue
`inc/baton_node.h`	Node: roles, return codes, configuration, statistics, application API

11.2 Version

Macro	Value	Meaning
`BATON_VERSION_MAJOR`, `BATON_VERSION_MINOR`, `BATON_VERSION_PATCH`	`1`, `0`, `0`	From the Git tag through CMake; fallbacks in `baton.h`
`BATON_VERSION`	`"1.0.0"`	Version string

11.3 Configuration macros

See Configuration (p. 28) for `BATON_MAX_NODES`, `BATON_MAX_PAYLOAD`, `BATON_MAX_MSGS_PER_FRAME`, `BATON_QUEUE_HIGH_SIZE`, `BATON_QUEUE_NORMAL_SIZE`, `BATON_RX_BUF_SIZE`, the message lists and the derived sizes.

Generated from the lists: `BATON_MSG_<NAME>` (enum `baton_msg_id`), `BATON_MSG_<NAME>_MAX_LEN` (enum `baton_msg_max_len`), `BATON_STATE_COUNT`, `BATON_EVENT_COUNT`, `BATON_STATE_DATA_SIZE`.

11.4 Protocol constants (`baton_defs.h`)

Macro	Value	Meaning
`BATON_SYNC`	`0xA5`	First byte of a frame
`BATON_HEADER_SIZE`	5	SYNC, DST, SRC, CTRL, LEN

Macro	Value	Meaning
`BATON_CRC_SIZE`	2	CRC bytes
`BATON_FRAME_OVERHEAD`	7	Non-payload bytes
`BATON_ADDR_BROADCAST`	`0xFF`	"Every other node" for `baton_node_send()`
`BATON_FT_POLL`, `BATON_FT_RESP`, `BATON_FT_DATA`, `BATON_FT_ACK`	0, 1, 2, 3	Frame types
`BATON_ACKB_VALID`	`0x80`	Valid bit of ACKB
`BATON_SEQ_MASK`	`0x0F`	Sequence number bits
`BATON_TLV_HEADER_SIZE`	2	ID and LEN of a message

Function	Description
`uint8_t baton_ctrl(uint8_t type, bool first, uint8_t seq)`	Build a control byte
`uint8_t baton_ctrl_type(uint8_t ctrl)`	Frame type of a control byte
`bool baton_ctrl_first(uint8_t ctrl)`	FIRST flag
`uint8_t baton_ctrl_seq(uint8_t ctrl)`	Sequence number
`uint16_t baton_get_u16be(const uint8_t* data)`	Read big endian
`void baton_put_u16be(uint8_t* data, uint16_t value)`	Write big endian
`uint32_t baton_get_u32le(const uint8_t* data)`	Read little endian
`void baton_put_u32le(uint8_t* data, uint32_t value)`	Write little endian

These are `static inline`; they do not check their pointer (like `memcpy()`).

11.5 Node API

Return codes

Macro	Value
`BATON_OK`	0
`BATON_ERR_PARAM`	−1
`BATON_ERR_NOT_INIT`	−2
`BATON_ERR_MSG_ID`	−3
`BATON_ERR_LEN`	−4
`BATON_ERR_ADDR`	−5
`BATON_ERR_FULL`	−6

Roles, priorities, limits

Macro	Value	Meaning
`BATON_ROLE_MASTER`, `BATON_ROLE_SLAVE`	0, 1	`baton_node_config::role`
`BATON_PRIO_NORMAL`, `BATON_PRIO_HIGH`	0, 1	Queue of an event
`BATON_MASK_BYTES`	`(BATON_MAX_NODES + 7) / 8`	NACK mask size
`BATON_POLL_PREFIX`, `BATON_RESP_PREFIX`	1, `1 + BATON_MASK_BYTES`	Payload prefixes
`BATON_MAX_MSG_LEN`	`BATON_MAX_PAYLOAD − BATON_RESP_PREFIX − 2`	Largest message
`BATON_DEAD_STREAK`	3	Failures before a node is treated as unreachable
`BATON_DEAD_POLL_DIV`	4	An unreachable slave is polled every 4th cycle

Callback types

Type	Signature	Called
`baton_write_fn`	`int (*)(void* context, const uint8_t* data, size_t length)`	To start a transmission; return 0 or negative
`baton_critical_fn`	`void (*)(void* context)`	Around the few instructions shared with interrupts
`baton_message_fn`	`void (*)(void* context, uint8_t src, uint8_t msg_id, const uint8_t* data, uint8_t length)`	Once per received message; `data` valid only during the call
`baton_drop_fn`	`void (*)(void* context, uint8_t msg_id, uint8_t dst, uint32_t missed_nodes)`	When an event expired
`baton_poll_done_fn`	`void (*)(void* context, uint8_t slave, bool responded, uint32_t nack_nodes)`	Master: when a slot ended

`struct baton_node_config`

Described field by field in Using the Library → Set up a node (p. 19). `baton_node_init()` copies it.

`struct baton_stats`

Field	Meaning
`tx_frames`	Transmissions started
`rx_frames`	Valid frames seen on the bus, any destination
`crc_errors`	Frames rejected: CRC mismatch or invalid LEN
`rx_timeouts`	Frames aborted by the inter-byte timeout
`rx_overflows`	Bytes lost because the receive ring was full

Field	Meaning
`collisions`	Transmissions whose echo did not match
`tx_timeouts`	Transmissions that failed to start or did not end in time
`ack_timeouts`	Slave: DATA frames without ACK
`retries`	Slave: DATA retransmissions
`resp_timeouts`	Master: POLLs without RESP
`duplicates`	Retransmitted content filtered out
`bad_messages`	Unknown ids or bad lengths inside frames
`events_dropped`	Events removed by the TTL
`queue_full`	Events rejected by `baton_node_send()`
`queue_high_used`, `queue_normal_used`	Bytes in use in each queue

`struct baton_node`

The node state, owned by the caller (normally a `static` variable). Treat its members as private; their layout may change in a minor version.

`baton_node_default_config()`

```
void baton_node_default_config(struct baton_node_config* config);
```

Fills `config` with zeros and the defaults for 9600 bit/s (slave, slot 250 ms, ACK timeout 20 ms, 1 retry, gap 2 ms, inter-byte timeout 5 ms, TTL 5000 ms, echo on). NULL is ignored.

`baton_node_init()`

```
int baton_node_init(struct baton_node* node, const struct baton_node_config* config,
                    uint32_t now_ms);
```

Parameter	Description
`node`	Node state; cleared and initialized
`config`	Configuration; copied
`now_ms`	Current time

Return	When
`BATON_OK`	The node is running
`BATON_ERR_PARAM`	NULL argument or invalid configuration; the node is not initialized

`baton_node_process()`

```
void baton_node_process(struct baton_node* node, uint32_t now_ms);
```

Runs the receive path, the transmitter, the state machine and the housekeeping (TTL, queue release). Call every millisecond from one context. Ignored for a NULL or uninitialized node.

`baton_node_send()`

```
int baton_node_send(struct baton_node* node, uint8_t dst, uint8_t msg_id, const void*
data,
                    uint8_t length, uint8_t priority);
```

Parameter	Description
`dst`	Destination address or `BATON_ADDR_BROADCAST`
`msg_id`	An EVENT id
`data`, `length`	Data, copied; `data` may be NULL when `length` is 0
`priority`	`BATON_PRIO_NORMAL` or `BATON_PRIO_HIGH`

Return	When
`BATON_OK`	Queued
`BATON_ERR_NOT_INIT`	NULL or uninitialized node
`BATON_ERR_PARAM`	`data` NULL with `length` > 0
`BATON_ERR_MSG_ID`	Not an EVENT id
`BATON_ERR_LEN`	`length` above `MAX_LEN`
`BATON_ERR_ADDR`	Unknown destination or the node itself
`BATON_ERR_FULL`	No room in the queue

The TTL of the event starts at the time of the last `baton_node_process()` call.

`baton_node_publish()`

```
int baton_node_publish(struct baton_node* node, uint8_t msg_id, const void* data,
                       uint8_t length);
```

Stores the new value of a STATE and marks it for every other node. Returns `BATON_OK`, `BATON_ERR_NOT_INIT`, `BATON_ERR_PARAM`, `BATON_ERR_MSG_ID` (not a STATE id) or `BATON_ERR_LEN`; never fails for lack of space.

`baton_node_rx_isr()`, `baton_node_tx_done_isr()`

```
void baton_node_rx_isr(struct baton_node* node, uint8_t byte);
void baton_node_tx_done_isr(struct baton_node* node);
```

Interrupt entry points: store a received byte (counted in `rx_overflows` when the ring is full); signal the end of a transmission when `rx_echo` is false. NULL is ignored.

`baton_node_get_stats()`

```
void baton_node_get_stats(const struct baton_node* node, struct baton_stats* stats);
```

Copies the counters; nothing is written when an argument is NULL.

11.6 Codec (`baton_protocol.h`)

Function	Returns
`uint16_t baton_crc16_update(uint16_t crc, uint8_t byte)`	Updated CRC register
`uint16_t baton_crc16(const uint8_t* data, size_t length)`	CRC of the buffer; `BATON_CRC_INIT` (0xFFFF) for NULL
`void baton_parser_init(struct baton_parser* parser)`	— (NULL ignored)
`int baton_parser_feed(struct baton_parser* parser, uint8_t byte)`	1 frame ready in `parser->frame`; 0 more bytes needed; `BATON_PARSE_ERROR` (−1) invalid LEN, CRC mismatch, corrupted state or NULL
`int baton_encode(uint8_t dst, uint8_t src, uint8_t ctrl, const uint8_t* payload, size_t payload_length, uint8_t* output, size_t output_capacity, size_t* output_length)`	0, or `BATON_PARSE_ERROR` for invalid input or too little capacity

`struct baton_frame` has `dst`, `src`, `ctrl`, `length` and `payload[BATON_MAX_PAYLOAD]`. `struct baton_parser` holds the frame being received, the CRC register, an index and the state (`BATON_PARSER_IDLE` between frames). `BATON_FRAME_MAX` is the largest frame.

11.7 Link (`baton_link.h`)

Function	Returns
`int baton_link_init(struct baton_link*, const struct baton_link_config*, uint32_t now_ms)`	0, or `BATON_PARSE_ERROR` for NULL, no write function, zero baud rate or timeout
`void baton_link_rx_isr(struct baton_link*, uint8_t byte)`	—
`void baton_link_tx_done_isr(struct baton_link*)`	—
`int baton_link_receive(struct baton_link*, uint32_t now_ms)`	`BATON_LINK_FRAME`, `BATON_LINK_TX_DONE` (echo complete) or `BATON_LINK_NONE`

Function	Returns
`int baton_link_transmit(struct baton_link*, uint32_t now_ms)`	`BATON_LINK_TX_DONE` or `BATON_LINK_NONE`
`int baton_link_send(struct baton_link*, const uint8_t* data, size_t length)`	0, or `BATON_PARSE_ERROR`
`void baton_link_cancel(struct baton_link*)`	—
`const struct baton_frame* baton_link_frame(const struct baton_link*)`	Last frame reported
`bool baton_link_rx_busy(const struct baton_link*, uint32_t now_ms)`	A frame is being received
`uint32_t baton_link_frame_ms(const struct baton_link*, uint32_t bytes)`	Wire time of `bytes` in ms, rounded up

`struct baton_link_config`: `write`, `context`, `baudrate`, `tx_gap_ms`, `inter_byte_timeout_ms`, `rx_echo`. `struct baton_link_counters`: `tx_frames`, `rx_frames`, `crc_errors`, `rx_timeouts`, `collisions`, `tx_timeouts`.

11.8 Event queue (`baton_queue.h`)

Function	Description
`baton_queue_init(queue, memory, size)`	Empty queue over `memory`; size rounded down to 4
`baton_queue_record_size(length)`	16 + length rounded up to 4
`baton_queue_alloc(queue, need, &offset)`	Reserve a record at the tail (with the critical section held)
`baton_queue_first(queue, head, tail)`, `baton_queue_next(queue, offset, tail)`	Walk the records of a head/tail snapshot
`baton_queue_used(queue)`	Bytes in use
`baton_queue_read`, `baton_queue_write`, `baton_queue_set_masks`, `baton_queue_set_flags`	Access the serialized header (`struct baton_record`)
`baton_queue_data(queue, offset)`	Data bytes of a record

Every queue function checks its pointers; offsets come from the queue itself.

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 `baton.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`, `-Werror` in the test builds	No warnings
Manual review	Pull request review	Reviewer

`misra-analyze` analyzes `src/*.c` with the configuration `tests/config/baton_user_config.h` and fails when the report contains a violation. Set `-DBATON_MISRA_FAIL_ON_VIOLATION=OFF` to get the report without failing. A violation that cannot be fixed must be added to the deviation register below — and, when the addon reports it, suppressed at that line with a `cppcheck-suppress` comment that names the deviation.

The addon checks most decidable rules, not the directives or the rules that need the whole program or the intent of the code; those are covered by the manual review notes.

12.2 Current status

Item	Result
Cppcheck MISRA addon (2.16.2)	0 violations; 4 suppressed findings, all DEV-04
Cppcheck, all checks (2.16.2)	0 findings
Unit test coverage of `src/`	100 % lines (1053), 100 % branches (601)
Deviations	4, 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, by the configuration invariants checked with `#error` and `_Static_assert` (payload, message lengths, queue capacity), or by the parser's own range checks, which also catch a corrupted parser structure. Unsigned time differences wrap by design.
Dir 4.11 (validity of arguments)	Every public function checks its pointer arguments and lengths. The `static inline` helpers of `baton_defs.h` 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 the caller-owned `struct baton_node`.
Rule 8.9, 8.4 (definitions)	Constant tables are `static const` at the narrowest scope.
Rule 11.3, 11.5 (pointer casts)	None: queue records are serialized byte by byte instead of casting the byte array to a structure.
Rule 15.5 (single point of exit)	Every function has one `return` at the end.
Rule 17.2 (recursion)	No recursion; the stack 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 the functions that return a status (`baton_node_init()`, `baton_node_send()`, `baton_node_publish()`, `baton_parser_feed()`, `baton_encode()`, `baton_link_*()`), the `baton_write_fn` callback, and the internal `pack()` result.
- **Rationale:** the values are status codes, not quantities whose size matters. `int` lets an integrator pass an existing driver function as `baton_write_fn` without a wrapper. `int` has at least 16 bits on every conforming compiler, enough for every value used.
- **Risk control:** the library does no arithmetic on status values; it compares them with 0 or with the named constants.

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

- **Location:** public constants that only applications and tools use, for example `BATON_VERSION_*` in `baton.h` and the generated `BATON_MSG_<NAME>_MAX_LEN`.
- **Rationale:** they are the public names of protocol and API values.

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

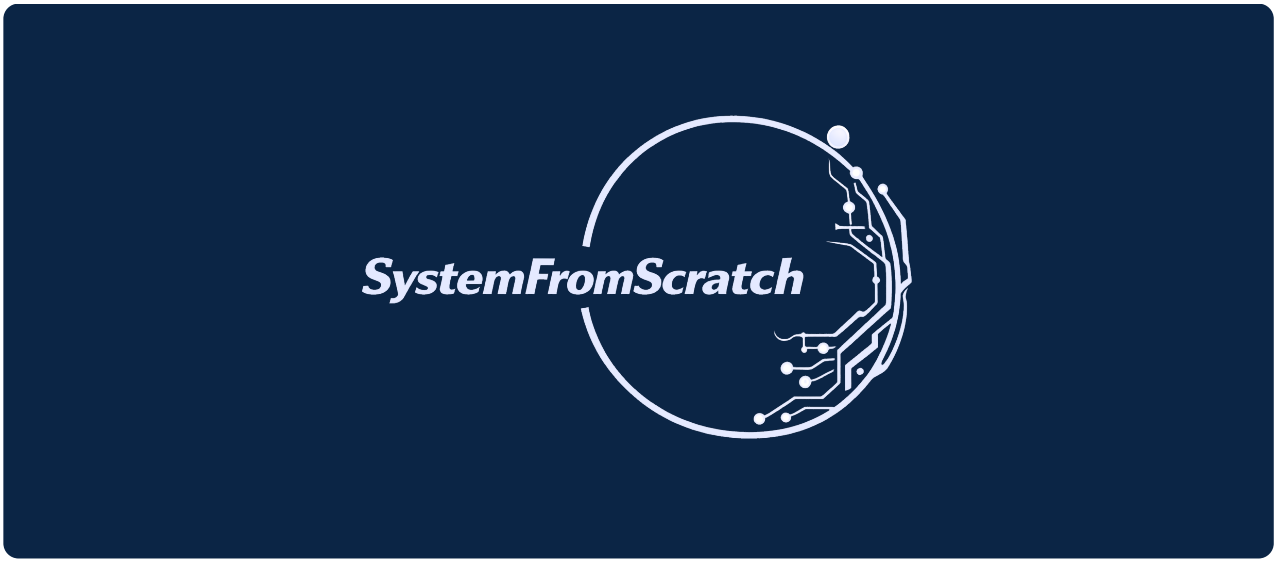
- **Location:** the public API functions that the library itself does not call, for example `baton_node_get_stats()` and `baton_crc16()`.

- **Rationale:** they are the library interface; internal linkage would remove them.

DEV-04: application message lists as X-macros (Rules 20.7 required, 20.10 and Dir 4.9 advisory)

- **Location:** `BATON_STATE_LIST(X)` / `BATON_EVENT_LIST(X)` in the application's `baton_user_config.h` (Rule 20.7: the parameter `X` is used as a macro name, which cannot be parenthesized); `BATON_X_MSG_ENUM_` and `BATON_X_MSG_SIZE_` in `inc/baton_node.h` (Rule 20.10: `##` builds `BATON_MSG_<NAME>`); the list macros themselves are function-like macros (Dir 4.9).
- **Rationale:** the lists are the single definition of the messages of a bus. From them the library generates, at compile time, the message identifiers, the length table, the state mailbox layout, a `switch` whose duplicate `case` values reject duplicate IDs, and `_Static_assert` checks of every length and of the queue capacity. Writing these by hand would duplicate each message in several places, which is the error the rules aim to prevent.
- **Risk control:** the expansion is checked by the compiler (`-Wall -Wextra -Wpedantic -Wconversion -Werror` in the tests) and by the compile-time assertions; the generated code is covered by the unit tests; the suppressions name this deviation at each line (`cppcheck-suppress misra-c2012-20.7` / `20.10`). Applications that run their own MISRA analysis inherit this deviation for their `baton_user_config.h`.

Author & Company



Baton-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 Baton-C, Anton designed the bus protocol and wrote the library, its unit, simulation and integration tests, 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

Baton-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/Baton-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. 26).
- **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 Baton-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 Baton-C without prior written permission.

The Baton bus protocol is specified by this project itself, in the Protocol (p. 31) chapter; the reference frames used in the tests are taken from that chapter.

APPENDIX A

Changelog

All notable changes to Baton-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 `baton.h` and `inc/` and the wire format stay compatible within a major version.

A.1 Unreleased

A.2 1.0.0 - 2026-10-25

First public release. Baton-C continues the internal `intercom` library (C99, single static instance, `Intercom_*` API) under a new name, API and repository layout; the wire format and the delivery rules are unchanged.

Added

- Layered library: definitions (`baton_defs.h`), codec (`baton_protocol`: CRC-16/CCITT-FALSE, incremental parser, in-place encoder), half-duplex link (`baton_link`: receive ring, inter-byte timeout, bus gap, echo check, transmit timeout), event queue (`baton_queue`) and node (`baton_node`: master/slave schedule, TLV packing, stop-and-wait channels, duplicate filter, TTL).
- Instance-based application API `baton_node_*` on a caller-owned `struct baton_node`; several buses in one program; explicit `now_ms` instead of a clock callback.
- Compile-time checks of the message lists: unique IDs, message lengths, queue capacity.
- Unit tests (GoogleTest, 132 tests) with the bus simulation as test cases, a multi-seed simulation runner (`baton_sim`, `tests/run_seeds.*`), and integration tests with a Python twin of the codec and a C/Python cross-check.
- Examples: bare-metal slave (runs in CI against a simulated bus with line noise), PC serial-port master (Linux, macOS, Windows), Zephyr slave.
- Documentation website (MkDocs Material) with the protocol specification, guides, measured memory, code size and stack, API reference, MISRA C:2012 page, and a printable PDF user manual; generated diagrams (`tools/docs/diagrams.py`) and a stack analysis script (`tools/docs/stack_usage.py`).
- Baton Inspector (`tools/baton_inspector`): diagnostic GUI (PySide6) and headless Python API for automated tests. Master, passive monitor and simulation modes on the real library (native bridge built per `baton_user_config.h`), live UML sequence diagram with frame timing, slot/ACK/cycle measurement and timing and bus-usage rules, JSON field layouts, JSON Lines session logs, live feed for watching a test from the GUI, example pytest suite.
- CI workflows: build, unit tests with the coverage gate, integration tests, static analysis (Cppcheck, MISRA, CodeQL), format check, static library, documentation, SBOM and release.

Changed

- C11 instead of C99; `include/Intercom.h` replaced by `baton.h` and `inc/`; `IntercomConfig.h` replaced by `baton_user_config.h` (`BATON_*` macros); CMake target `baton` in `baton.cmake`, presets `BATON[...]`.
- Queue records are serialized byte by byte (no pointer casts); every function has a single exit point (MISRA C:2012).

Quality

- MISRA C:2012: 0 violations reported by the Cppcheck addon; four deviations recorded in `docs/misra.md`. CI fails on any violation.
- 100 % line and branch coverage of `src/`, enforced by the `coverage` target.
- Code size 6620 B and 280 B of worst-case stack on Cortex-M4 (`-Os`), 1696 B of RAM per node with the configuration template.

APPENDIX B

License

Baton-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 (`baton.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 Baton-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 (Baton-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

SystemFromScratch® is a registered trademark; the license grants no rights to use the name or logo. The Apache License 2.0 grants no rights to use the SystemFromScratch® name or logo; they may not be used to endorse or promote products derived from Baton-C without prior written permission.

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.
```