



OPEN AQUARIUM PROJECT SPECIFICATION

Module Bus

PRE-DRAFT

Revision	0.4
Status	Pre-Draft. A candidate document submitted for consideration.
Date	2026-10-18
Identifier	module-bus
Authors	Trent Scholl, Project Lead, Open Aquarium Project
Editors	Trent Scholl
License	Community Specification License 1.0

Pre-Draft. This is a proposed initial draft. The Working Group has not approved it, and it will change. Several physical-layer values, and the values of the bus's protection, are provisional until the breadboard tests and the first circuit boards are complete (Appendix C). Comment by opening an issue with the `module-bus` label.

1 License

This Specification is subject to the Community Specification License 1.0 (the "Specification License"), available at <https://github.com/CommunitySpecification/1.0> and in `LICENSES/Community-Spec-1.0.md`. Contributions to this Specification are made under the Community Specification Contributor License Agreement 1.0.

The data type definitions published with this Specification are licensed under the Apache License 2.0 (`LICENSES/Apache-2.0.txt`).

The Scope that bounds the patent commitments made under the Specification License is set out in `Scope.md`. Exclusion notices, license acceptances, and withdrawals are recorded in `Notices.md`.

Notes:

1. Appendices are informative (non-normative) unless an appendix is explicitly marked normative.

THESE MATERIALS ARE PROVIDED "AS IS." The Contributors and Licensees expressly disclaim any warranties (express, implied, or otherwise), including implied warranties of merchantability, non-infringement, fitness for a particular purpose, or title, related to the materials. The entire risk as to implementing or otherwise using the materials is assumed by the implementer and user. IN NO EVENT WILL THE CONTRIBUTORS OR LICENSEES BE LIABLE TO ANY OTHER PARTY FOR LOST PROFITS OR ANY FORM OF INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS DELIVERABLE OR ITS GOVERNING AGREEMENT, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND WHETHER OR NOT THE OTHER MEMBER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

2 Compliance with Open Aquarium Tenets

2.1 Openness

The module bus is built on Cyphal version 1.0, whose specification is licensed CC BY 4.0 and whose reference libraries are MIT-licensed, with no fees, memberships, or vendor IDs. The Open Aquarium data types are published under the Apache License 2.0 and checked with the open-source Cyphal DSDL parser. Anyone can build a module or a controller.

2.2 Interoperability

Any conforming module works with any conforming controller, whoever made them. Modules present the classes of the Device Classes specification, so a controller needs no driver for a particular product. Because the bus uses standard Cyphal, general-purpose Cyphal tools, such as Yakut [Yakut], can inspect it.

2.3 Local Control

The module bus is a wired connection inside the equipment cabinet. It needs no network, no internet, and no account.

2.4 Safety

The bus carries the supervision that the Fail-Safe Behavior specification depends on, at a higher priority than routine traffic. A module that can no longer transmit stops counting its controller's messages as contact, and a controller that hears no module stops supervising them, so that outputs fail safe, and the controller knows when they have (Section 12). Modules tolerate being connected while powered and reversed supply polarity,

refuse firmware updates while dosing or filling, and keep their previous firmware until a new version has proved itself. A short circuit of bus power, or bus power on a CAN line, damages no node, and a fault inside one module stops no other (Section 7.6). Commands that stop equipment are sent at the highest priority in use, and a module carries them out even before it has checked which controller it is on (Section 9.3). A stop that a controller latches holds through a release, a takeover, and an emergency stop (Section 11.7). A module keeps its recent events, so that a dose or a test result is not lost while its controller is not listening (Section 11.4).

2.5 Sustainability

Modules can be added, replaced, and updated individually, so a system grows instead of being replaced. The bus suits small, inexpensive microcontrollers, which keeps modules cheap to make and to repair.

Revision History

Revision	Date	Description	Authors
0.1	2026-10-13	Initial Pre-Draft, revised with the findings of the module bus simulator.	Trent Scholl
0.2	2026-10-16	Protection of the bus against a fault inside a module, a short circuit of bus power, and bus power on a CAN line, with provisional values; a module that has not transmitted a Heartbeat for 3 s stops counting its controller's messages as contact, and a controller treats the outputs of a lost module as unknown until their <code>failSafeTimeout</code> plus 5 s after its last Heartbeat; a moved module also clears <code>oap.controller.unique_id</code> , asks for its controller's unique ID until it has an answer, and carries out stops meanwhile; register names for config attributes of the device and for extension config attributes, and the registers of a maximum, a minimum, and a default named; extension names within the sizes of the data types; a controller follows a firmware update's progress from the module's reads; the example of <code>oap.Attribute.0.1</code> a command parameter; a test procedure, mapped to its requirements, for every requirement; references cited by tag; one sentence per line; every open issue given a status; the Project Lead's decisions of 16 October 2026 applied: the values of Section 7.6 adopted, provisionally, and the supplying nodes of one bus limited to 4 A together, with redundant supplies combined behind one node's limit.	Trent Scholl
0.3	2026-10-17	The Project Lead's decisions of 17 October 2026 applied: another controller takes over a module only while it reports <code>communication-lost</code> , as Fail-Safe Behavior says; only commands that change an output resume control, as Fail-Safe Behavior says; <code>COMMAND_RELEASE</code> , by which a controller releases a module; an array within an object split across Attributes, which the receiver joins, so that a calibration record with several points can be carried; and a state attribute too long for one Attribute sent when it changes and at least every 60 s, rather than in every state message, with values that stay JSON.	Trent Scholl
0.4	2026-10-18	The Project Lead's decisions of 17 October 2026 applied: a module keeps its recent events with sequence numbers, in a new last field <code>seq</code> of <code>oap.Event.0.1</code> , and publishes again those that its controller missed when the controller asks with <code>COMMAND_SEND_EVENTS</code> ; a controller that hears no configured module for 3 s stops publishing OPERATIONAL Heartbeats until it hears one again; data types of major version 0 may change between revisions of this Specification, and a module states the revision it implements; latched stops carried, and kept through a release, a takeover, a move to another bus, and <code>COMMAND_EMERGENCY_STOP</code> . Also: a stop defined, as Fail-Safe Behavior defines it, to include a command that puts one attribute of the inactive state in its inactive value; <code>COMMAND_EMERGENCY_STOP</code> described as a stop without <code>revertAfter</code> that counts as contact, and never sent to a module whose outputs include one that an alarm function drives; a module that stays powered while its controller is replaced checks its controller's unique ID again; a new controller's takeover puts every endpoint in its fail-safe behavior at once, and a module without outputs can be taken over; the controller serves the program files of lights and pumps; the accuracy of network time, a module's clock without synchronization, <code>clockUncertainty</code> , and the time zone; firmware updates refused by Fail-Safe Behavior's definitions of dosing, filling, and water changes; the 4 A that the supplying nodes of one bus share is the installation's, and their makers document it; "A pass criterion that checks what a SHOULD asks applies to a product that does it"; a test for each; references to the final revisions of the other specifications; issues 5, 6, 9, 10, 11, and 13 deferred beyond revision 1.0. As published, revision 0.2 gave the names in the data types 32 bytes, and <code>oap.EndpointState.0.1</code> an extent of 8192 bytes; revision 0.3 gives them 64 bytes and 9216 bytes (Section 8.1.4).	Trent Scholl

Contents

1	License	2
2	Compliance with Open Aquarium Tenets	2
2.1	Openness	2
2.2	Interoperability	2
2.3	Local Control	2
2.4	Safety	2
2.5	Sustainability	3
	Revision History	3
3	Scope	6
4	References	6
4.1	Normative References	6
4.2	Informative References	6
5	Terms, Definitions, and Conventions	7
5.1	Requirements Language	7
5.2	Terms and Definitions	7
5.3	Notation	7
6	Introduction	8
6.1	Purpose	8
6.2	Design Principles	8
6.3	Overview	8
7	Physical Layer	9
7.1	Signaling and Bit Rates	9
7.2	Topology, Cable, and Termination	9
7.3	Power	9
7.4	Connectors	9
7.5	Connecting While Powered	9
7.6	Protection	10
8	Transport	10
8.1	Cyphal Version	10
8.2	Cyphal Functions	11
8.3	Priorities	11
8.4	Bus Errors	11
9	Identity and Addressing	12
9.1	Identity	12
9.2	Node-IDs	12
9.3	Node-ID Allocation	12
9.4	Network Time	13
10	Automatic Configuration	13
10.1	Discovery	13
10.2	Device Description	13
10.3	Ports	14
10.4	Taking Control	14
10.5	Registers	14
11	Carrying the Device Classes	15
11.1	Endpoint State	15
11.2	Configuration	15
11.3	Commands	16
11.4	Events	16
11.5	Faults and Health	17
11.6	Value Encoding	17
11.7	Node Commands	17
11.8	Program Files	18
12	Supervision and Fail-Safe Behavior	18
13	Firmware Updates	19
14	Conformance and Testing	20

14.1	Modules	20
14.2	Controllers	20
14.3	Test Procedures	20
15	Security Considerations	24
Appendix A	— Approval Checklist	25
Appendix B	— A Module Joining the Bus (Informative)	26
Appendix C	— Open Issues (Informative)	26

3 Scope

This Specification defines the module bus: the wired connection between a controller and its modules. It defines:

- the physical layer: CAN FD signaling, bit rates, topology, termination, power, protection, and connector requirements;
- the use of Cyphal/CAN version 1.0 as the transport;
- how modules are identified and addressed;
- how a controller configures a new module without human help;
- how the device classes are carried on the bus; and
- supervision, firmware updates, and conformance tests.

This Specification does not define:

- the device classes themselves, which the Device Classes specification defines;
- fail-safe behavior, which the Fail-Safe Behavior specification defines; this Specification only defines how it is carried on the bus;
- the connector and pinout, which remain open (Appendix C); or
- how firmware images are signed, which the Security and Access specification defines [OAP-SA].

4 References

4.1 Normative References

- **[RFC 2119]** Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- **[RFC 8174]** Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, May 2017.
- **[RFC 8259]** Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, December 2017.
- **[ISO 11898-1]** ISO 11898-1:2015, "Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling".
- **[ISO 11898-2]** ISO 11898-2, "Road vehicles — Controller area network (CAN) — Part 2: High-speed physical layer".
- **[Cyphal]** OpenCyphal, "Cyphal Specification", Version 1.0, revision 2025-05-16, https://specification.opencyphal.org/Cyphal_Specification.pdf.
- **[Cyphal-DT]** OpenCyphal, "Public regulated data types", commit 86e2f332419ff015f4cc9f1bb7843b02d5f7a5ed, https://github.com/OpenCyphal/public_regulated_data_types.
- **[OAP-DC]** Open Aquarium Project, "Open Aquarium Device Classes", Revision 0.5, [specs/device-classes/spec.md](#).
- **[OAP-FS]** Open Aquarium Project, "Open Aquarium Fail-Safe Behavior", Revision 0.4, [specs/fail-safe-behavior/spec.md](#).
- **[OAP-FP]** Open Aquarium Project, "Open Aquarium Flow Programs", Revision 0.3, [specs/flow-programs/spec.md](#).
- **[OAP-LP]** Open Aquarium Project, "Open Aquarium Lighting Programs", Revision 0.3, [specs/lighting-programs/spec.md](#).
- **[OAP-SA]** Open Aquarium Project, "Open Aquarium Security and Access", Revision 0.3, [specs/security-and-access/spec.md](#).

ISO 11898 is available for purchase; every other normative reference is free.

4.2 Informative References

- **[P0001]** Open Aquarium Project, "Proposal 0001: Reference Controller Architecture", [proposals/0001-reference-controller-architecture.md](#).

- **[Breadboard]** Open Aquarium Project, “Breadboard Prototype”, [hardware/controller/breadboard/README.md](#).
- **[OAP-DM]** Open Aquarium Project, “Open Aquarium Data Model”, Revision 0.4, [specs/data-model/spec.md](#).
- **[OAP-DO]** Open Aquarium Project, “Open Aquarium Dosing”, Revision 0.3, [specs/dosing/spec.md](#).
- **[OAP-DP]** Open Aquarium Project, “Open Aquarium Device Protocol”, Revision 0.3, [specs/device-protocol/spec.md](#).
- **[OAP-PI]** Open Aquarium Project, “Open Aquarium Probe Interface”, Revision 0.4, [specs/probe-interface/spec.md](#).
- **[OAP-PM]** Open Aquarium Project, “Open Aquarium Power Module”, Revision 0.3, [specs/power-module/spec.md](#).
- **[OAP-WT]** Open Aquarium Project, “Open Aquarium Water Testing”, Revision 0.3, [specs/water-testing/spec.md](#).
- **[Yakut]** OpenCyphal, Yakut, a command-line tool for Cyphal, <https://github.com/OpenCyphal/yakut>.

5 Terms, Definitions, and Conventions

5.1 Requirements Language

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “NOT RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in BCP 14 [RFC 2119] [RFC 8174] when, and only when, they appear in all capitals, as shown here.

5.2 Terms and Definitions

Terms defined in the Device Classes and Fail-Safe Behavior specifications, such as device, endpoint, controller, contact, and lost device, have the same meaning here [OAP-DC] [OAP-FS]. Cyphal terms, such as node, node-ID, subject, service, transfer, and register, have the meanings given in [Cyphal].

Module: A device connected to a controller through the module bus.

Bus: One module bus: a single CAN segment with its controller and modules.

Unique ID: The 128-bit identifier of a node, reported in `uavcan.node.GetInfo` as `unique_id`.

Port: A subject or service used by a node. Ports are named in the node’s registers (Section 10.3).

Network time: The time shared on the bus by Cyphal time synchronization (Section 9.4).

Configuration authority: The node that assigns node-IDs and port-IDs to the other nodes. On the module bus, this is the controller.

Supplying node: A node designed to supply power to the bus (Section 7.3.6), such as a controller with a power supply of its own.

Stop: A command that puts an endpoint, or one attribute of its inactive state, in its inactive state, as Fail-Safe Behavior Section 8.2 defines it [OAP-FS]. On the module bus, a stop is an `oap.Command` request such as `stop`, or `set` with `on`, `running`, or `enabled false`, with or without `latch` [OAP-DC Section 7.4.13], or `COMMAND_EMERGENCY_STOP` (Section 11.7).

5.3 Notation

Cyphal data types are written with their full name and version, such as `uavcan.node.Heartbeat.1.0`. Open Aquarium data types are in the `oap` namespace and defined in the `dsdl` directory.

6 Introduction

6.1 Purpose

Commercial controllers each connect their modules with a bus of their own, so modules from one maker cannot be used with another maker's controller [P0001]. This Specification defines one open bus, so that any conforming module works with any conforming controller.

6.2 Design Principles

1. **Use an open standard where one exists.** The bus uses CAN FD and Cyphal version 1.0, and defines only what they leave open.
2. **Pin what we depend on.** This Specification uses Cyphal version 1.0 and a fixed commit of its standard data types, and does not follow later revisions until the Working Group decides to.
3. **No setup by hand.** A controller finds, addresses, and configures a new module without help from the owner.
4. **Self-describing modules.** A module tells the controller what it is, using the device description of the Device Classes specification.
5. **Safety traffic first.** Supervision and commands that stop equipment are sent at higher priority than routine traffic.
6. **One fault stops one module.** A fault inside a module stops no other node, and a fault in the cable damages none (Section 7.6).

6.3 Overview

Table 1 — Layers of the module bus

Layer	Defined by	Section
Open Aquarium application: device descriptions, endpoint state, commands, events, and configuration	This Specification, using [OAP-DC] and [OAP-FS]	10 to 13
Node functions: heartbeat, node information, registers, file transfer, commands, time synchronization, node-ID allocation	[Cyphal], [Cyphal-DT]	8, 9
Transport: Cyphal/CAN	[Cyphal]	8
Data link and physical layer: CAN FD, cable, termination, power, protection, connectors	[ISO 11898-1], [ISO 11898-2], and this Specification	7

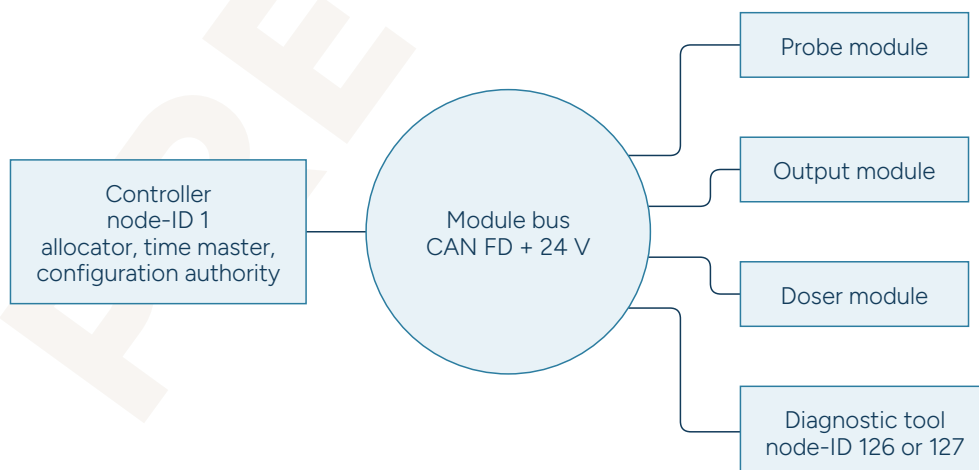


Figure 1 — A module bus

7 Physical Layer

7.1 Signaling and Bit Rates

1. The bus **MUST** use CAN FD as defined in [ISO 11898-1], with ISO CAN FD frame format.
2. The default bit rates are 1 Mbit/s for the arbitration phase and 2 Mbit/s for the data phase. These values are provisional (Appendix C).
3. A module **MUST** support CAN FD and classic CAN operation. Its bit rates and mode are set by the Cyphal register `uavcan.can.bitrate` (arbitration rate, then data rate, in bit/s). As [Cyphal-DT] defines, equal values select classic CAN.
4. All nodes on a bus **MUST** use the same mode and bit rates. Classic CAN at 1 Mbit/s is intended for development hardware whose CAN controller does not support CAN FD.
5. Transceivers **MUST** comply with [ISO 11898-2] and be rated for CAN FD at a data rate of at least 2 Mbit/s. Transceivers rated for 5 Mbit/s are RECOMMENDED.

7.2 Topology, Cable, and Termination

1. The bus **MUST** be a single line, terminated with 120 Ω at each end.
2. Modules **SHOULD** have separate in and out connectors, so that the bus passes through each module and stubs are avoided. Where a stub is unavoidable, it **MUST NOT** be longer than 0.3 m.
3. The total length of the bus **MUST NOT** exceed 20 m. This value is provisional (Appendix C).
4. CAN H and CAN L **MUST** be carried on a twisted pair with a characteristic impedance of about 120 Ω . Bus power (Section 7.3) **MUST** be carried on separate conductors in the same cable.
5. A module **SHOULD** have a termination resistor that can be switched on and off with the register `oap.can.termination` (Section 10.5), so that the controller can enable it on the last module. The controller cannot tell which module is last, so it asks the owner during installation, for example by having each module identify itself (Section 11.7). The controller **SHOULD** terminate its own end of the bus.

7.3 Power

1. The bus carries 24 V DC nominal power, within the limits of safety extra-low voltage.
2. A module **MUST** operate with between 18 V and 30 V at its connector.
3. A module **MUST** tolerate reversed supply polarity without damage.
4. A module **MUST** limit its inrush current so that connecting it does not disturb other nodes (Section 7.5).
5. A module **MUST** report the most current it can draw from the bus in the register `oap.bus.max_current` (Section 10.5), so that the controller can check the power budget.
6. A node designed to supply power to the bus **MUST** prevent current from flowing back into its own supply, so that several supplies can share a bus and the failure of one does not stop it. A node not designed to supply the bus **MUST NOT** feed power into it.

7.4 Connectors

The connector and pinout are not yet chosen (Appendix C). Whatever connector is chosen:

1. it **MUST** carry CAN H, CAN L, bus power, and ground, and **MAY** carry a shield;
2. it **MUST** be keyed by function, so that it cannot be mated with a connector of a different function, and in particular so that nothing carrying bus power can be plugged into a sensor port;
3. it **MUST** lock or latch when mated;
4. it **MUST** resist splashing when mated; and
5. its contacts for bus power and ground, and the conductors of the cable that carry them, **MUST** be rated for at least the current that the supplying nodes of a bus may deliver into it together (Section 7.6.7).

7.5 Connecting While Powered

1. Connecting or disconnecting a module while the bus is powered **MUST NOT** damage any node, restart any other node, or change any output of any other node.
2. Transient errors on the bus while a module is connected or disconnected are permitted. They **MUST NOT** cause any node to remain in the bus-off state (Section 8.4).

7.6 Protection

A damaged cable, or a fault inside a module, can short-circuit bus power or put it on a CAN line. This section keeps such a fault from damaging any node, and a fault inside one module from stopping the others. None of its requirements depends on the choice of connector, and its values are provisional (Appendix C, issue 3).

1. A fault inside a module, or in equipment connected to it, such as a short circuit between the module's supply rails or across one of its outputs, MUST NOT restart any other node, change any output of another node, or stop other nodes from communicating.
2. A module MUST have protection at its bus power input, so that, whatever fails behind that protection, the module draws no more than twice its `oap.bus.max_current` (provisional) once the protection has acted. An electronic fuse at the input, which limits the current at once and then disconnects the module, meets items 1 and 2; a fuse or a resettable fuse alone may act too slowly for item 1.
3. A supplying node MUST limit the current it delivers into the bus to no more than 4 A (provisional), and MUST withstand a short circuit between bus power and ground, for as long as it lasts, without damage.
4. A supplying node MUST restore bus power by itself, without restarting, within 5 s (provisional) after a short circuit or an overload ends. A supplying node that also runs functions of its own, such as a controller, MUST keep running them while bus power is short-circuited.
5. Every node MUST withstand, without damage, any voltage from -36 V to +36 V (provisional) on CAN H or CAN L, relative to its ground, for as long as it lasts, such as bus power on a CAN line of a damaged cable. Once such a fault ends, every node MUST resume communicating by itself, without restarting (Section 8.4).
6. A controller that supplies the bus SHOULD warn its owners while it is limiting the current it delivers into the bus.
7. The supplying nodes of one bus together deliver no more than 4 A (provisional) into it, because the conductors beside a short circuit carry the current of every supplying node on one side of it. An installation therefore gives a bus one supplying node, or several whose limits add up to no more than 4 A. A supplying node that combines two supplies behind its own limit, such as a controller with two supply inputs, gives a bus a second supply without adding to its total. The maker of a supplying node MUST document the most current it delivers into the bus, and that the supplying nodes of one bus together must deliver no more than 4 A, so that an installer can check the total. The total is a property of the installation, not of one node, which meets item 3 alone.

While bus power is short-circuited, modules that the bus powers lose power, and their outputs take their unpowered states [OAP-FS Section 8.3]. Equipment that must keep running, such as a return pump, belongs on an output whose unpowered state is on, or on a module with a supply of its own, such as a power module whose electronics run from mains [OAP-PM Section 10.1].

8 Transport

8.1 Cyphal Version

1. Nodes MUST implement Cyphal/CAN as defined in [Cyphal] version 1.0, over CAN FD, or over classic CAN when the bus runs in classic mode.
2. Nodes MUST use the data types of [Cyphal-DT] at the versions listed in Table 2, and the Open Aquarium data types in the `dsdl` directory.
3. Later revisions of Cyphal do not apply to this Specification unless a later revision of this Specification adopts them.
4. The Open Aquarium data types have major version 0, and MAY change between revisions of this Specification without a new version number. The layouts of a revision are those of the `dsdl` directory published with it. Revision 0.2, as first published, gave name of `oap.Attribute.0.1`, command of `oap.Command.0.2`, event of `oap.Event.0.1`, and value of `oap.Name.0.1` 32 bytes, and `oap.EndpointState.0.1` an extent of 8192 bytes; revision 0.3 gives them 64 bytes and 9216 bytes; revision 0.4 adds the last field `seq` to `oap.Event.0.1` (Section 11.4).

5. A module **MUST** state the revision of this Specification that it implements, as `module-bus` in the `specRevisions` of its device description [OAP-DC Section 8.1]. A controller **MUST** use with each module the layouts of the revision that the module states, and **MUST NOT** configure a module that states no revision, or a revision whose layouts it does not support, as it does not configure a port whose type it does not support (Section 10.3). A controller's maker documents the revisions it supports.

8.2 Cyphal Functions

Table 2 — Cyphal data types used on the module bus

Data type	Port-ID	Module	Controller	Used for
<code>uavcan.node.Heartbeat.1.0</code>	7509	Publishes	Publishes	Liveness and supervision (Section 12)
<code>uavcan.node.GetInfo.1.0</code>	430	Serves	Calls	Identity (Section 9.1)
<code>uavcan.node.ExecuteCommand.1.3</code>	435	Serves	Calls	Restart, identify, firmware update, and other node commands (Section 11.7)
<code>uavcan.register.Access.1.0</code>	384	Serves	Calls	Configuration (Sections 10 and 11.2)
<code>uavcan.register.List.1.0</code>	385	Serves	Calls	Discovering registers (Section 10.3)
<code>uavcan.file.Read.1.1</code>	408	Serves its device description; calls for firmware images and program files	Calls for device descriptions; serves firmware images and program files	Sections 10.2, 11.8, and 13
<code>uavcan.file.GetInfo.0.2</code>	405	Serves	Calls, and serves	File sizes
<code>uavcan.time.Synchronization.1.0</code>	7168	Subscribes	Publishes	Network time (Section 9.4)
<code>uavcan.pnp.NodeIDAllocationData.2.0</code>	8165	Publishes anonymously	Subscribes and publishes	Node-ID allocation on CAN FD (Section 9.3)
<code>uavcan.pnp.NodeIDAllocationData.1.0</code>	8166	Publishes anonymously	Subscribes and publishes	Node-ID allocation in classic mode
<code>uavcan.diagnostic.Record.1.1</code>	8184	MAY publish	SHOULD subscribe	Diagnostic messages

8.3 Priorities

Cyphal transfers carry one of eight priorities, from Exceptional (0) to Optional (7). Nodes **MUST** use the priorities in Table 3.

Table 3 — Transfer priorities

Priority	Used for
Immediate (1)	Stops (Section 5.2), latched or not, and their responses
High (3)	All other <code>oap.Command</code> requests and responses; the controller's Heartbeat
Nominal (4)	Module Heartbeats; time synchronization; <code>oap.EndpointState</code>
Low (5)	<code>oap.Event</code> , including events published again (Section 11.4); register access; <code>GetInfo</code> ; other <code>ExecuteCommand</code> requests; node-ID allocation; reading device descriptions and program files, and their file information
Slow (6)	Reading firmware images and their file information
Optional (7)	Diagnostic messages

Exceptional (0) and Fast (2) are reserved for future use.

8.4 Bus Errors

1. A node **MUST** recover from the bus-off state automatically.

2. Bus errors do not change any output directly. If they prevent contact for long enough, the Fail-Safe Behavior specification applies, and a module that cannot transmit stops counting its controller's messages as contact (Section 12.3).

9 Identity and Addressing

9.1 Identity

1. Each node MUST have a unique ID that no other node of any maker shares. A module SHOULD derive it from a manufacturer prefix and the microcontroller's own unique identifier.
2. A module's `GetInfo name` MUST be a reversed Internet domain name that identifies the product, such as `com.example.pm5`.
3. A module's `GetInfo software_version` MUST match the major and minor numbers of the `firmwareVersion` in its device description.

9.2 Node-IDs

Table 4 — Node-ID ranges

Node-ID	Use
0	Not used.
1	The controller.
2	Reserved for a standby controller.
3–125	Modules.
126–127	Diagnostic and maintenance tools, as in [Cyphal].

9.3 Node-ID Allocation

1. The controller MUST act as the only node-ID allocator on its bus, using `uavcan.pnp.NodeIDAllocationData.2.0` on CAN FD and `uavcan.pnp.NodeIDAllocationData.1.0` in classic mode.
2. The controller MUST keep a persistent table of the node-IDs it has allocated, keyed by unique ID, and MUST give a module the same node-ID every time it asks. In classic mode, `uavcan.pnp.NodeIDAllocationData.1.0` carries a 48-bit hash of the unique ID instead of the unique ID. A module MUST use the lowest 48 bits of the CRC-64-WE of its unique ID as the hash, as [Cyphal-DT] recommends. The controller MUST match the hash against the unique IDs in its table. It MUST keep a hash it does not know in the table until it has read the module's unique ID with `GetInfo`, and then record the unique ID instead, so that a module receives the same node-ID in CAN FD and classic mode.
3. A module that has no node-ID (`uavcan.node.id` is 65535) MUST request one by plug-and-play allocation, and MUST store the allocated node-ID in `uavcan.node.id`.
4. A module that has a stored node-ID MUST, before using it, listen for 3 s. If it hears a Heartbeat from another node using the same node-ID, it MUST clear its node-ID and request a new one.
5. After joining the bus, and whenever the node named in `oap.controller.id`, normally node-ID 1, publishes a Heartbeat after none for 3 s, or one whose `uptime` is lower than in its previous Heartbeat, a module whose register `oap.controller.unique_id` (Section 10.5) is set MUST read that node's unique ID with `GetInfo`, and compare the two. So a module that stays powered while its controller is replaced finds the new controller, as a module connected to another bus does. It MUST repeat the request at least every 2 s (provisional) until it has an answer. Until the comparison is complete, the module MUST refuse register writes and commands from that node as it refuses those of a node other than its controller, and MUST NOT count the node's messages as contact, because a new controller may have the same node-ID. It MUST nevertheless carry out a stop (Section 5.2) from that node as it would a stop from its controller, because stopping is always allowed [OAP-FS Section 8.2]; such a stop does not count as contact, and ends no latched stop, because only a command from the endpoint's controller ends one [OAP-FS Section 8.4.3]. If the unique IDs differ, the module has been moved to a different system, or its controller replaced: it MUST clear

`uavcan.node.id`, its port-IDs, `oap.controller.id`, and `oap.controller.unique_id`, and request a new node-ID. Its endpoints keep their fail-safe configuration, and each that is not in its fail-safe behavior MUST enter it at once, as when a device gets a new controller [OAP-FS Section 7.5.3], except that an endpoint that a latched stop holds keeps it [OAP-FS Section 8.4].

6. A controller MUST support at least 32 modules on its bus.

9.4 Network Time

1. The controller is the time synchronization master and MUST publish `uavcan.time.Synchronization.1.0` once per second while its clock is valid [OAP-FS Section 10.5].
2. Network time is Unix time: microseconds since 1970-01-01T00:00:00Z, not counting leap seconds.
3. While its clock is not valid, the controller MUST NOT publish time synchronization messages, and modules MUST report timestamps as unknown (zero).
4. Modules MUST synchronize their clocks to the controller's messages, and MUST use network time for every timestamp they send.
5. A module MUST report timestamps as unknown when it has received no synchronization message for 3 s.
6. While it receives synchronization messages, a module MUST keep its clock within 1 ms (provisional) of network time. Each message carries the time at which the controller sent the previous one, so a module that notes when it receives each message can measure its own error, however long a message waited before it was sent [Cyphal-DT]. This is well within what devices that keep in step need, such as the 20 ms of the Flow Programs specification [OAP-FP Section 11.4].
7. A module's clock stays valid without synchronization messages for as long as Fail-Safe Behavior Section 10.5 says: until the drift that its maker documents, over the time since its last synchronization message, could have moved it 60 s from network time [OAP-FS Section 10.5.4]. So a module that has lost contact keeps running its local programs, and reports `clock-invalid` only once its clock is no longer valid [OAP-DC Section 7.6], although it reports its timestamps as unknown after 3 s (item 5) [OAP-FS Section 10.5.5].
8. A module that implements `clockUncertainty` [OAP-DC Section 8.1] reports in it the largest difference it estimates between its clock and network time: at most 1 ms while it receives synchronization messages, and growing by its documented drift after the last one. It is a state attribute, carried as Section 11.1 says.
9. Network time is UTC, and the module bus carries no time zone. A module that needs the local time takes the time zone from the `posixTimeZone` that its controller writes in the program it runs, such as a light's fixture program, a pump program, a head program, or an analyzer program [OAP-LP Section 16.1; OAP-FP Section 15.1; OAP-DO Section 15.1; OAP-WT Section 14.1]. One that runs no such program, or whose program has none, uses its own setting, as those specifications say [OAP-DO Section 15.1].

10 Automatic Configuration

10.1 Discovery

When a module with a node-ID the controller has not configured, or whose configuration is incomplete, publishes a Heartbeat, the controller MUST configure it as described in this section. The controller SHOULD complete the configuration within 10 s.

10.2 Device Description

1. A module MUST serve its device description, as defined in Section 12 of [OAP-DC], as a file at the path `oap/description.json`, readable with `uavcan.file.Read.1.1`.
2. The description MUST be UTF-8 JSON [RFC 8259], and MUST NOT be larger than 64 KiB.
3. On the module bus, endpoint identifiers MUST consist of lowercase letters, digits, and single underscores, MUST begin with a letter, MUST NOT end with an underscore, and MUST NOT be longer than 32 characters. This lets them appear in register names (Section 11.2).
4. The controller MUST read and validate the description before configuring the module, and MUST NOT command a module whose description is invalid.

5. Config attribute values in the description, where given, are the manufacturer's defaults: the values that `COMMAND_FACTORY_RESET` restores. Current values are in registers (Section 11.2). An endpoint's `attributes` in the description MUST NOT include state attributes, and the description MUST NOT change until the module restarts.

10.3 Ports

A module MUST provide the ports in Table 5, each named by the standard Cyphal registers `uavcan.pub.PORT_NAME.id` and `.type` (or `uavcan.srv.PORT_NAME.id` and `.type`).

Table 5 — Open Aquarium ports of a module

Port name	Kind	Data type	RECOMMENDED port-ID
<code>oap_state</code>	Publication	<code>oap.EndpointState.0.1</code>	1000 + the module's node-ID
<code>oap_event</code>	Publication	<code>oap.Event.0.1</code>	2000 + the module's node-ID
<code>oap_command</code>	Service server	<code>oap.Command.0.2</code>	100

1. The controller MUST read each port's `.type` register and MUST NOT configure a port whose type it does not support.
2. The controller MUST assign a port-ID to each port by writing its `.id` register. It SHOULD use the RECOMMENDED port-IDs in Table 5, which keep traffic easy to identify with general-purpose tools.
3. A module MUST NOT publish or serve an Open Aquarium port until its port-ID is set.

10.4 Taking Control

1. The controller MUST write its own node-ID to `oap.controller.id` and its unique ID to `oap.controller.unique_id`.
2. The controller MUST then send `uavcan.node.ExecuteCommand.1.3` with `COMMAND_STORE_PERSISTENT_STATES`, so that registers the module does not store when they are written, such as port-IDs on some modules, also survive a restart. Configuration registers are stored when they are written (Section 11.2).
3. The controller resumes control of each endpoint, as defined in Section 7.4 of [OAP-FS], by sending it a command.

10.5 Registers

The registers in Table 6 are defined by this Specification, in addition to the configuration registers of Section 11.2 and the standard Cyphal registers `uavcan.node.id`, `uavcan.node.description`, `uavcan.can.bitrates`, and the port registers of Section 10.3.

Table 6 — Registers defined by this Specification

Register	Type	Flags	Default	Meaning
<code>oap.controller.id</code>	natural16[1]	mutable, persistent	65535	Node-ID of the module's controller. 65535 means none.
<code>oap.controller.unique_id</code>	unstructured	mutable, persistent	empty	Unique ID of the controller that configured the module.
<code>oap.can.termination</code>	bit[1]	mutable, persistent	false	Whether the module's bus termination resistor is switched on. Optional (Section 7.2).
<code>oap.bus.max_current</code>	real64[1]	immutable	set by the manufacturer	The most current, in amperes, the module can draw from the bus.

1. A module MUST accept register writes only from its controller, as named in `oap.controller.id`. While `oap.controller.id` is 65535, it MUST accept them from any node.
2. While the module reports `communication-lost` [OAP-FS Section 7.3], it MUST also accept a write to `oap.controller.id` and `oap.controller.unique_id` from any node, so that another controller can take over as Section 7.5.2 of [OAP-FS] allows. Once contact returns, it accepts them only from its controller again, although its endpoints stay in their fail-safe behavior until their controller resumes control. When another node takes over this way, each of the module's output and unit endpoints that is not yet in its fail-safe

behavior, because its `failSafeTimeout` is longer, MUST enter it at once, and one that a latched stop holds keeps it [OAP-FS Sections 7.5.3 and 8.4].

3. A module without output or unit endpoints never reports `communication-lost` (Section 11.5.3). It MUST accept a write to `oap.controller.id` and `oap.controller.unique_id` from any node once it has had no contact with its controller for 30 s (provisional), the RECOMMENDED `failSafeTimeout` of module endpoints (Section 12.2), counted from power-up if it has had none since, and until contact returns, so that another controller can take it over too [OAP-FS Section 7.5.2].

11 Carrying the Device Classes

11.1 Endpoint State

1. Each endpoint MUST publish its state as `oap.EndpointState.0.1` on the `oap_state` port. The module itself publishes its device attributes the same way, with an empty `endpoint`.
2. A state message MUST contain the endpoint's `controlSource` (for output and unit endpoints), its active faults, and the current values of all the state attributes it implements [OAP-DC Sections 7.2 and 7.5.3], except as item 4 allows. Fixed attributes are in the device description, and config attributes are in registers, so neither is repeated here.
3. State messages from one endpoint MUST be at least 100 ms apart, which limits them to 10 per second. Within that limit, an endpoint MUST publish its state:
 - within 100 ms of a change to its `controlSource`, its faults, or the state of an output; and
 - at least once per second for input endpoints, and at least once every 5 s for other endpoints and the module itself.
4. A state attribute whose value takes more than one `oap.Attribute.0.1` (Section 11.6.2) MAY be left out of a state message. An endpoint that leaves one out MUST include it in a state message within 100 ms of a change of its value, and at least once every 60 s (provisional). A receiver keeps the last value it received of such an attribute until a state message brings a new one, or until the module restarts or is lost. So a value that rarely changes, such as a probe's `lastCalibration` [OAP-PI Section 14.9], which may take 1 KB, does not go out twice a second for as long as the module runs.

11.2 Configuration

1. Each config attribute that an endpoint implements is a register named `oap.ep.ENDPOINT.ATTRIBUTE`, where `ENDPOINT` is the endpoint's identifier and `ATTRIBUTE` is the attribute's name converted from lower camel case to lowercase words separated by underscores. For example, the `failSafeTimeout` attribute of endpoint `outlet_1` is the register `oap.ep.outlet_1.fail_safe_timeout`.
2. A config attribute whose type is a list is stored as one register per item, named `oap.ep.ENDPOINT.ATTRIBUTE.N`, where `N` counts from 0. Each holds one item as JSON, or an empty string for no item. The immutable register `oap.ep.ENDPOINT.ATTRIBUTE.count` (natural16[1]) gives the number of item registers, which is the most items the module can hold. The list's value is its non-empty items, in order. A controller adds an item by writing it to an empty item register, and removes one by writing an empty string.
3. The device's `name` attribute is the standard register `uavcan.node.description`, and an endpoint's `name` attribute is the register `oap.ep.ENDPOINT.name`.
4. Configuration registers MUST be mutable and persistent. A module MUST store a configuration register when it is written, without waiting for `COMMAND_STORE_PERSISTENT_STATES`, so that it holds its value across power loss as Section 7.3 of [OAP-DC] requires.
5. A module MUST reject a register write whose value is outside the attribute's permitted range, and MUST keep the previous value.
6. A module SHOULD provide the optional Cyphal registers for each configuration register's maximum, minimum, and default values, named by the configuration register's name followed by `<`, `>`, and `=` [Cyphal-DT]. For example, `oap.ep.outlet_1.fail_safe_timeout<` holds the largest `failSafeTimeout` the endpoint accepts.

7. Each other config attribute of the device itself is a register named `oap.dev.ATTRIBUTE`, with `ATTRIBUTE` converted as in item 1. For example, the device's `updateChannel` is the register `oap.dev.update_channel`.
8. A config attribute whose name is an extension name [OAP-DC Section 7.9] is a register named `oap.ep.ENDPOINT.ext.NAME`, or `oap.dev.ext.NAME` for an attribute of the device, where `NAME` is the extension name with each hyphen replaced by an underscore. For example, an extension attribute `com.example.night-mode` of endpoint `light_1` is the register `oap.ep.light_1.ext.com.example.night_mode`. Extension names hold no underscores, so the name can be read back, and register names hold no hyphens and no two underscores in a row [Cyphal-DT]. So the name of an extension config attribute carried on the module bus MUST NOT have two hyphens in a row, and its register name MUST NOT be longer than the 255 bytes of a register name.

11.3 Commands

1. A controller sends commands as `oap.Command.0.2` requests to the module's `oap_command` service. The request names the endpoint, the command, and its parameters, using the names defined by [OAP-DC].
2. The module MUST respond with one of the status codes defined in `oap.Command.0.2`. It responds when it has accepted or rejected the command; operations that take time, such as a dose, report their progress through state and events.
3. A module MUST reject commands from any node other than its controller with `STATUS_NOT_CONTROLLER`, except `identify`, which it MUST accept from any node.
4. A module MUST NOT respond with `STATUS_LOCAL_CONTROL` to a stop (Section 5.2), such as a dosing head's hold (set with `enabled false`) [OAP-FS Section 8.2].
5. A command from the controller counts as contact [OAP-FS Section 7.1], whether or not the module accepts it, except as Sections 9.3.5 and 12.3 say. A command that the module accepts, and that changes the endpoint's output, resumes control of the endpoint [OAP-FS Section 7.4.2]. One that only records, clears, shows, or silences something, such as `identify`, `silence`, `refill`, or `resetFault`, does not; the module carries it out, if it accepts it, also during `powerOnDelay` [OAP-FS Section 9.1].
6. A module uses `STATUS_BUSY` for the commands it refuses during `powerOnDelay` [OAP-FS Section 9.1] or a firmware update (Section 13).

11.4 Events

1. A module MUST publish each event defined by [OAP-DC] as `oap.Event.0.1` on the `oap_event` port.
2. The event's `time` field, if it has one, is carried in the message's `timestamp`; its other fields are carried in `fields`.
3. Cyphal publications are not acknowledged, so a controller can miss an event. A module therefore keeps its recent events, as a device of the Device Protocol does [OAP-DP Section 11.6], and publishes again those that its controller asks for. State attributes that events affect, such as a doser's `dosedToday` and an analyzer's `lastResults`, also show what happened.
4. Each event MUST carry in `seq` a sequence number one greater than that of the module's previous event, across all its endpoints, starting from 1. `seq` MUST NOT go back, not even after a restart, a firmware update, or a factory reset, so that a gap shows that events were lost. An event of a module of revision 0.3 or earlier, whose `oap.Event.0.1` has no `seq`, reads as `seq 0` (Section 8.1.4), and such a module keeps no events.
5. A module SHOULD keep at least its 16 most recent events.
6. When its controller sends `COMMAND_SEND_EVENTS` with a sequence number (Section 11.7), the module MUST publish again, in order, each event it keeps whose `seq` is greater, with the `seq`, `timestamp`, and fields it had, and then go on publishing new events.
7. A controller SHOULD send `COMMAND_SEND_EVENTS`, with the last `seq` it received from the module, when it hears again from a module that it had lost (Table 9), once it has configured a module after its own start-up, and when it receives an event whose `seq` is more than one greater than the last, so that it records doses, fills, and results that happened while it was not listening. It sends it only for a module from which it has received an event, and so keeps the last `seq` of each module across its own restarts. It ignores an event whose `seq` it has already received, and sends the command again if the module refuses it, such as while the module checks its controller's unique ID (Section 9.3.5). An event published again whose `seq` is more than one greater than the one asked for shows that the module no longer kept the events between.

For example, an analyzer module runs a nitrate test while its controller restarts, and publishes its `result` and `testEnded` events with `seq` 41 and 42, which nobody receives. The controller, whose last event from the module had `seq` 40, configures the module, and sends `COMMAND_SEND_EVENTS` with the parameter 40; the module publishes events 41 and 42 again, and the controller records the result.

11.5 Faults and Health

1. Device-level faults, including `communication-lost`, are reported in the module's own state message.
2. A module's Heartbeat `health` MUST be `NOMINAL` when no faults are active, and at least `ADVISORY` otherwise. A module SHOULD report `WARNING` when a fault prevents an endpoint from working.
3. A module without output or unit endpoints has no `failSafeTimeout`, and does not report `communication-lost`. The controller detects the loss of such a module from its Heartbeat (Table 9).

11.6 Value Encoding

Attribute values, command parameters, and event fields are carried as `uavcan.register.Value.1.0`, both in messages and in registers.

Table 7 — Value encoding

Device Classes type	Value field	Encoding
boolean	<code>bit</code>	One element.
integer	<code>integer64</code>	One element.
number	<code>real64</code>	One element, in the attribute's unit.
string, enum, parameter, unit	<code>string</code>	UTF-8 text.
timestamp	<code>natural64</code>	One element: network time in microseconds (Section 9.4). Zero means unknown.
date	<code>string</code>	YYYY-MM-DD.
list	<code>string</code>	A JSON array.
object	<code>string</code>	A JSON object.

1. A value MUST NOT be longer than the `uavcan.register.Value.1.0` field allows (256 bytes).
2. In a message, a list or object value that is too long for one `oap.Attribute.0.1` MUST be split across consecutive Attributes with the same name. Each part is a JSON array of some of the items, or a JSON object of some of the members, and the receiver joins the arrays or merges the objects. A member of an object whose value is an array too long for one Attribute is split in the same way: each part gives that member with some of its items, in order, and the receiver joins them. Each item, and every other member, MUST fit in one Attribute. For example, a calibration record [OAP-PI Section 14.9] with three points may be carried as `{"outcome":"completed","parameter":"ph","model":"nernst"}`, then `{"points":[P1]}` and `{"points":[P2,P3]}`, then `{"result":R,"previous":Q,"extensions":X}`, where each of P1, P2, P3, R, Q, and X stands for a JSON value; the receiver gets one object whose `points` has three items. Values stay JSON: an encoding of JSON in binary would save only about a fifth of such a record, whose size is mostly names and units, and would make the bus harder to read (Appendix C, issue 4).
3. A controller converts timestamps to RFC 3339 form, with the local UTC offset, when it records them in the Data Model [OAP-DM].
4. Attributes, parameters, fields, commands, events, and fault codes are carried under their names, unchanged, and those that are extensions under their extension names [OAP-DC Section 7.9]. So an extension name MUST NOT be longer than 64 bytes, the size of the `name` of `oap.Attribute.0.1`, the `command` of `oap.Command.0.2`, the `event` of `oap.Event.0.1`, and `oap.Name.0.1`.

11.7 Node Commands

Table 8 — Node commands

<code>uavcan.node.ExecuteCommand.1.3</code> command	Meaning on the module bus
<code>COMMAND_IDENTIFY</code>	Identify the whole module, as the <code>identify</code> command does for one endpoint.

uavcan.node.ExecuteCommand.1.3 command	Meaning on the module bus
COMMAND_RESTART	Restart the module. Section 9.1 of [OAP-FS] applies after the restart.
COMMAND_STORE_PERSISTENT_STATES	Store the module's registers.
COMMAND_BEGIN_SOFTWARE_UPDATE	Start a firmware update (Section 13).
COMMAND_EMERGENCY_STOP	Put every output and unit endpoint in its inactive state, as a stop without <code>revertAfter</code> to each would: it cancels each pending revert [OAP-DC Section 7.4.7], and ends operations, which never resume [OAP-DC Section 7.14]. It counts as a command from the controller, and so as contact (Table 9). It latches nothing, and an endpoint that a latched stop holds treats it as another latched stop, so keeps the latch [OAP-FS Section 8.4.3].
COMMAND_FACTORY_RESET	Restore every register, including fail-safe configuration, to the manufacturer's defaults. A controller MUST send it only after the owner has confirmed it [OAP-FS Section 13.1].
COMMAND_POWER_OFF	Not supported; modules respond with <code>STATUS_BAD_COMMAND</code> .
COMMAND_RELEASE, 16384 (provisional)	Release the module from its controller [OAP-FS Section 7.5.4]: the module clears <code>oap.controller.id</code> and <code>oap.controller.unique_id</code> , and each output and unit endpoint applies its fail-safe behavior at once, except that one that a latched stop holds keeps it [OAP-FS Section 8.4].
COMMAND_SEND_EVENTS, 16385 (provisional)	Publish again each kept event whose <code>seq</code> is greater than the number in <code>parameter</code> , written in decimal digits (Section 11.4.6). A module responds with <code>STATUS_BAD_PARAMETER</code> to a parameter that is not such a number, or that is larger than a <code>seq</code> can be.

1. A module MUST accept `ExecuteCommand` only from its controller, except `COMMAND_IDENTIFY`, which it MUST accept from any node. While `oap.controller.id` is 65535, it MUST accept `ExecuteCommand` from any node.
2. `COMMAND_RELEASE` and `COMMAND_SEND_EVENTS` are Open Aquarium commands, in the range below 32768 that `uavcan.node.ExecuteCommand.1.3` leaves to vendors. A module MUST carry out `COMMAND_RELEASE` as Table 8 says, and MUST do the same when its controller writes 65535 to `oap.controller.id`, so that a module never has endpoints under the control of no controller. A module that has no controller MUST NOT report `communication-lost`, and accepts a new controller as Section 10.5.1 allows.
3. A module MUST carry out `COMMAND_EMERGENCY_STOP` as Table 8 says, at any time, as it carries out every stop [OAP-FS Section 8.2]. It stops every output, including one that signals alarms, so a controller sends it for `stop-all` only as Security and Access Section 11.8.3 allows, and MUST NOT send it for `stop-all` to a module with an output that an alarm function drives, which `stop-all` leaves to that function [OAP-SA Section 11.8].

11.8 Program Files

A light or a pump that runs its own program can be written a reference to a program too long for one value, which names a file that its controller serves [OAP-LP Section 16.2; OAP-FP Section 15.2].

1. A controller that writes a module's `program` as such a reference MUST serve the file it names, at the path that the Lighting Programs or Flow Programs specification gives it, `oap/lighting/` or `oap/flow/` followed by the hash and `.json`, with `uavcan.file.Read.1.1` and `uavcan.file.GetInfo.0.2`.
2. It MUST serve the file for as long as the module reports a `program` that names it, or the controller's latest write of the module's `program` names it, so that the module can read the file while it loads it, and again after a restart.
3. A module MUST read such a file only from its controller, the node named in `oap.controller.id`.

12 Supervision and Fail-Safe Behavior

Table 9 — Fail-Safe Behavior on the module bus

[OAP-FS] concept	On the module bus
Valid message from the controller (Section 7.1)	A Heartbeat from the node named in <code>oap.controller.id</code> with mode <code>OPERATIONAL</code> , or an <code>oap.Command</code> request or a <code>COMMAND_EMERGENCY_STOP</code> from that node, received while the module

[OAP-FS] concept	On the module bus
	has transmitted a Heartbeat within the last 3 s (item 3). Register writes are not valid messages, because a controller that is starting up writes registers before it controls anything.
Supervision interval (Section 7.1)	The controller's Heartbeat, published at least once per second.
Controller start-up (Section 9.4)	The controller publishes its Heartbeat with mode <code>INITIALIZATION</code> , which does not count as contact.
Loss of a device (Section 7.6)	No Heartbeat from a module for 3 s, the Cyphal offline timeout. The outputs of a lost module are unknown until each one's <code>failSafeTimeout</code> plus 5 s after its last Heartbeat (item 4).
One controller at a time (Section 7.5)	<code>oap.controller.id</code> (Sections 10.5 and 11.3); a controller releases a module with <code>COMMAND_RELEASE</code> (Section 11.7).
<code>communication-lost</code>	Reported in the module's own state message.
Firmware update (Section 11.2)	Heartbeat mode <code>SOFTWARE_UPDATE</code> (Section 13).

1. The controller **MUST** publish its Heartbeat from the same process that runs its control functions [OAP-FS Section 7.1]. If its control functions stop, its Heartbeats **MUST** stop, or **MUST** change to a mode other than `OPERATIONAL`.
2. The **RECOMMENDED** `failSafeTimeout` for module endpoints is 30 s [OAP-FS Section 7.2].
3. A module that has not transmitted a Heartbeat successfully for 3 s, for example because it is in the bus-off state or cannot drive the bus, **MUST NOT** count messages from its controller as contact until it has transmitted one successfully again. A Heartbeat is transmitted successfully when its frame has been sent without error, which on CAN means that another node has acknowledged it [ISO 11898-1]. The controller's Heartbeat reaches every module, so without this rule a module that its controller has lost could keep counting it as contact, and keep its outputs as last commanded, for as long as it can still receive.
4. A controller **MUST** treat the state of each output of a lost module as unknown until that output's `failSafeTimeout` plus 5 s after the last Heartbeat it received from the module, and then as being in its fail-safe behavior, as Section 7.6.3 of [OAP-FS] lets a binding define. The 5 s are the 3 s of item 3, and the 2 s within which an endpoint begins its fail-safe behavior [OAP-FS Section 7.2.3].
5. A controller that has configured modules on its bus (Section 10), and has received no Heartbeat from any of them for 3 s, **MUST** stop publishing Heartbeats with mode `OPERATIONAL`, by stopping its Heartbeats or publishing them with another mode, until it receives a Heartbeat from one of them again. A module counts as configured until the owner removes it from the controller's configuration. Item 3 covers a module that cannot transmit; this item covers a controller that cannot receive, such as one whose receive path has stalled while its CAN hardware still acknowledges frames. Without it, such a controller would treat every module as lost, and its outputs as in their fail-safe behavior (item 4), while every module kept counting its Heartbeats as contact.

For example, a controller receives a module's last Heartbeat at 10:00:00, and the module's heater output has a `failSafeTimeout` of 30 s. The controller treats the module as lost from 10:00:03, and the heater as unknown until 10:00:35; it does not, for example, let a chiller that must not run with the heater start before then. The module, which hears the controller's Heartbeats but cannot transmit, stops counting them as contact by 10:00:03, so its heater output stops by 10:00:35. Had the controller stopped receiving at 10:00:00 instead, it would have published no `OPERATIONAL` Heartbeat after about 10:00:03, and the heater would likewise have stopped by about 10:00:35.

13 Firmware Updates

1. A controller starts an update by sending `COMMAND_BEGIN_SOFTWARE_UPDATE` with the path of the image on its own file server.
2. A module **MUST** refuse with `STATUS_BAD_STATE` while any of its endpoints is dosing, filling, or taking part in a water change, as Fail-Safe Behavior Section 11.2.2 defines them [OAP-FS]: a dosing head running for any reason, such as priming or calibration; a top-off unit filling, or an output whose `loadCategory` is `ro-di` that

- is on, open, or running; or an output whose `loadCategory` is `water-changer` that is on, open, or running, such as an outlet that runs a water change pump.
3. During the update, the module MUST publish its Heartbeat with mode `SOFTWARE_UPDATE`, read the image from the controller with `uavcan.file.Read.1.1`, and keep its outputs as Section 11.2 of [OAP-FS] requires, keeping any latched stop [OAP-FS Section 8.4]. Until the final restart, it MUST accept stops (Section 5.2) [OAP-FS Section 8.2], and SHOULD refuse other commands with `STATUS_BUSY`.
 4. The module MUST check the image's integrity before running it, at least with the CRC-64-WE that Cyphal recommends for `software_image_crc`. The Security and Access specification adds signature checks [OAP-SA Section 15.2].
 5. The module MUST keep its previous firmware until the new firmware has started and confirmed that it works. If the new firmware fails to confirm, or the module restarts before it does, the module MUST return to the previous firmware.
 6. After the update, the controller SHOULD check the module's `GetInfo` `software_version` and `software_image_crc`, and read its device description again [OAP-DC Section 12].
 7. A module SHOULD read its image in order, from its start. Its controller, which serves the image, can then show the update's progress from the offsets the module reads and the image's size, and its end from the module's Heartbeat mode and item 6, so a module reports no progress of its own.

14 Conformance and Testing

14.1 Modules

A conforming module meets every requirement for modules in Sections 7 to 13, implements the Device Classes and Fail-Safe Behavior specifications for its endpoints, and passes the tests in Table 10 that apply to modules.

14.2 Controllers

A conforming controller meets every requirement for controllers in Sections 7 to 13, implements the Fail-Safe Behavior specification, and passes the tests in Table 10 that apply to controllers.

14.3 Test Procedures

These tests are in addition to those of the Fail-Safe Behavior specification, which also apply. Each test names the requirements it checks. Every MUST and MUST NOT of Sections 7 to 13 is checked by at least one test. A pass criterion that checks what a SHOULD asks applies to a product that does it. Some tests need a sample that its maker has prepared, for example with test points from which its supply rails or an output can be short-circuited, or its transmissions stopped.

Table 10 — Test procedures

Test	Requirements	Method	Pass criteria
Bring-up	7.1.1, 8.1, 9.3.1, 9.3.3, 10.1, 10.3, 10.4	Start a controller and three modules at the default bit rates, and record every frame.	Every module is allocated a node-ID and configured within 10 s, with no human action. The controller reads each port's type, sets each port-ID, writes <code>oap.controller.id</code> and <code>oap.controller.unique_id</code> , sends <code>COMMAND_STORE_PERSISTENT_STATES</code> , and then resumes control of each output. No module publishes or serves a port of Table 5 before its port-ID is set. Every frame is an ISO CAN FD frame of a transfer of a data type of Table 2 or of the <code>dsdl</code> directory.
Classic mode	7.1.3, 7.1.4, 9.3.2	Set <code>uavcan.can.bitrate</code> to equal values on every node and restart. Then connect a module that has no node-ID.	The bus works in classic CAN, and no frame is longer than 8 bytes. Each module has the node-ID it had in CAN FD mode, and the new module is allocated one with <code>uavcan.pnp.NodeIDAllocationData.1.0</code> .

Test	Requirements	Method	Pass criteria
Same node-ID	9.3.2, 9.3.3	Restart the modules, then the controller, several times. Then clear each module's stored node-ID, and repeat.	Each module always receives the same node-ID.
Many modules	9.3.6	Connect 32 modules to one controller, simulated ones where real ones are not at hand, and command an output of each.	Every module is allocated a node-ID, configured, and controlled.
Moved module	9.3.5	Connect a configured module, one of whose outputs a latched stop holds, to a different controller's bus, whose controller answers GetInfo only after 10 s. Meanwhile, from that controller, write a register of the module, command one of its outputs on, and send a stop to another. Then, with a module that stays powered on its own bus, restart its controller; then stop the controller's Heartbeats for 5 s, and start a different controller at node-ID 1.	The module asks for the controller's unique ID at least every 2 s until it is answered. Meanwhile, it refuses the write and the command, and carries out the stop. It then clears its node-ID, port-IDs, and controller registers, keeps its fail-safe configuration, puts the stopped endpoint back in its fail-safe behavior, keeps the latched stop, requests a node-ID, and is configured by the new controller. The module that stays powered reads its controller's unique ID again after the restart, and keeps its node-ID; after the replacement, it reads the unique ID again, and is configured by the different controller.
Duplicate node-ID	9.3.4	Connect a module whose stored node-ID is already in use.	The module requests a new node-ID; the other node is not disturbed.
Description	8.1.5, 9.1, 10.2.1 to 10.2.3, 10.2.5	Read <code>oap/description.json</code> from the module, and its GetInfo. Change its configuration, and read the description again.	The description is valid against the Device Classes schema and validator, is UTF-8 JSON of no more than 64 KiB, has endpoint identifiers as Section 10.2.3 requires, gives no state attributes, and names <code>module-bus</code> in <code>specRevisions</code> with the revision the module implements. It is the same when read again. GetInfo gives a unique ID, a reversed domain name as <code>name</code> , and the major and minor numbers of <code>firmwareVersion</code> .
Invalid description	8.1.5, 10.2.4	Connect a module whose description the Device Classes validator rejects. Then connect one whose description names no revision of this Specification in <code>specRevisions</code> , and one that names a revision whose layouts the controller does not support.	The controller never commands the first module, and configures and commands neither of the others.
Registers	7.3.5, 10.5, 11.2	List the module's registers and read each. Write each config register with a value in range, a value out of range, and a value of the wrong type. Then remove the module's power, without sending <code>COMMAND_STORE_PERSISTENT_STATES</code> , and read them again.	The registers of Tables 5 and 6, and one for each config attribute the module implements, exist with the names, types, and flags of Sections 10.3, 10.5, and 11.2. <code>oap.bus.max_current</code> is set. Out-of-range and wrong-type writes are rejected, and the previous value stays. Values written survive the power cut. Registers named with <code><</code> , <code>></code> , and <code>=</code> hold a maximum, a minimum, and a default.
State timing	11.1.1, 11.1.3	Change an output, and watch state messages, also while the output changes faster than 10 times per second.	State is published within 100 ms of the change and at the required minimum rates. Measured from their timestamps, no more than 10 messages from one endpoint fall within any 0.99 s, which allows for adjustments of the module's clock.
State contents	11.1.2, 11.1.4, 11.6	Record the state messages of every endpoint and of the module, with a list or object state attribute longer than one Attribute where the module has one, such as a calibration record with several points; change it, and record for 70 s more.	Each message gives the endpoint's <code>controlSource</code> , for outputs and units, its faults, and every state attribute it implements, under its name and encoded as Table 7 says, except that an attribute longer than one Attribute may be left out: it appears within 100 ms of each change, and at least every 60 s. No value is longer than 256 bytes; longer lists and objects, and arrays within objects, are split, and join to the full value. No extension name is longer than 64 bytes.

Test	Requirements	Method	Pass criteria
Events	11.4, 11.6, 11.7	Make the module produce each event that its endpoints define, such as <code>doseCompleted</code> . Then make it produce 20 more events while the bus's other nodes receive none of them, for example with a test hook, and send <code>COMMAND_SEND_EVENTS</code> with the <code>seq</code> of the last event received before the 20. Then restart the module, and make it produce one more event; update its firmware, and make it produce one more.	Each event is published on <code>oap_event</code> , with its <code>time</code> in <code>timestamp</code> , its other fields encoded as Table 7 says, and a <code>seq</code> one greater than the previous event's, across the restart and the update. No extension name is longer than 64 bytes. On the request, the module publishes again, in order, every event it keeps after the <code>seq</code> sent, at least the 16 most recent of the 20, with their <code>seq</code> , <code>timestamp</code> , and fields unchanged.
Missed events	11.4.7	Make a module produce events while the controller has lost it, and while the controller restarts; then skip the publication of one event, for example with a test hook.	The controller sends <code>COMMAND_SEND_EVENTS</code> , with the <code>seq</code> of the last event it received, when it hears from the module again, when it configures it after the restart, and when it receives the event after the one skipped. It records each event once.
Command statuses	11.3.2, 11.3.4, 11.3.6	Send commands that should produce each status code, including stops while a local control is in effect and during <code>powerOnDelay</code> : set with <code>on false</code> , a stop of one attribute of the inactive state, such as a dosing head's hold (set with <code>enabled false</code>); and a latched stop.	The module responds with the right status, and partial execution never occurs. No stop gets <code>STATUS_LOCAL_CONTROL</code> .
Commands from other nodes	10.5.1 to 10.5.3, 11.3.3, 11.7.1	Send commands, register writes, and node commands from a node other than the controller. Then let the module lose contact, with one output's <code>failSafeTimeout</code> 60 s longer than another's, and, once it reports <code>communication-lost</code> , write <code>oap.controller.id</code> and <code>oap.controller.unique_id</code> from that node. Repeat after contact has returned, while the endpoints are still in their fail-safe behavior. Then, on a module without output or unit endpoints, stop its controller's Heartbeats, and write the controller registers from the other node after 10 s, and again after 35 s.	They are rejected, except <code>identify</code> and <code>COMMAND_IDENTIFY</code> . Once the module reports <code>communication-lost</code> , the writes to its controller registers are accepted, the other node becomes its controller, and the output with the longer timeout enters its fail-safe behavior at once; after contact has returned, they are rejected. The module without outputs rejects the writes after 10 s, and accepts them after 35 s.
Node commands	11.7	Send each command of Table 8 from the controller: <code>COMMAND_EMERGENCY_STOP</code> with one output held by a latched stop, one with a pending revert, and a dose in progress; and <code>COMMAND_SEND_EVENTS</code> with the parameter <code>x</code> . Ask the controller to factory reset the module, and decline the confirmation; then accept it. Then have the controller run <code>stop-all</code> while two modules, one with an output that an alarm function drives, delay their responses to stops by more than 1 s.	Each command does what Table 8 says, and <code>COMMAND_POWER_OFF</code> gets <code>STATUS_BAD_COMMAND</code> . After <code>COMMAND_EMERGENCY_STOP</code> , every output and unit endpoint is in its inactive state, the latched output still reports <code>latched true</code> , the pending revert never falls due, the dose does not resume, and the module does not report <code>communication-lost</code> . <code>COMMAND_SEND_EVENTS</code> with <code>x</code> gets <code>STATUS_BAD_PARAMETER</code> . The controller sends <code>COMMAND_FACTORY_RESET</code> only after the owner has confirmed it. It sends <code>COMMAND_EMERGENCY_STOP</code> to the module without an alarm output, and never to the other.
Release	11.7.2	With an output on, and another held by a latched stop, send <code>COMMAND_RELEASE</code> from the controller, then wait longer than the outputs' <code>failSafeTimeout</code> . Repeat with the controller writing 65535 to <code>oap.controller.id</code> . Then configure the module from another node.	Every output applies its fail-safe behavior at once, except the latched one, which stays stopped; both controller registers are cleared, and <code>communication-lost</code> is never reported. The other node's configuration is accepted.
Commands that do not resume control	11.3.5	With an endpoint in its fail-safe behavior and contact restored, send <code>identify</code> , and every command of its class that only records, clears, shows, or silences something; repeat during <code>powerOnDelay</code> .	Each command the endpoint accepts is carried out, and <code>controlSource</code> stays <code>fail-safe</code> .

Test	Requirements	Method	Pass criteria
Health	11.5.2	Make an endpoint report a fault, then clear it.	The module's Heartbeat health is NOMINAL without the fault, and at least ADVISORY with it.
Supervision	12.1	Stop the controller's Heartbeats, then set its mode to INITIALIZATION, then stop its control functions while its bus driver keeps running.	Endpoints apply fail-safe behavior after failSafeTimeout in each case. When the control functions stop, the Heartbeats stop, or leave OPERATIONAL.
Lost module	12.3, 12.4	With a second module on the bus, which keeps transmitting, so that the controller meets Section 12.5 by hearing it, command an output on, with a failSafeTimeout of 10 s. Stop the module's transmissions while it still receives, for example by holding its transceiver's transmit input recessive, and restore them after 30 s.	The controller treats the module as lost from 3 s after its last Heartbeat, and the output as unknown until 15 s after it, and then as in its fail-safe behavior. The output applies its fail-safe behavior no later than 15 s after the module's last Heartbeat, although the controller's Heartbeats still reach the module. After the module transmits again, the controller resumes control.
Controller that cannot receive	12.5	With two modules configured, each with an output on, stop the controller from reading received frames for 60 s, for example with a test hook, while its CAN hardware keeps acknowledging them; then let it read them again. Then remove every module from the controller's configuration, and disconnect them.	The controller's Heartbeats stop, or leave mode OPERATIONAL, within 3 s of the last Heartbeat it received from either module, and each output applies its fail-safe behavior no later than its failSafeTimeout plus 5 s after it. When the controller reads frames again, its Heartbeats return to OPERATIONAL, and it resumes control. With no module configured, its Heartbeats stay OPERATIONAL.
Priority	8.3	From another node, load the bus with more than 0.5 s of state traffic, and send stops: set with on false, a dosing head's hold, set with enabled false, a latched stop, and COMMAND_EMERGENCY_STOP. Record every frame of the other tests too.	Each stop and its response are delivered ahead of the queued state traffic, and each response arrives within 20 ms. This value is provisional. Every frame has the priority of Table 3.
Network time	9.4	Record time synchronization messages, and the modules' timestamps, with the controller's clock valid, then not valid; then stop the controller. Then, on a module that runs a local program that needs the time of day, with synchronization messages, compare the module's clock with network time every minute for an hour, for example with a test hook or an output its program changes at a set time, and read clockUncertainty where the module implements it. Stop the synchronization messages, and keep the module running until its documented drift could have moved its clock 60 s.	Synchronization messages come once per second while the clock is valid, and not while it is not. Module timestamps are network time while they come, and zero within 3 s after the last of them. While they come, the module's clock is within 1 ms of network time, and its clockUncertainty at most 1 ms. After they stop, the module keeps running its program, and reports clockInvalid only once its drift could have moved its clock 60 s.
Firmware update	13.3 to 13.5, 13.7	Update a module, one of whose outputs a latched stop holds. Then cut its power partway through an update; give it a corrupt image; give it firmware that never confirms; and send a stop, and a stop of one attribute of the inactive state, during an update.	The update succeeds, with the Heartbeat in SOFTWARE_UPDATE mode, the image read in order, outputs kept as Fail-Safe Behavior Section 11.2 requires, and the latched stop kept; the stops are carried out. After the power cut, and with the firmware that never confirms, the module returns to its previous firmware. The corrupt image is never run.
Update refused	13.2	Start an update while an endpoint is dosing, while one primes, while a valve whose loadCategory is ro-di is open, while a switch whose loadCategory is ro-di is on, and while a switch whose loadCategory is water-changer is on.	The module responds with STATUS_BAD_STATE each time.
Bus-off recovery	7.5.2, 8.4.1	Briefly short CAN H to CAN L.	All nodes recover without a restart.

Test	Requirements	Method	Pass criteria
Connecting while powered	7.3.4, 7.5.1	Connect and disconnect a module repeatedly while the bus is running.	No other node restarts or changes an output, and no node stays bus-off.
Reverse polarity	7.3.3	Apply reversed supply polarity to a module.	The module is not damaged.
Supply range	7.3.2, 7.3.5	Run a module at 18 V and at 30 V at its connector, with each of its outputs on and off, and measure its current.	It works at both voltages, and, once connected, never draws more than its <code>oap.bus.max_current</code> .
Back-feed	7.3.6	Power a bus from two supplying nodes, and switch off the supply of each in turn. Then power a module that does not supply the bus through any other input it has, with the bus's supply off.	The bus stays powered from the other supply, and no current flows back into the supply that is off. The module puts no voltage on the bus's power conductors.
Module fault	7.6.1, 7.6.2	On a sample prepared by its maker, short-circuit the module's supply rails behind its protection, and then one of its outputs, while it shares a bus with a controller and two other modules whose outputs are on.	No other node restarts or changes an output, and the others keep communicating. Once its protection has acted, the module draws no more than twice its <code>oap.bus.max_current</code> .
Bus power short	7.6.3, 7.6.4, 7.6.6	With every supplying node of the bus connected, their documented limits adding up to no more than 4 A, short-circuit bus power to ground at a module's connector for 60 s, then remove the short. Repeat at each end of the bus.	Each supplying node delivers no more than 4 A, and no more than its documented limit; none is damaged, and each keeps running its own functions; a controller that supplies the bus warns its owners. Bus power returns within 5 s of the end of the short, without the supplying node restarting, and every module rejoins and is configured with no human action.
CAN line fault	7.6.5	With the bus running, connect CAN H to bus power for 60 s, then CAN L. Then apply -36 V and +36 V, relative to ground, to each line in turn, for 60 s each.	No node is damaged, and every node resumes communicating, without restarting, after each fault ends.
Termination	7.2.5	Write <code>oap.can.termination</code> on a module.	The resistance between CAN H and CAN L changes accordingly.
Program files	11.8	On a light or a pump module that runs its own program, write a program too long for one value, so that the controller writes a reference, and record every file transfer. Restart the module after it has loaded the program. Then write another program too long for one value.	The controller serves the file at the path that the reference names, and the module reads it only from the controller, and runs the program; after the restart, the module reads it again, and runs it. The controller serves the first file until the module reports the second program.
Cable and connectors	7.1.5, 7.2.1 to 7.2.4, 7.4, 7.6.7	Inspect the data sheets of the nodes' transceivers, and the cables, connectors, and installation instructions of the controller and the modules.	Transceivers comply with ISO 11898-2 and are rated for CAN FD at 2 Mbit/s. The instructions give a single line, terminated at each end, of no more than 20 m, with no stub longer than 0.3 m, CAN H and CAN L on a twisted pair of about 120 Ω , and bus power on separate conductors. Connectors carry CAN H, CAN L, bus power, and ground, are keyed by function, lock, resist splashing, and carry the current of Section 7.6.7 on their contacts for power and ground. Each supplying node's documentation gives the most current it delivers into the bus, and says that the supplying nodes of one bus together must deliver no more than 4 A.

15 Security Considerations

1. Cyphal version 1.0 does not authenticate messages. The module bus relies on physical access control: anyone who can reach the bus inside the equipment cabinet can also unplug the equipment [OAP-FS Section 15].
2. `oap.controller.id` prevents mistakes, such as two controllers on one bus. It does not prevent a deliberate attack.

3. A later revision may add message authentication for CAN FD, where frames have room for it. Until then, the hash of a program file shows a file damaged on the way, not one that someone on the bus wrote together with its reference (Section 11.8).
4. Firmware images are checked before they run (Section 13.4). Signature checks are defined by the Security and Access specification [OAP-SA Section 15].
5. A node on the bus that is faulty or compromised can short-circuit or load the bus, as a damaged cable can; Section 7.6 limits what that does to other nodes, and the outputs of every module that loses contact fail safe.

Appendix A — Approval Checklist

Complete this checklist before requesting Working Group approval. Link each item to the evidence.

Item	Status	Details
Section 2 describes compliance with all five tenets.	Done	
A public review period of at least 30 days has been announced and completed.	Not started	
At least two independent implementations exist, at least one from an organization not represented among the Authors.	Not started	
The implementations have passed the test procedures in Section 14.3 and in the Fail-Safe Behavior specification.	Not started	
The provisional values have been confirmed with measurements on the breadboard and the first circuit boards.	Not started	
Every normative reference is publicly available.	Done	ISO 11898 is available for purchase; all others are free.
The data types are published and pass the DSDL check.	Done	
The Open Issues appendix is empty, or each open issue is explicitly deferred.	Not started	
No unresolved Exclusion Notices for this Specification are recorded in Notices.md .	Done	
The Maintainer has recorded consensus to approve.	Not started	

Appendix B — A Module Joining the Bus (Informative)

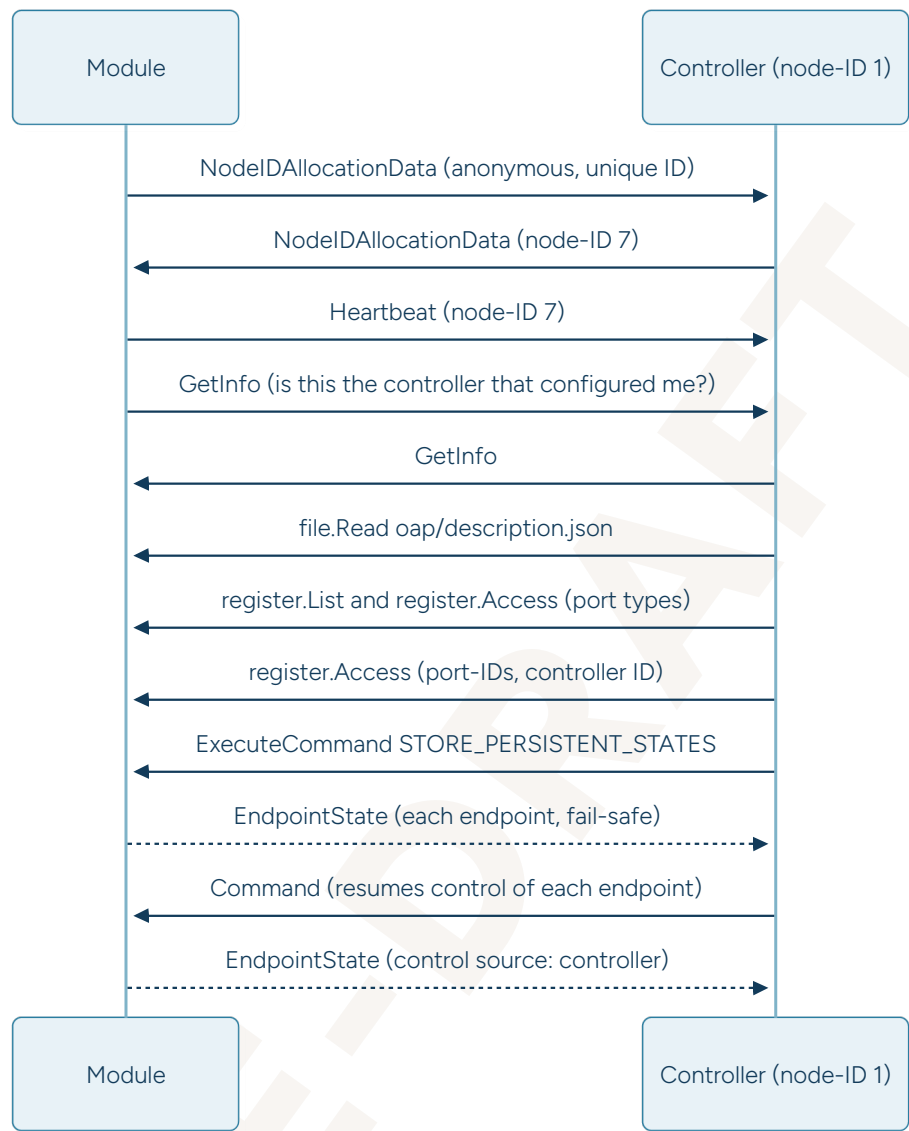


Figure 2 — A new module joins the bus and is configured

Appendix C — Open Issues (Informative)

Issues 1 to 16 were open in revision 0.1. The Project Lead decided issue 3, in part, on 16 October 2026, accepting the recommendation. Each issue says how a revision resolves it, or what it waits for. Issues 17 to 20 came from the breadboard firmware and the module bus simulator, and the Project Lead decided them, and issues 4 and 11, on 17 October 2026; revision 0.3 applies the decisions. On 17 October 2026 the Project Lead also froze the scope of revision 1.0, and decided issue 8; revision 0.4 applies the decision, and resolves issues 14 and 15, which other specifications raised. Issues 5, 6, 9, 10, 11, and 13 are deferred beyond revision 1.0, and are tracked outside this Specification. Issues 1, 2, 3, 7, and 12 stay open: each waits on the bench tests. Issue numbers do not change, because other specifications refer to them.

1. **Connector and pinout.** *Open: the Project Lead chooses them after the bench tests.* The candidates are M12, small circular connectors, and RJ45, with current ratings per contact [P0001]. Whichever is chosen meets Section 7.4, which revision 0.2 extends to the current that the supplying nodes of a bus may deliver together (Section 7.6.7).

2. **Bit timing and length.** *Deferred until breadboard tests 9 and 11 [Breadboard].* The data bit rate, sample points, maximum length, and stub length are provisional until those tests are done. Test 9 runs the bus on 10 m and on 20 m of cable, the longest that Section 7.2.3 allows, with stubs of 0.3 m to each module, and test 11 repeats it with CAN FD.
3. **Power budget and protection.** *Resolved in part in revision 0.2: the values of Section 7.6 are adopted, provisionally; the power budget stays open.* Revision 0.2 adds the protection of Section 7.6, with these provisional values:
 - twice a module's `oap.bus.max_current` as the most that a faulty module draws once its protection has acted, which leaves room for the module's own load and keeps one fault from taking much of a supply;
 - 4 A as the most that the supplying nodes of a bus deliver into it, about 100 W at 24 V, enough for 32 modules that each draw 125 mA, and the current that the contacts and conductors of the bus must carry (Section 7.4);
 - 5 s for a supplying node to restore bus power after a short circuit or an overload ends, so that modules restart soon after the fault is cleared; and
 - -36 V to +36 V on CAN H and CAN L: the highest voltage at a module's connector, 30 V (Section 7.3.2), with a margin of 20 %, either way round, because a module connected with its supply reversed sees the bus voltage below its ground. The Project Lead decided on 16 October 2026 to adopt these values provisionally, as recommended, and to confirm them on the first circuit boards, because the breadboard's development boards have no protection of their own [Breadboard]. Where several nodes supply one bus, the cable beside a short circuit carries the current of every supplying node on one side of it, more than Section 7.4.5 rates it for. The Project Lead therefore also decided that the supplying nodes of one bus deliver no more than 4 A together, and that a second supply for redundancy is combined behind one supplying node's limit, as a controller with two supply inputs does (Section 7.6.7). A power module may supply the bus too, within that total [OAP-PM Section 7.2.4]. Still open is the current available per connector, which waits for the connector (issue 1).
4. **Compact encodings.** *Resolved in revision 0.3, as the Project Lead decided on 17 October 2026: named attributes, with JSON for lists and objects, stay.* Named attributes are simple but verbose. A binary encoding of JSON saves only about a fifth of a typical structured value: a calibration record of three points takes 1046 bytes as JSON and 853 in the binary encoding, because most of it is names and units, and decimal numbers grow as binary floating-point numbers. Data types per structure would be far smaller, but would need a new version for every change of the Device Classes, and could not carry extension attributes. Revision 0.3 sends long attributes only when they change and at least every 60 s instead (Section 11.1.4), which removes most of their load.
5. **Moved and duplicate modules.** *Open until the bench tests.* Whether listening before using a stored node-ID (Section 9.3) is reliable in practice is not yet known.
6. **Event delivery.** *Resolved in revision 0.4, as the Project Lead decided on 17 October 2026.* Cyphal publications are not acknowledged, so analyzers' `result` and `testEnded` events of tests run while the controller was away were lost on the module bus, which the Device Protocol avoids by keeping recent events [OAP-DP Section 11.6] (Device Classes Appendix D, issue 19). Revision 0.4 has a module keep its recent events with sequence numbers, as the Device Protocol has devices do, and publish again those that its controller asks for with `COMMAND_SEND_EVENTS` when contact returns (Section 11.4). The sequence number is a new last field of `oap.Event.0.1`, which a controller reads as 0 from a module of an earlier revision (Section 8.1.4).
7. **Measuring termination.** *Open until the bench tests.* Whether the Termination test measures with bus power on, and whether a switched termination must keep its state without power, are open.
8. **Program files.** *Resolved in revision 0.4: the controller serves them (Section 11.8).* The Lighting Programs specification writes a fixture program longer than one register to a light as a reference, and has the controller serve the program as the file `oap/lighting/` followed by its SHA-256 hash and `.json`, read with `uavcan.file.Read.1.1` as firmware images are [OAP-LP Section 16.2]. The Flow Programs specification does the same for pump programs, as files under `oap/flow/` [OAP-FP Section 15.2]. Revision 0.4 has the controller serve such a file for as long as the module reports a program that names it, or the controller's latest write names it, so that the module can read it again after a restart, and has the module read it only

from its controller. Raised by the Lighting Programs specification (Appendix D, issue 4) and the Flow Programs specification (Appendix D, issue 4).

9. **Time accuracy.** *Resolved in revision 0.4 (Section 9.4).* The Flow Programs specification asks pumps that run their own programs to keep their clocks within 20 ms of their controller's, so that pumps in anti-sync stay half a period apart [OAP-FP Section 11.4]. This Specification should state the accuracy that network time gives a module (Section 9.4). It should also say when a module's own clock stays valid without synchronization messages, so that a module keeps running its program after it loses contact, and reports the Device Classes fault `clock-invalid` only when its clock is not valid; and it could carry a module's `clockUncertainty` (Device Classes Sections 7.6 and 8.1). Revision 0.4 requires a module to keep its clock within 1 ms (provisional) of network time while it receives synchronization messages, which Cyphal's method of sending each message with the time of the previous one allows; takes the validity of a module's clock from Fail-Safe Behavior Section 10.5, which revision 0.4 of that specification gives every device; carries `clockUncertainty` as a state attribute; and says that a module takes the local time zone from the `posixTimeZone` of the program its controller writes, as the Dosing specification asked (its Appendix D, issue 14). Raised by the Flow Programs specification (its Section 11.4).
10. **Firmware update progress.** *Resolved in revision 0.2: the controller follows the transfer, and the module reports nothing more.* Proposal 0001 promised progress reporting for module firmware updates [P0001]. A module reads its image from its controller, which therefore knows how much of the image has been read and how large it is; the module's Heartbeat mode shows when the transfer has ended and the module is installing, and its `GetInfo` shows the result (Section 13). The module bus simulator follows updates this way, and needs nothing more of a module than that it reads its image in order, which Section 13.7 recommends.
11. **Taking over.** *Resolved in revision 0.3, as the Project Lead decided on 17 October 2026.* Section 10.5.2 let another node write the controller registers while an endpoint was in its fail-safe behavior after a loss of contact, but Fail-Safe Behavior Section 7.5.2 lets another controller take control only while the device reports `communication-lost`. Endpoints stay in their fail-safe behavior after contact returns, so the two disagreed then. Revision 0.3 follows Fail-Safe Behavior.
12. **Commands that resume control.** *Resolved in revision 0.3, as the Project Lead decided on 17 October 2026.* Section 11.3.5 said that every command a module accepts, other than `identify` and `silence`, resumes control, but Fail-Safe Behavior Section 7.4.2 says that commands that only record, clear, show, or silence something, such as `refill` and `resetFault`, do not. Revision 0.3 follows Fail-Safe Behavior.
13. **Long attributes in state messages.** *Resolved in revision 0.3, as the Project Lead decided on 17 October 2026.* Every state message carried every state attribute, so a calibration record of about 1 KB went out with each message of a probe, twice a second on the breadboard firmware: about 4 % of a bus at 1 Mbit/s, and much more at lower bit rates. Revision 0.3 lets an endpoint send such an attribute when it changes and at least every 60 s (Section 11.1.4).
14. **Release.** *Resolved in revision 0.3, as the Project Lead decided on 17 October 2026.* Revision 0.2 did not say how a controller releases a module [OAP-FS Section 7.5.4]; the simulator and the breadboard firmware treated a write of 65535 to `oap.controller.id` as one. Revision 0.3 adds `COMMAND_RELEASE`, which is deliberate, answers with a status, and clears both controller registers, and keeps the write as a way to the same end (Section 11.7).