



OPEN AQUARIUM PROJECT SPECIFICATION

Device Classes

PRE-DRAFT

Revision	0.5
Status	Pre-Draft. A candidate document submitted for consideration.
Date	2026-10-18
Identifier	device-classes
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. Comment by opening an issue with the `device-classes` 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 class definitions, JSON Schema, and example files 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 Specification is published under the Community Specification License 1.0, and its class definitions, schema, and examples under the Apache License 2.0. The class definitions are a machine-readable file, `classes.json`, from which implementers can generate firmware and application code. Anyone may implement the classes, in open source or proprietary products, without permission or payment.

2.2 Interoperability

Each class describes one kind of equipment the same way for every maker, so any controller can operate any device that implements it. The same classes are used whether a device is a module on the wired module bus or a device on the local network. Readings use the parameters and units of the Open Aquarium Data Model, so they can be recorded without translation. The classes were checked against the features of four widely used commercial controller ecosystems. Each device lists the optional attributes and commands it implements, so a controller knows what it can use before it uses it (Section 12). A replacement device takes over the automation of the one it replaces, whoever made either (Section 7.12). A light can give the spectrum and the photon flux of each of its channels, so that any controller can estimate how much light it gives, and at which wavelengths, whoever made it (Section 7.15).

2.3 Local Control

Every output endpoint defines its behavior when it loses contact with its controller, and devices can run lighting, flow, dosing, and test programs by themselves. A controller can hold such a program, and return the device to it, with one command (Section 7.13). A device whose `failSafeAction` is `continue` keeps what it was

last told when it loses its controller, so a program its controller held stays held, and one it ran keeps running (Section 8.3.2). Nothing in this Specification requires a network service or account.

2.4 Safety

Every output and unit endpoint has a fail-safe action, a fail-safe state, a timeout, and a power-on behavior (Section 8.3), used as defined by the Fail-Safe Behavior specification. Every fail-safe state gives a value for each attribute of the class's inactive state, and entering it cancels any temporary change still pending, so that a heater held off for a while stays off when its controller fails (Sections 7.4 and 8.3). Temporary changes, such as a pump paused for feeding, can be made to end on the device itself, and a series of temporary changes ends in the state that came before them (Section 7.4). A stop with no known end, such as one that keeps a return pump from pumping water onto the floor during a leak, can be latched, so that the endpoint keeps it when its controller fails and through power cuts (Section 7.4.13). A light that comes back on after a power cut fades up, and a pump ramps up, rather than jumping, whether it restores its level or enters a fail-safe state that lights it or runs it (Section 8.3). One that enters such a fail-safe state at any other time, such as when its controller fails, fades or ramps to it too, but stops at once when its fail-safe state stops it. A cap on a light's own program, such as one that dims it while the water is too warm, reaches it at once, and stays when a scene or another temporary change ends (Section 7.13). Devices enforce their own safety limits, such as maximum fill times, maximum dose volumes, and temperature cutoffs, even when a controller commands otherwise (Section 7.8). Every dosing head has a dose limit, a daily limit, and a longest run without a rest, and one that primes has a daily limit on priming; every top-off unit has a minimum interval between fills, and refuses a manual fill while it is not enabled; and a top-off unit with its own leak sensor stops on a leak until a person resets it. An outlet with a backup thermostat holds its heater or chiller off past a cutoff, and regulates the water by itself only while it follows its program, which an interlock's stop takes away (Section 7.13). A dose, a feed, or an advance that is stopped or cut short never starts again by itself (Section 7.14). Supervised inputs and current-loop inputs report broken wiring instead of a false reading, sensor readings carry a quality that controllers must check before using them for control, and valves declare their unpowered position. Battery-powered devices report a weak battery before they stop, and devices that need the time say when their clock is not valid (Section 7.6).

2.5 Sustainability

Because any conforming controller can operate any conforming device, equipment can outlive the controller it was bought with. Operating hours, consumable tracking, and fault reporting support maintenance and repair. A replacement device takes over the settings the owner chose for the one it replaces (Section 7.12), and devices report the end of their makers' security support (Section 8.1).

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	2026-10-15	Optional elements listed in device descriptions; rules for devices and controllers of either revision; revert targets kept across temporary commands, and at power-up; replacing a device; returning lights and pumps to their programs, and holding dosing heads' programs; calibration points for dosing heads; temporary setpoints for heaters and chillers; program attributes defined by the Lighting Programs, Flow Programs, Dosing, and Water Testing specifications; analyzer tests that end with an event, raw results, standards, method types, and reagent and waste capacities; sensor ports, selectable parameters, calibration records, and probe and compensation settings; top-off start delay, overrun, daily limit, and learning settings; 4–20 mA inputs; battery, clock, and firmware information; device sounders; the faults <code>clock-invalid</code> , <code>battery-weak</code> , <code>leak</code> , and <code>daily-limit</code> ; test procedures. Descriptions of revision 0.1 remain valid (Section 7.11).	Trent Scholl
0.3	2026-10-15	<code>failSafeAction continue</code> keeps everything last commanded, including <code>enabled</code> , holds, and a pending revert, and a unit runs its own logic only while enabled; entering <code>failSafeState</code> cancels a pending revert; every <code>failSafeState</code> gives its class's inactive state, which <code>classes.json</code> now records; <code>runProgram</code> with <code>revertAfter</code> keeps a pending	Trent Scholl

Revision	Date	Description	Authors
		revert's target; a revert that falls due fades or ramps as its command did; a fade or ramp to the inactive state is honored, and never delays a stop; lights fade, and pumps ramp, to a level they restore after power-up; a dosing head's hold survives a power cut; operations that are stopped or interrupted never resume by themselves, and which stops carry <code>revertAfter</code>; primes and calibration runs no longer than <code>maxContinuousRun</code>, a daily limit on primes (<code>maxDailyPrimeTime</code>), and primes and calibration runs only when a person asks; doses longer than <code>maxContinuousRun</code> refused; <code>enabled</code> for analyzers; <code>measurementInterval</code> for sensors; descriptions without state attributes, and config values in them that are the manufacturer's defaults; analyzer results that are observations once screened; <code>overtemperature</code> required while a light dims for heat; commands that a condition holding an output off accepts, and carries out when it ends; who may command and configure a device; a test procedure for every requirement; references cited by tag; one sentence per line; every open issue resolved, decided, or deferred. Descriptions of revisions 0.1 and 0.2 remain valid (Section 7.11).	
0.4	2026-10-17	A cap of a light's own program, <code>programMaxLevel</code>, which a controller writes without changing what the light shows, which <code>runProgram</code>'s <code>maxLevel</code> also sets, and which a revert does not change; fades of color temperature that change evenly in mireds; lights that fade, and pumps that ramp, up to a fail-safe state that lights or runs them after power-up, and to one they enter at any other time from what they show, while they still reach their inactive state at once; the spectral photon distribution and the photon flux of each channel of a light, <code>spectrum</code> and <code>photonFlux</code>, and how a controller uses them; a test for each. Descriptions of revisions 0.1 to 0.3 remain valid (Section 7.11).	Trent Scholl
0.5	2026-10-18	The Project Lead's decisions of 17 October 2026 applied: latched stops, sent with <code>latch</code>, a parameter of every <code>set</code> of an output or unit class, and reported as <code>latched</code>, which an endpoint keeps through loss of contact and power cuts until a controller ends them; a command that starts or ends an operation leaves a pending revert as it is; the attributes of an outlet's rating, switching device, unpowered state, endurance, and count of operations, the device attribute <code>mainsPresent</code>, the fault <code>cutoff</code>, and the feature <code>backup-thermostat</code> of <code>switch</code>, in place of the Power Module's provisional extensions, with a program that <code>runProgram</code> returns an outlet to and <code>set</code> takes it off; <code>maxContinuousRun</code> REQUIRED for every dosing head; a revert to a light's program restores its effects; <code>resetCalibration</code> described as the Probe Interface defines it; sleepy devices that report changes of faults within 1 s. Also: entering a fail-safe state replaces a command that a condition holds back; a top-off unit that is not enabled refuses a manual fill; <code>prime</code> and <code>calibrate</code> sent only when a person asks applies to dosing heads; the validator accepts a prime limit whose value a description gives; a test for each; references to the final revisions of the other specifications; issue 27 deferred beyond revision 1.0. Descriptions of revisions 0.1 to 0.4 remain valid (Section 7.11).	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	3
2.5	Sustainability	3
	Revision History	3
3	Scope	7
4	References	7
4.1	Normative References	7
4.2	Informative References	8
5	Terms, Definitions, and Conventions	8
5.1	Requirements Language	8
5.2	Terms and Definitions	8
5.3	Notation	9
6	Introduction	9
6.1	Purpose	9
6.2	Design Principles	9
6.3	Model Overview	9
7	Device Model	10
7.1	Devices and Endpoints	10
7.2	Classes and Features	10
7.3	Attributes	11
7.4	Commands	11
7.5	Events	13
7.6	Faults	13
7.7	Parameters and Units	13
7.8	Device-Enforced Limits	13
7.9	Extensions	14
7.10	Bindings	14
7.11	Revisions and Compatibility	14
7.12	Replacing a Device	18
7.13	Local Programs	18
7.14	Operations	20
7.15	Light Output	20
8	Common Elements	21
8.1	Device Attributes	21
8.2	Endpoint Attributes	22
8.3	Output Attributes	22
8.4	Fault Codes	24
9	Output Classes	25
9.1	Switch (switch)	25
9.2	Level Output (level-output)	27
9.3	Pump (pump)	27
9.4	Light (light)	29
9.5	Heater (heater)	31
9.6	Chiller (chiller)	32
9.7	Doser (doser)	32
9.8	Valve (valve)	34
9.9	Feeder (feeder)	35
10	Unit Classes	35
10.1	Auto Top-Off (auto-top-off)	35
10.2	Filter Roller (filter-roller)	37
10.3	Analyzer (analyzer)	37
11	Input Classes	39
11.1	Sensor (sensor)	39

11.2	Level Switch (<code>level-switch</code>)	42
11.3	Leak Sensor (<code>leak-sensor</code>)	43
11.4	Contact Input (<code>contact-input</code>)	43
11.5	Analog Input (<code>analog-input</code>)	43
12	Device Description	44
13	Conformance	45
13.1	Devices	45
13.2	Controllers	45
13.3	Machine-Readable Definitions and Tools	45
13.4	Test Procedures	46
14	Safety and Security Considerations	51
Appendix A — Approval Checklist		52
Appendix B — Equipment Mapping (Informative)		52
Appendix C — Examples (Informative)		53
Appendix D — Open Issues (Informative)		54

3 Scope

This Specification defines:

- a model of devices, endpoints, classes, and features;
- classes for aquarium equipment, with their attributes, commands, events, and faults;
- attributes common to every device, every endpoint, and every output;
- the device description, which tells a controller what a device is and what it can do;
- how devices and controllers of different revisions of this Specification work together; and
- how a replacement device takes over from the one it replaces.

This Specification does not define:

- how devices are connected or how messages are encoded, which the Module Bus and Device Protocol specifications define;
- automation, such as rules, schedules, and modes, which the Automation specification defines;
- the content of lighting, flow, dosing, and test programs, which their own specifications define;
- fail-safe behavior, which the Fail-Safe Behavior specification defines; or
- electrical interfaces of probes, which the [Probe Interface](#) specification defines.

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.
- **[JSON Schema]** "JSON Schema: A Media Type for Describing JSON Documents" and "JSON Schema Validation", draft 2020-12, <https://json-schema.org/draft/2020-12>.
- **[OAP-DM]** Open Aquarium Project, "Open Aquarium Data Model", Revision 0.4, [specs/data-model/spec.md](#).
- **[OAP-FS]** Open Aquarium Project, "Open Aquarium Fail-Safe Behavior", Revision 0.4, [specs/fail-safe-behavior/spec.md](#).

The specifications of kinds of equipment are normative for the features and attributes that refer to them, such as the program formats of the `local-program` and `wave` features and the backup thermostat of the `backup-thermostat` feature, and the Security and Access specification for who may command and configure a device (Section 14):

- **[OAP-DO]** Open Aquarium Project, "Open Aquarium Dosing", Revision 0.3, [specs/dosing/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-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-SA]** Open Aquarium Project, "Open Aquarium Security and Access", Revision 0.3, [specs/security-and-access/spec.md](#).
- **[OAP-WL]** Open Aquarium Project, "Open Aquarium Water Level and Water Change", Revision 0.3, [specs/water-level-and-water-change/spec.md](#).
- **[OAP-WT]** Open Aquarium Project, "Open Aquarium Water Testing", Revision 0.3, [specs/water-testing/spec.md](#).

4.2 Informative References

- **[OAP-AU]** Open Aquarium Project, “Open Aquarium Automation”, Revision 0.3, [specs/automation/spec.md](#).
- **[OAP-AN]** Open Aquarium Project, “Open Aquarium Alarms and Notifications”, Revision 0.3, [specs/alarms-and-notifications/spec.md](#).
- **[OAP-MB]** Open Aquarium Project, “Open Aquarium Module Bus”, Revision 0.4, [specs/module-bus/spec.md](#).
- **[OAP-DP]** Open Aquarium Project, “Open Aquarium Device Protocol”, Revision 0.3, [specs/device-protocol/spec.md](#).
- **[OAP-MM]** Open Aquarium Project, “Open Aquarium Matter Mapping”, Revision 0.3, [specs/matter-mapping/spec.md](#).
- **[IEC 60381-1]** IEC 60381-1:1982, “Analogue signals for process control systems — Part 1: Direct current signals”.
- **[NE 43]** NAMUR, Recommendation NE 43, on the signal levels by which transmitters with 4–20 mA outputs report failures. The levels used in Section 11.5 were taken from manufacturers’ documentation that cites it, not from its text.

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 Fail-Safe Behavior specification [OAP-FS], such as inactive state, stop, hold, and latched stop, have the same meaning here; each class gives its inactive state (Section 8.3).

Device: A physical product that a controller communicates with: a module on the module bus, a device on the local network, or a controller itself for its own built-in inputs and outputs.

Endpoint: One function of a device, such as one outlet of a power strip or one head of a doser.

Class: The definition of one kind of endpoint: its attributes, commands, events, and faults.

Feature: An optional group of attributes, commands, and events within a class.

Attribute: A named value of a device or endpoint, such as a setting or a reading.

Command: A request from a controller for an endpoint to do something.

Event: A report from an endpoint that something happened.

Fault: A condition that stops an endpoint or device from working normally, identified by a fault code.

Controller: The device or software that operates other devices through their endpoints.

Binding: A specification that defines how this model is carried over a particular connection, such as the Module Bus or Device Protocol specification.

Fail-safe behavior: What an output endpoint does when it loses contact with its controller.

Optional element: An OPTIONAL attribute or command, which a device or endpoint may implement, and lists in its device description when it does (Section 12).

Local program: A program that an endpoint runs by itself, such as a light’s fixture program or a dosing head’s head program (Section 7.13).

Revert target: The state an endpoint returns to when a pending `revertAfter` falls due (Section 7.4).

Operation: Something an endpoint does for a time once a command starts it, such as a dose, a prime, or a calibration run of a dosing head, a feed, or an advance of a filter roll (Section 7.14).

Port: A connector of a device for a probe or other equipment, such as an electrode port.

Replacement: A device that takes the place of another in a system, and takes over its automation (Section 7.12).

Photon flux: The number of photons that a source of light emits each second, in $\mu\text{mol/s}$ (Section 7.15).

Spectral photon flux: The photon flux of a source of light at one wavelength, per nanometre.

Spectral photon distribution: How a source's photon flux is spread over wavelengths: its spectral photon flux at each wavelength (Section 7.15).

Photosynthetic photon flux: The photon flux from 400 nm to 700 nm.

5.3 Notation

Class, feature, attribute, command, and event names appear in `monospace`. Units are Data Model unit codes [OAP-DM]. In the class tables, the Feature column shows which feature an element belongs to; “—” means it belongs to the class itself. Elements that a revision after 0.1 added are marked with it, as in “Added in 0.3”, and elements that it made REQUIRED are marked as such (Section 7.11).

References to other Open Aquarium specifications name them, as in “Fail-Safe Behavior Section 8.1”, or use their tags from Section 4, as in “[OAP-FS Section 8.1]”. A section number with a third part refers to a numbered item of that section: “Section 7.4.6” means item 6 of Section 7.4.

6 Introduction

6.1 Purpose

Every commercial controller ecosystem defines its own device types: its own idea of what an outlet, a dosing pump, or a wave pump is, and what it can do. A controller from one maker cannot operate a device from another, except through adapters written for each product.

This Specification defines those device types once, for everyone. A controller that implements a class can operate any device that implements it, whether the device is a module on the wired bus or a product on the local network, and whoever made it.

6.2 Design Principles

1. **Describe function, not products.** A class describes what equipment does, such as switching, pumping, or dosing, so that very different products can share one class.
2. **Small required core, optional features.** Each class requires only what every product of its kind can do. Everything else is a feature that a device declares.
3. **One transport-independent model.** The same classes are used over the module bus and the local network.
4. **Fixed units.** Each attribute has one unit. Devices never negotiate units, and readings use Data Model parameters and units directly.
5. **Safety at the device.** Fail-safe behavior and safety limits live in the device, so they work when the controller or the network does not.

6.3 Model Overview

A device has device-level attributes and one or more endpoints. Each endpoint implements one class and may declare features of that class. A controller reads the device description (Section 12) to learn what the device is and which optional elements it implements, then reads and writes attributes, sends commands, and receives events and faults through a binding.

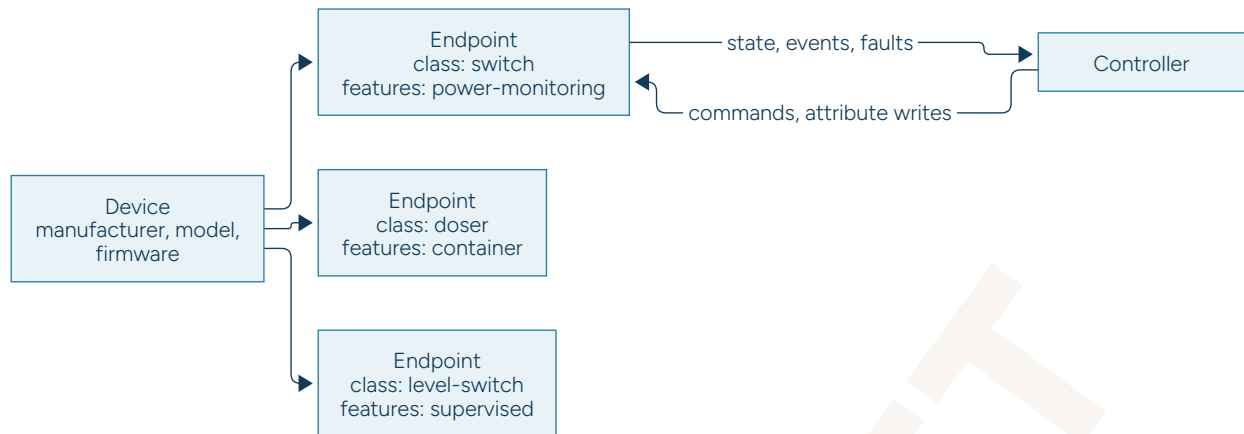


Figure 1 — Devices, endpoints, and classes

7 Device Model

7.1 Devices and Endpoints

1. A device **MUST** present one endpoint for each of its functions. For example, a power strip presents one `switch` endpoint per outlet, and a two-head doser with a level sensor in each container presents two `doser` endpoints and two `level-switch` endpoints.
2. Each endpoint **MUST** implement exactly one class.
3. Endpoint identifiers **MUST** be unique within the device and **MUST** stay the same across restarts and firmware updates. They **SHOULD** match labels on the device, such as outlet numbers. They **SHOULD** consist of lowercase letters, digits, and single underscores, begin with a letter, not end with an underscore, and be no longer than 32 characters, so that they are valid in every binding; the module bus requires this.
4. A sensor that measures more than one parameter, such as a combined pH and temperature probe, **MUST** present one `sensor` endpoint per parameter.

7.2 Classes and Features

1. An endpoint MUST implement every REQUIRED attribute, command, and event of its class, and of each feature listed in its `features` attribute.
2. An endpoint MUST NOT list a feature it does not implement.
3. An endpoint MAY implement OPTIONAL elements. An element belonging to a feature is required or optional only when the endpoint implements that feature.
4. A device of revision 0.2 or later MUST show in its device description which optional elements it and each of its endpoints implement (Section 12), and MUST refuse an OPTIONAL command, or a write to an OPTIONAL attribute, that it does not show, as one it does not implement.
5. A controller MUST NOT rely on an optional element that a device or endpoint does not show. A device of revision 0.1 lists none, so a controller MAY try an optional element of such a device, and treats a refusal as the element's absence.
6. Classes are grouped by kind:
 - **output classes** (Section 9) act on equipment;
 - **unit classes** (Section 10) are equipment that runs its own control logic, such as a top-off unit or an analyzer; and
 - **input classes** (Section 11) report what they sense.
7. The output attributes (Section 8.3) apply to every endpoint of an output or unit class.

7.3 Attributes

Table 1 — Attribute types

Type	Values
boolean	true or false.
integer	A whole number.
number	A number. Where the attribute has a unit, it is shown after the type, for example "number (W)".
string	UTF-8 text.
enum	One of the listed strings.
timestamp	A Data Model Timestamp: an RFC 3339 date and time with a UTC offset.
date	A Data Model Date.
parameter	A Data Model parameter identifier.
unit	A Data Model unit code.
list	An ordered list. Where the items are objects, their fields are listed after the class's attribute table.
object	A JSON object, described with the attribute. Where its members are defined, they are listed as fields after the attribute table.

Table 2 — Attribute access

Access	Meaning
fixed	Describes the hardware. Set by the manufacturer, included in the device description (Section 12), and never changed by a controller.
config	A setting that a controller or the owner can change. A device description gives its manufacturer's default, if any (Section 12).
state	The device's current condition or latest reading. Read-only, and never in a device description.

1. A numeric attribute **MUST** use the unit shown for it. Conversion to other units for display is done by consumers, as described in the Data Model.
2. An attribute in % ranges from 0 to 100 unless its definition gives another range.
3. A device **MUST** keep the values of config attributes across power loss.
4. A device **MUST** reject a write to a config attribute whose value is outside the permitted range, and **MUST** keep the previous value.

7.4 Commands

1. A command has a name and named parameters.
2. A device **MUST** either carry out a command completely or reject it. It **MUST NOT** carry out part of a command.
3. A device **MUST** reject a command that is missing a **REQUIRED** parameter, has an unknown parameter, or has a value outside the permitted range.
4. Bindings define how commands, results, and errors are encoded.
5. Every `set`, `setColorTemperature`, and `runProgram` command accepts an optional `revertAfter` parameter, in seconds. With it, the change is temporary: after `revertAfter`, the endpoint returns to its revert target, without any further message from the controller. Controllers use this for temporary changes, such as pausing a pump for feeding, so that the change ends even if the controller fails [OAP-AU Section 17.5]. Commands that start or end an operation, such as `dose`, `feed`, and the `stop` of a doser, a feeder, or a filter roller, have no `revertAfter`: an operation ends by itself, and one that a command ends never resumes (Section 7.14).
6. The revert target is the state the endpoint had before the command, unless a revert was already pending. Then the new command keeps the pending revert's target, and only the time of the revert changes. This holds for every command with `revertAfter`, `runProgram` included. So a temporary state that a controller extends, or one temporary state that follows another, ends in the state that came before the first of them. For an endpoint that was following its local program, the revert target is the program.
7. A command without `revertAfter` that changes the endpoint's output, such as a `set`, a `runProgram`, or a `pulse`, cancels a pending revert. A command that starts or ends an operation, such as `dose`, `prime`,

`calibrate`, `feed`, `advance`, `fill`, `test`, or `abort`, or the `stop` of a doser, a feeder, or a filter roller, leaves a pending revert as it is (Section 7.14). So a dosing head's `stop`, sent with a temporary hold of its program, never takes away the hold's end, whichever of the two arrives first. A command that does not change the output, such as `identify`, `silence`, or `refill`, also leaves a pending revert as it is. Entering `failSafeState` cancels a pending revert, so that a revert target that is not safe never follows the fail-safe state [OAP-FS Section 7.3]. With `failSafeAction continue`, a pending revert stays, and falls due as it would have (Section 8.3.2).

8. An endpoint that loses power while a revert is pending, and whose `powerOnBehavior` is `last`, restores the revert's target after power-up, not the temporary state [OAP-FS Section 9.1].
9. An endpoint of revision 0.1 returns to the state it had before the latest command, whether or not a revert was pending (Section 7.11). A controller sends such an endpoint a temporary state that follows another as Automation Section 17.5 describes.
10. A `fadeTime` or `rampTime` changes each level or speed evenly, in a straight line from its value when the endpoint accepts the command to the new value, over that many seconds. A later command starts from the value at the moment it is accepted. A fade of a light's color temperature, such as one of `setColorTemperature`, changes the intensity in the same way, and the color temperature evenly in mireds, one million divided by the color temperature in kelvin [OAP-LP Section 8.4]: with m_0 the mireds at the start and m_1 at the end of a fade of T seconds, the color temperature t seconds into it is $1\,000\,000 / (m_0 + (m_1 - m_0) \times t / T)$ K. A light of revision 0.3 or earlier may change its color temperature in another way (Section 7.11).
11. A fade or ramp in progress never delays a stop [OAP-FS Section 8.2]: a command that puts an endpoint in its inactive state ends any fade or ramp in progress as soon as the endpoint accepts it. Without a `fadeTime` or `rampTime`, the endpoint reaches its inactive state at once. With one, the endpoint fades or ramps to its inactive state over that time, as the command asks, such as at the end of a sunset.
12. When a revert falls due, the endpoint fades or ramps to its revert target over the `fadeTime`, `rampTime`, or `transition` of the latest command with `revertAfter`, if that command gave one. If it gave none, a light or a pump whose revert target is its program returns to it with the program's restore fade or restore ramp, as `runProgram` does by default [OAP-LP Section 13.4; OAP-FP Section 12.3], and any other endpoint returns at once.
13. A controller latches a stop that has no known end by sending it with `latch true`, a parameter of the `set` of every output and unit class from revision 0.5 [OAP-FS Section 8.4]. A device MUST reject `latch true` on a `set` that is not such a stop: one that gives no attribute of the endpoint's inactive state (Section 8.3.1), that gives one a value other than its inactive value, or that has `revertAfter`. An endpoint that accepts a latched stop MUST report `latched true` until the latch ends, and MUST keep each attribute that the stop gave at its inactive value, except while a local control is in effect, as Fail-Safe Behavior Section 8.4 says: in place of its fail-safe behavior and its `powerOnBehavior`, and across power loss, until a command from its controller that changes its output, other than another latched stop, ends the latch. Its program, or a unit's own logic, starts nothing meanwhile that would change those attributes. An endpoint whose class has no `set`, such as a feeder, cannot be latched, and reports `latched false`.

For example, a controller pauses a running pump for feeding with `revertAfter 600`, and 300 s later the owner extends the pause, so the controller sends the same stop with `revertAfter 600` again. 900 s after the first command, the pump runs again, as it did before the pause. A pump of revision 0.1 would instead return to the state it had before the second command, and stay stopped.

Likewise, a controller shows a scene on a light that runs its own program with `set`, `fadeTime 120`, and `revertAfter 3600`. If the controller fails during the scene, the light fades back to its program over 120 s when the revert falls due, rather than jumping to it.

While a leak sensor is wet, an interlock stops a return pump with `set`, `running false`, and `latch true`. If the controller then fails, the pump stays stopped after its `failSafeTimeout`, although its fail-safe state runs it, and after any power cut, until a controller sends it a command that runs it, or a stop without `latch`.

A light that fades with `setColorTemperature` from 2700 K to 6500 K over 60 s shows about 3815 K after 30 s, where its mireds are halfway from 370.4 to 153.8, rather than 4600 K. Equal steps in mireds look about equally large, so the fade looks even, as the color temperature programs of the Lighting Programs specification do.

7.5 Events

An event reports something that happened, such as a dose finishing or a button being pressed. Changes to state attributes are not events.

Each binding defines how and how often state is reported (Module Bus Section 11.1; Device Protocol Section 11.5). Every binding MUST:

1. report a change of `controlSource`, of faults, or of an output's state promptly: both bindings report it within 100 ms, except that a sleepy device of the Device Protocol, which has no output or unit endpoints, reports a change of its faults within 1 s (provisional), as it reports a change of `leak`, `wet`, or `closed` [OAP-DP Section 14.3];
2. report every endpoint's state at least once within an interval that it defines, whether or not anything changed; and
3. carry every state attribute that the endpoint implements in each report, so that one report replaces any that were missed.

This Specification sets no change thresholds for readings. A binding that can lose events SHOULD let a controller recover what it missed: the Device Protocol keeps a device's recent events (its Section 11.6), and on the module bus, state attributes such as a doser's `dosedToday` and an analyzer's `lastResults` show what happened (Module Bus Section 11.4).

7.6 Faults

1. A device lists its active faults in the device's or endpoint's `faults` attribute, using codes from Section 8.4 or extension names.
2. A device MUST add a fault when its condition begins and remove it when the condition ends, except where a class says a fault stays until it is cleared by a command, such as `resetFault`.
3. A device MUST report faults whether or not its endpoints are under the control of a controller.
4. A device that needs the time of day, such as one that runs a local program or a schedule, MUST report `clock-invalid` among its device faults while its clock is not valid [OAP-FS Section 10.5].
5. A device that can run on a battery MUST report `battery-weak` among its device faults while its battery needs charging or replacing soon, and at the latest once `batteryLevel` has fallen to 10 % (provisional).
6. A device that sounds or shows an alarm of its own for an endpoint's faults SHOULD implement `sounding` and `silence` for that endpoint (Section 8.2), so that an owner can silence every alarm in one place [OAP-AN Section 14.4].

How faults become alarms is defined by the Alarms and Notifications specification.

7.7 Parameters and Units

Parameter identifiers and unit codes are those of the Data Model registry [OAP-DM]. A `sensor` endpoint's reading corresponds directly to a Data Model Observation: `parameter`, `value`, `unit`, `qualifier`, and `updated` (the observation's `time`).

7.8 Device-Enforced Limits

A device MUST enforce its own safety limits even when a controller commands otherwise. These include, where the device has them, maximum run and fill times, maximum dose volumes, temperature cutoffs, and current limits. From revision 0.2, every dosing head has `maxDoseVolume` and `maxDailyVolume`, and every top-off unit `minFillInterval`. From revision 0.3, every dosing head that primes has `maxDailyPrimeTime`, and a head refuses a run longer than its `maxContinuousRun` (Section 7.14).

1. A command that would exceed a limit MUST be rejected.
2. An operation in progress MUST stop when it reaches a limit.
3. A condition that holds an output off until it ends, such as a top-off unit's `daily-limit` or a power module's backup-thermostat cutoff [OAP-PM Section 11.4], is not a limit that a command would exceed. While it lasts, the endpoint MUST accept a command that the condition holds back, keep it as what its controller last commanded, and carry it out when the condition ends, unless a later command has replaced it, or the endpoint has entered `failSafeState` since [OAP-FS Section 7.3]. A stop takes effect at once, as always

(Section 7.4.11). So a heater outlet commanded on while its backup thermostat's cutoff holds it off, whose controller then fails, enters its fail-safe state, off, and stays off when the cutoff ends.

Controllers apply their own limits too. A device's limits are a second, independent layer of protection.

7.9 Extensions

1. Classes, features, attributes, commands, events, and fault codes that this Specification does not define MUST use extension names, as defined in Section 7.6 of the Data Model, for example a class named `com.example.coral-camera`.
2. Extension attributes of a standard class are placed in the endpoint's `extensions` object.
3. Consumers MUST ignore classes, features, attributes, events, and faults they do not understand.

7.10 Bindings

This Specification does not depend on any transport. The Module Bus specification [OAP-MB] defines how modules carry this model over CAN, the Device Protocol specification [OAP-DP] defines how network devices carry it over IP, and the Matter Mapping specification [OAP-MM] defines how some classes appear in Matter. Bindings MUST preserve the names, types, units, and meanings defined here.

7.11 Revisions and Compatibility

Before revision 1.0, any revision of this Specification may make incompatible changes. From revision 1.0 onward, a revision with the same major number as an earlier one will be backward compatible with it.

Revisions 0.2 to 0.5 add features, attributes, commands, parameters, events, fields, values, and fault codes, which `classes.json` marks with `since`, `valuesSince`, and `faultsSince`, and make some OPTIONAL attributes REQUIRED, which it marks with `requiredSince`. Table 3 lists them, and Table 4 the elements whose meaning they change.

1. A device declares the revision it implements in its description's `oapDeviceClasses`, and as `device-classes` in its `specRevisions`; the two MUST be equal (Section 12).
2. A device implements none of what revisions later than its own added, and need not implement what they made REQUIRED. Its description MUST NOT use what later revisions added.
3. A controller MUST also operate devices of revisions earlier than its own, by the rules of their revisions for those devices: for example, the revert target of revision 0.1 (Section 7.4.9); for an analyzer of revision 0.2, which has no `enabled`, its `state` (Section 7.13); for a light of revision 0.3, which has no `programMaxLevel`, the cap that only `runProgram` sets (Section 7.13.7); and, for a device of revision 0.4 or earlier, which has no `latch`, a stop that it does not latch (Section 7.4.13).
4. A controller cannot validate a description of a revision later than its own, so it does not operate a device of that revision [OAP-MB Section 10.2.4]. For example, a controller of revision 0.1 cannot validate a description of revision 0.2, which lists optional elements.

Table 3 — What revisions 0.2 to 0.5 add, and make REQUIRED

Revision	Where	Added	Now REQUIRED
0.2	Device attributes (Section 8.1)	<code>powerSource</code> , <code>batteryLevel</code> , <code>batteryVoltage</code> , <code>clockUncertainty</code> , <code>updateChannel</code> , <code>securityVersion</code> , <code>firmwareSigner</code> , <code>updateFeed</code> , <code>updateKeys</code> , <code>supportEnds</code> , <code>automaticInstall</code> , <code>lastUpdate</code>	—
0.2	Endpoint attributes and commands (Section 8.2)	<code>sounding</code> , <code>silence</code>	—
0.2	Fault codes (Section 8.4)	<code>clock-invalid</code> , <code>battery-weak</code> , <code>leak</code> , <code>daily-limit</code>	—
0.2	<code>pump</code> (Section 9.3)	<code>flowCurve</code> , <code>set parameter rampTime</code> , <code>reversePause</code> , <code>programRunning</code> , <code>runProgram</code> , <code>flowRateSource</code>	—
0.2	<code>light</code> (Section 9.4)	<code>channels</code> field <code>minLevel</code> , <code>channels</code> field <code>resolution</code> , <code>setColorTemperature</code> parameter <code>revertAfter</code> , <code>programRunning</code> , <code>runProgram</code> , <code>thermalScale</code>	—
0.2	<code>heater</code> (Section 9.5)	<code>set parameter setpoint</code>	—

Revision	Where	Added	Now REQUIRED
0.2	chiller (Section 9.6)	set parameter setpoint	—
0.2	doser (Section 9.7)	calibrate parameter rate, enabled, set, feature calibration-points	maxDoseVolume, maxDailyVolume, dosedToday
0.2	auto-top-off (Section 10.1)	startDelay, overrun, maxDailyFillTime, learningFactor, learningSamples, faults daily-limit and leak	minFillInterval
0.2	analyzer (Section 10.3)	methods, lastResults field raw, test parameter standard, result field raw, result field standard, event testEnded, reagents field tests, reagents field parameters, wasteTests, corrections field unit	—
0.2	sensor (Section 11.1)	port, probeNominal, tdsFactor, lastCalibration, calibratePoint parameter solution, compensationMethod, compensationCoefficient, compensationSalinity, feature selectable	—
0.2	level-switch (Section 11.2)	position values drain and reservoir-empty	—
0.2	analog-input (Section 11.5)	feature current-loop, fault wiring-fault	—
0.3	doser (Section 9.7)	maxDailyPrimeTime	—
0.3	analyzer (Section 10.3)	enabled	—
0.3	sensor (Section 11.1)	measurementInterval	—
0.4	light (Section 9.4)	channels field spectrum, channels field photonFlux, programMaxLevel	—
0.5	Device attributes (Section 8.1)	mainsPresent	—
0.5	Output attributes (Section 8.3)	latched	—
0.5	Fault codes (Section 8.4)	cutoff	—
0.5	switch (Section 9.1)	ratedCurrent, switchingDevice, unpoweredState, endurance, switchCount, set parameter latch, feature backup-thermostat, fault cutoff	—
0.5	level-output (Section 9.2)	set parameter latch	—
0.5	pump (Section 9.3)	set parameter latch	—
0.5	light (Section 9.4)	set parameter latch	—
0.5	heater (Section 9.5)	set parameter latch	—
0.5	chiller (Section 9.6)	set parameter latch	—
0.5	doser (Section 9.7)	set parameter latch	maxContinuousRun
0.5	valve (Section 9.8)	set parameter latch	—
0.5	auto-top-off (Section 10.1)	set parameter latch	—
0.5	analyzer (Section 10.3)	set parameter latch	—

Table 4 — Changes of meaning in revisions 0.2 to 0.5

Element	Changed in	Before	After
revertAfter (Section 7.4)	0.2	The endpoint returns to the state it had before the latest command.	A command sent while a revert is pending keeps that revert's target.
powerOnBehavior last (Section 8.3)	0.2	The endpoint restores the state it had before it lost power.	With a revert pending, it restores the revert's target.
fadeTime (Section 7.4)	0.2	—	Fades evenly, in a straight line.

Element	Changed in	Before	After
thermalDimming of light	0.2	—	While it is true, the light reports overtemperature.
reverse feature of pump	0.2	—	The pump stops, and pauses for reversePause, before it changes direction.
syncGroup of pump	0.2	Pumps with the same group run their programs together.	It names the pump's group; pumps keep in step by their clocks.
program of light, pump, doser, and analyzer	0.2	The program, as its specification defines it.	Also how it is encoded and written (Section 7.13).
dosedToday, containerRemaining, and prime of doser	0.2	—	Primes and calibration runs come out of the container but do not count as dosed, and dosedToday survives power loss.
lastResults of analyzer	0.2	The latest result of each parameter.	The latest result of each parameter from the system's water, not from a standard.
failSafeAction continue (Section 8.3)	0.3	The endpoint continues with its last commanded state or local program, and a unit keeps running its own logic.	The endpoint keeps everything last commanded, including enabled, holds, and a pending revert, and runs its program, or a unit its own logic, only while that lets it.
failSafeState (Sections 7.4 and 8.3)	0.3	Values of the class's state attributes.	Values that include each attribute of the class's inactive state; entering it cancels a pending revert.
runProgram with revertAfter (Section 7.4)	0.3	It cancels any pending revert.	A pending revert keeps its target.
A revert that falls due (Section 7.4)	0.3	The endpoint returns to its revert target at once.	It fades or ramps as the latest command with revertAfter did, or, back to a program, with the program's restore fade or ramp.
A command to the inactive state with a fade or ramp (Section 7.4)	0.3	It takes effect at once.	It fades or ramps as it asks; a stop still ends a fade or ramp in progress at once.
powerOnBehavior last of light and pump (Section 8.3)	0.3	The endpoint restores its level or speed at once.	A light fades, and a pump ramps, up to it.
enabled of doser (Section 8.3)	0.3	—	It survives power loss.
maxContinuousRun of doser (Section 7.14)	0.3	The head enforces its own rests.	The head refuses a dose, prime, or calibration run longer than it.
prime and calibrate of doser (Section 7.14)	0.3	No limit of their own.	Each runs no longer than maxContinuousRun, primes no longer in a day than maxDailyPrimeTime, and a controller sends them only when a person asks.
result of analyzer	0.3	It can be recorded directly as an observation.	It is an observation once a controller has screened it.
An endpoint's attributes in a description (Section 12)	0.3	It may give current values of config and state attributes.	It gives no state attributes, and config values in it are the manufacturer's defaults.
A command that a condition holding an output off holds back (Section 7.8)	0.3	Not stated.	Accepted, and carried out when the condition ends.
setColorTemperature with fadeTime (Section 7.4)	0.4	Fades evenly, without saying how the color temperature changes.	The color temperature changes evenly in mireds, and the intensity evenly.
maxLevel of runProgram of light (Section 7.13)	0.4	It caps the program until the next runProgram, by default 100 %, and a revert back to the program restores the cap it had.	It sets programMaxLevel, which a revert does not change; without it, the cap stays as it is.
A failSafeState that runs a pump or lights a light (Section 8.3)	0.4	Entered at once.	Reached evenly, over the times for restoring a level: from 0 % after power-up, and otherwise from what the endpoint

Element	Changed in	Before	After
			shows or runs; its inactive state is still reached at once.
A command without <code>revertAfter</code> that starts or ends an operation, such as <code>dose</code> or a doser's <code>stop</code> (Section 7.4)	0.5	It cancels a pending revert.	It leaves a pending revert as it is.
A change of the faults of a sleepy device (Section 7.5)	0.5	Reported within 100 ms, as every change of faults.	Reported within 1 s.
A command that a condition holding an output off holds back (Section 7.8)	0.5	Carried out when the condition ends, unless a later command has replaced it.	Also not carried out once the endpoint has entered <code>failSafeState</code> .
effects of <code>runProgram</code> of <code>light</code> (Section 7.13)	0.5	<code>false</code> lasts until the next <code>runProgram</code> , whatever reverts fall due.	A revert that returns the light to its program restores its effects.
<code>resetCalibration</code> of <code>sensor</code>	0.5	Restores the factory calibration.	Restores a smart probe's factory calibration, or the probe type's nominal values for another probe, as the Probe Interface specification defines.
<code>fill</code> of <code>auto-top-off</code>	0.5	Not stated for a unit that is not enabled.	A unit that is not enabled refuses it.

Before revision 0.2, other specifications of the project carried some elements under provisional extension names. Table 5 lists those that revision 0.2 replaces. A consumer SHOULD read each as its replacement, and a device of revision 0.2 or later SHOULD implement the replacement instead. The specifications that defined the provisional names move to the replacements in their next revisions, and keep the provisional names only for reading devices of revision 0.1.

Table 5 — Provisional extension names that revision 0.2 replaces

Before revision 0.2	Replacement	Where it was defined
<code>org.openaquariumproject.calibration-points</code> in a doser endpoint's extensions	The <code>calibration-points</code> feature and its <code>calibrationPoints</code>	[OAP-DO Section 8.3]
Command <code>org.openaquariumproject.set-program</code> , and <code>org.openaquariumproject.program-enabled</code> in a doser endpoint's extensions	<code>set</code> and <code>enabled</code> of the <code>local-program</code> feature of <code>doser</code>	[OAP-DO Section 15.3]
Command <code>org.openaquariumproject.run-program</code> , and <code>org.openaquariumproject.program-running</code> in a light endpoint's extensions	<code>runProgram</code> and <code>programRunning</code> of the <code>local-program</code> feature of <code>light</code>	[OAP-LP Section 16.4]
Command <code>org.openaquariumproject.run-program</code> , and <code>org.openaquariumproject.program-running</code> in a pump endpoint's extensions	<code>runProgram</code> and <code>programRunning</code> of the <code>wave</code> feature of <code>pump</code>	[OAP-FP Section 15.4]
<code>org.openaquariumproject.probe-calibration</code> in a sensor endpoint's extensions	<code>lastCalibration</code> , as a Data Model Calibration Record	[OAP-PI Section 14.9]
<code>org.openaquariumproject.probe-nominal</code> in a sensor endpoint's extensions	<code>probeNominal</code> , <code>tdsFactor</code> , and <code>compensationSalinity</code>	[OAP-PI Section 7.9]
<code>org.openaquariumproject.compensation-method</code> and <code>org.openaquariumproject.compensation-coefficient</code> in a sensor endpoint's extensions	<code>compensationMethod</code> and <code>compensationCoefficient</code>	[OAP-PI Section 7.9]
<code>org.openaquariumproject.firmware</code> in a description's device extensions	The device attributes <code>updateChannel</code> (its member <code>channel</code>), <code>securityVersion</code> , <code>firmwareSigner</code> (its member <code>signer</code>), <code>updateFeed</code> , <code>updateKeys</code> , <code>supportEnds</code> , and <code>automaticInstall</code>	[OAP-SA Section 15.9]

7.12 Replacing a Device

An owner replaces a device when it fails, or with a better one, and expects the new device to do what the old one did. The Automation specification separates what a configuration does from the endpoints that carry it out: an output or input names a class and features, and a binding names its endpoint [OAP-AU Section 8.5]. Roles therefore belong to the configuration, not to endpoints, and a replacement takes over by being bound in its predecessor's place. Endpoints describe their own use with `loadCategory`, `position`, `parameter`, and `port`, which help a controller find the right one.

1. A controller SHOULD offer to move the bindings of a device that is lost or removed to another device. For each binding, it proposes an endpoint of the new device that implements the class and features the output or input needs, and that measures the parameter an input names [OAP-AU Section 8.5.6].
2. When the new device has the same manufacturer and model as the old one, the controller proposes the endpoints with the same identifiers. Otherwise it proposes endpoints whose `loadCategory`, `position`, `parameter`, and `port` match, in the order of their identifiers.
3. The owner MUST confirm each binding a controller proposes, because only the owner knows what is plugged in where.
4. Before it resumes control of an endpoint of the new device, the controller MUST write to it the settings the owner chose for the endpoint it replaces: its fail-safe configuration, `loadCategory`, `name`, and the config attributes that describe the system rather than the hardware, such as limits, positions, and programs. A new device starts with its manufacturer's defaults, and fail-safe configuration MUST be the owner's choice [OAP-FS Section 13].
5. The controller MUST NOT copy values that describe the old hardware or its wear: calibrations, such as `calibratedFlowRate`, `calibrationPoints`, and a probe's calibration; counters, such as `energy` and `operatingHours`; and learned values, such as `learnedFillTime`. It asks the owner to calibrate the new device where its class has a calibration, and to check settings that depend on the hardware, such as `pulsesPerLitre`.
6. Where the old device tracked a container or a reservoir, the controller SHOULD give the new device its last estimate with the `refill` command, because the estimate describes the container, not the device.
7. A smart probe moved to the new device brings its own calibration [OAP-PI Section 16].
8. When the device is itself equipment of the system, such as a heater or a doser, the new device is new Data Model equipment: the controller SHOULD set the old Equipment's `removed`, and record a `maintenance` event, so that history before and after the replacement stays apart [OAP-DM Section 8.3].

For example, an owner replaces a failed two-outlet power strip with a four-outlet strip of another make. The controller proposes the new strip's first two outlets for the heater and the return pump, the owner confirms them, and the controller writes each outlet's fail-safe configuration and `loadCategory` before it switches them.

7.13 Local Programs

Some endpoints run programs of their own, defined by the specifications of their kinds of equipment. Table 6 shows how a controller holds them and returns them to their programs.

Table 6 — Local programs

Class	Feature	Program defined by	Taken off or held by	Returned by	Reported by
light	local-program	[OAP-LP] Section 16	set, <code>setColorTemperature</code>	runProgram	programRunning
pump	wave	[OAP-FP] Section 15	set	runProgram	programRunning
doser	local-program	[OAP-DO] Section 15	set with enabled false	set with enabled true	enabled
analyzer	local-program	[OAP-WT] Section 14	set with enabled false	set with enabled true	enabled; before revision 0.3, <code>state</code>
switch	backup-thermostat	[OAP-PM] Section 11.5	set	runProgram	programRunning

1. An endpoint's `program`, where it has one, holds its program, as JSON text without whitespace between tokens, as the specification in Table 6 defines it; an empty `program` means that the endpoint follows commands.
2. An endpoint MUST reject a write of a program it cannot run in full, and keep its previous program (Section 7.3).
3. A program too long for one value of the endpoint's binding is written as a reference to a file that the controller serves, where the specification in Table 6 allows it. The endpoint keeps reporting its previous `program` until it has read and checked the file.
4. A command that takes an endpoint off its program, with `revertAfter`, returns it to the program when the revert falls due (Section 7.4.6), with the fade or ramp of Section 7.4.12.
5. To resume control of an endpoint without changing what it does [OAP-FS Section 7.4], a controller sends `runProgram` to a light, a pump, or an outlet that follows its program, with the parameters that keep what it shows, such as its current cap, and `set` with the current `enabled` to a doser, analyzer, or top-off unit. An analyzer of revision 0.2, which has no `enabled`, is enabled while its `state` is other than `disabled`.
6. An endpoint whose `failSafeAction` is `continue` runs its program, after it loses contact, only while what its controller last commanded lets it (Section 8.3.2).
7. A light with `local-program` keeps the cap of its program in `programMaxLevel`: while the light shows its program, every channel, and the intensity, is at most the cap. A controller writes `programMaxLevel` to change the cap without changing what the light shows, so that a cap, such as one from dimming on a condition [OAP-LP Section 12.3], reaches the light during a scene, a stop, or a hold, and applies whenever the light shows its program. `runProgram` with `maxLevel` also sets it, and `runProgram` without `maxLevel` leaves it as it is. A revert that falls due does not change it: a light whose revert target is its program returns to the program with its current cap, so that a cap written during a scene stays after the scene's revert. So `runProgram` with `maxLevel` and `revertAfter` changes the cap for good, and a controller that shows a light's program only for a while, such as for a scene without effects, leaves out `maxLevel`. Like every config attribute, it is kept across power loss (Section 7.3.3), and writing it neither resumes control nor cancels a pending revert [OAP-FS Section 7.4.2]. A light of revision 0.3 or earlier has no `programMaxLevel`: only `runProgram` sets its cap, which lasts until the next `runProgram`, by default 100 %, and a revert back to its program restores the cap it had then [OAP-LP Section 16.4].
8. `runProgram` with `effects false` shows a light's program without its effects until the light leaves its program, or receives another `runProgram`. A revert that returns a light to its program MUST restore the program's effects, so that a scene that holds the light still, sent as `runProgram` with `effects false` and `revertAfter`, never leaves clouds and storms off for good [OAP-LP Section 16.4].
9. An outlet with `backup-thermostat` follows its program while its backup thermostat switches it on and off to its `setpoint` [OAP-PM Section 11.5]: after `runProgram`, and, with `failSafeAction continue`, as Fail-Safe Behavior Sections 7.3 and 9.1 say. It MUST refuse `runProgram` while its backup thermostat is not `enabled` with a `setpoint`, and MUST leave its program, and switch off, when a write disables its backup thermostat or removes its `setpoint` while it follows its program. A `set`, such as an interlock's stop, takes it off its program, as it takes a light or a pump off its own, so that with `continue` it keeps the state last set, rather than regulating, when its controller fails. Its cutoff holds it off whether or not it follows its program [OAP-PM Section 11.4].

For example, a controller shows a scene with `revertAfter 3600` on a light that runs its own program, and the water then becomes too warm, so the controller writes `programMaxLevel 60`. The scene goes on, and when its revert falls due, the light fades back to its program at no more than 60 %, whether or not the controller is still there.

Likewise, a controller runs a heater outlet on its program, so that its backup thermostat keeps the water at its `setpoint`, even while the controller is gone. A low-water interlock then stops the outlet with `set`, and the controller fails. With `failSafeAction continue`, the outlet keeps the stop, rather than heating a drained sump. An outlet that its controller drives with `set` instead, from the controller's own sensors, is off its program, so with `continue` it keeps the state last set when the controller is lost, and only its cutoff limits a heater left on.

7.14 Operations

An operation, such as a dose, a prime, or a calibration run of a dosing head, a feed, or an advance of a filter roll, runs for a time once a command or a program starts it, and then ends by itself.

1. An operation that ends before it finishes, because a stop, a hold, the endpoint's fail-safe state, a fault, a limit, or a loss of power ends it, MUST NOT resume by itself: neither a revert that falls due nor power-up starts it again [OAP-FS Section 9.1]. A program on the endpoint may later start a new operation, such as a dose of what was left of one that a hold stopped, only as the program's specification allows [OAP-DO Section 10].
2. So the commands that end an operation have no `revertAfter`: the `stop` of a doser, a feeder, or a filter roller, and the `abort` of an analyzer. A temporary stop of a dosing head's program is a hold, `set` with `enabled false` and `revertAfter` (Section 7.13). A binding's stop of a whole device with `revertAfter`, such as the Device Protocol's `device.stop` [OAP-DP Section 11.4], ends operations in the same way, and when its revert falls due, each endpoint returns to its revert target without them.
3. A dosing head MUST refuse a dose, a prime, or a calibration run that would run longer than its `maxContinuousRun`, the running time of a dose being its volume divided by its rate. A controller, or a program on the head, gives a longer dose in parts [OAP-DO Section 14.7]. A head MAY also refuse a run that starts before it has rested, since its previous run ended, for as long as that run lasted.
4. A dosing head that implements `prime` MUST implement `maxDailyPrimeTime`, MUST refuse a prime that would take its priming time in the last 24 hours past it, and MUST stop a prime that reaches it. It MUST keep its priming time of the last 24 hours across power loss, so that a restart does not reset it. It SHOULD ship with a `maxDailyPrimeTime` of no more than 60 s (provisional).
5. A controller MUST send a dosing head's `prime` and `calibrate` only when a person asks for them, never from automation or a program. Primes and calibration runs add product that the doses of a day do not count, so only a person who is there should start them.

For example, a head with a `maxContinuousRun` of 600 s and a `maxDailyPrimeTime` of 60 s refuses a prime of 90 s, and, after primes of 40 s and 20 s, refuses any other prime until 24 hours after the first. A water change that needs 1.5 L from the head at 100 mL/min, 900 s of running, gives it as two doses of 750 mL, with a rest between them.

7.15 Light Output

The `kind` of a light's channel names its color, which says little about how much light the channel gives, or at which wavelengths. A channel MAY also give, as fixed fields of `channels` (Section 9.4), its spectral photon distribution, `spectrum`, and its photon flux at 100 %, `photonFlux`. From them, a controller can estimate how much light a light gives, and at which wavelengths, such as its photosynthetic photon flux, and can show a color temperature program on any light [OAP-LP Section 8.4]. This Specification defines no standard spectra for the kinds of channels: each light gives its own.

1. `photonFlux` is the number of photons, in $\mu\text{mol/s}$, that the channel emits each second at 100 %, at every wavelength, once its output is steady. It counts the photons that leave the light, after its optics.
2. `spectrum` gives the channel's spectral photon flux at each of a list of wavelengths, as a share of its highest, which is 1. Its points MUST be in order of increasing wavelength, each wavelength once, and there MUST be at least two; its highest `relativeFlux` MUST be 1. Between two points, the relative flux changes in a straight line, and below the first point and above the last, it is 0. It SHOULD cover every wavelength at which the channel's spectral photon flux is at least 1 % of its highest; points 5 nm apart describe the spectrum of an LED well.
3. Where a channel gives both `spectrum` and `peakWavelength`, `peakWavelength` SHOULD be a wavelength at which `relativeFlux` is 1.
4. A controller estimates a channel's photon flux between two wavelengths as `photonFlux` times the area under `spectrum` between them, divided by the whole area under it. So it finds the channel's photosynthetic photon flux, from 400 nm to 700 nm, and, since a level is a share of the channel's output [OAP-LP Section 7.4], the light's at any levels, as the sum over its channels of each one's at 100 % times its level divided by 100.

For example, a violet channel with a `photonFlux` of 10 $\mu\text{mol/s}$ and the spectrum `[{"wavelength": 380, "relativeFlux": 0}, {"wavelength": 400, "relativeFlux": 1}, {"wavelength": 420, "relativeFlux": 0}]` emits half its photons below 400 nm. Its photosynthetic photon flux is 5 $\mu\text{mol/s}$ at 100 %, and 3 $\mu\text{mol/s}$ at 60 %.

8 Common Elements

8.1 Device Attributes

Every device has these attributes.

Attribute	Type	Access	Presence	Description
<code>manufacturer</code>	string	fixed	REQUIRED	Maker of the device.
<code>model</code>	string	fixed	REQUIRED	Model name or number.
<code>serialNumber</code>	string	fixed	OPTIONAL	Serial number, unique for the manufacturer and model.
<code>hardwareRevision</code>	string	fixed	OPTIONAL	Hardware revision.
<code>firmwareVersion</code>	string	state	REQUIRED	Version of the firmware currently running.
<code>specRevisions</code>	object	fixed	REQUIRED	The revision of each Open Aquarium specification the device implements, keyed by specification identifier, for example <code>{"device-classes": "0.3"}</code> .
<code>name</code>	string	config	OPTIONAL	Name shown to the owner.
<code>faults</code>	list of fault codes	state	REQUIRED	Device-level faults currently active (Section 8.4).
<code>uptime</code>	number (s)	state	OPTIONAL	Time since the device last started.
<code>supplyVoltage</code>	number (V)	state	OPTIONAL	Voltage of the device's power supply.
<code>internalTemperature</code>	number (degC)	state	OPTIONAL	Temperature inside the device.
<code>mainsPresent</code>	boolean	state	OPTIONAL	Whether mains power is present at the device's plug, for a device whose electronics keep running without it, such as a power module that the module bus also powers (Power Module Section 10.3). <i>Added in 0.5.</i>
<code>powerSource</code>	enum: <code>external</code> , <code>battery</code>	state	OPTIONAL	What the device runs on now: an external supply, such as mains, bus, or DC power, or its own battery. REQUIRED for a device that can run on a battery. <i>Added in 0.2.</i>
<code>batteryLevel</code>	number (%)	state	OPTIONAL	Charge left in the device's battery. REQUIRED for a device that can run on a battery. <i>Added in 0.2.</i>
<code>batteryVoltage</code>	number (V)	state	OPTIONAL	Voltage of the device's battery. <i>Added in 0.2.</i>
<code>clockUncertainty</code>	number (s)	state	OPTIONAL	The device's estimate of the largest difference between its clock and its controller's, for a device that takes the time from its controller, such as a pump that runs its program in step with others (Flow Programs Section 11.4). <i>Added in 0.2.</i>
<code>updateChannel</code>	enum: <code>stable</code> , <code>beta</code>	config	OPTIONAL	The device's firmware update channel (Security and Access Section 15.9). REQUIRED for a device that updates its firmware as that specification defines. <i>Added in 0.2.</i>
<code>securityVersion</code>	integer	state	OPTIONAL	The highest security version the device has confirmed (Security and Access Section 15.9). REQUIRED for a device that updates its firmware as that specification defines. <i>Added in 0.2.</i>
<code>firmwareSigner</code>	enum: <code>manufacturer</code> , <code>owner</code>	state	OPTIONAL	Who signed the firmware the device runs (Security and Access Section 15.8). REQUIRED for a device that updates its firmware as that specification defines. <i>Added in 0.2.</i>
<code>updateFeed</code>	string	fixed	OPTIONAL	The HTTPS URL of the model's update feed (Security and Access Section 15.9). A device with <code>updateFeed</code> MUST have <code>updateKeys</code> . <i>Added in 0.2.</i>

Attribute	Type	Access	Presence	Description
updateKeys	object	fixed	OPTIONAL	The public keys that sign the model's update manifests, as a JWK Set, each key with its thumbprint as <code>kid</code> (Security and Access Section 15.9). <i>Added in 0.2.</i>
supportEnds	date	fixed	OPTIONAL	The end of the model's security support (Security and Access Section 18.2). <i>Added in 0.2.</i>
automaticInstall	boolean	config	OPTIONAL	Whether the device installs firmware updates automatically (Security and Access Section 15.9). <i>Added in 0.2.</i>
lastUpdate	object	state	OPTIONAL	The outcome of the device's latest firmware update (Security and Access Section 15). <i>Added in 0.2.</i>

Fields of `lastUpdate`:

Field	Type	Presence	Description
outcome	enum: installed, reverted, refused	REQUIRED	<code>installed</code> : the new firmware was installed and confirmed; <code>reverted</code> : it failed to confirm, and the device returned to its previous firmware; <code>refused</code> : an image or update manifest failed verification.
version	string	OPTIONAL	The version installed, or the version refused or reverted from.
time	timestamp	REQUIRED	When the update ended.
reason	string	OPTIONAL	Why the update was reverted or refused, as text shown to the owner.

8.2 Endpoint Attributes

Every endpoint has these attributes.

Attribute	Type	Access	Presence	Description
id	string	fixed	REQUIRED	Identifies the endpoint within the device. Stays the same across restarts and firmware updates.
class	string	fixed	REQUIRED	The endpoint's class (Sections 9 to 11).
features	list of strings	fixed	REQUIRED	The features the endpoint implements. May be empty.
name	string	config	OPTIONAL	Name shown to the owner.
faults	list of fault codes	state	REQUIRED	Endpoint faults currently active (Section 8.4).
sounding	boolean	state	OPTIONAL	Whether the device is sounding, or showing, an alarm of its own for the endpoint's faults (Section 7.6). <i>Added in 0.2.</i>

Every endpoint can also implement these commands. Controllers use `identify` during installation, so the owner can match each endpoint to the right port and equipment, and `silence` to silence a device's own alarm wherever the owner silences alarms [OAP-AN Section 14.4]. Neither changes the endpoint's output, so neither resumes control [OAP-FS Section 7.4].

Command	Parameters	Presence	Description
identify	duration (number, s, optional)	OPTIONAL	Makes the endpoint show which physical port or device it is, for example by flashing an indicator next to its connector, for <code>duration</code> (default 10 s). It MUST NOT change the endpoint's output.
silence	duration (number, s, optional)	OPTIONAL	Silences the device's own sounder or alarm indicator for the endpoint's faults that are active now, for <code>duration</code> , or until they clear. A fault that becomes active later sounds again. It MUST NOT change the endpoint's output, and it does not resume control (Fail-Safe Behavior Section 7.4). <i>Added in 0.2.</i>

8.3 Output Attributes

Every endpoint of an output or unit class also has these attributes.

Attribute	Type	Access	Presence	Description
controlSource	enum: controller, local, fail-safe	state	REQUIRED	What the endpoint is following now: commands from its controller, a manual control on the device itself, or its fail-safe behavior.

Attribute	Type	Access	Presence	Description
<code>latched</code>	boolean	state	REQUIRED	Whether a latched stop holds the endpoint (Section 7.4.13). Always <code>false</code> for an endpoint that has no <code>set</code> . <i>Added in 0.5.</i>
<code>failSafeAction</code>	enum: <code>state</code> , <code>continue</code>	config	REQUIRED	What the endpoint does when it loses contact with its controller: enter <code>failSafeState</code> , or keep everything its controller last commanded, including <code>enabled</code> , holds, and a pending revert, and run its local program, or a unit its own logic, only while what was last commanded lets it (Section 8.3).
<code>failSafeState</code>	object	config	REQUIRED	The state entered when <code>failSafeAction</code> is <code>state</code> , given as values of the class's state attributes, for example <code>{"on": false}</code> . It gives a value for every attribute of the class's inactive state, and a device rejects one that does not (Section 8.3). Entering it cancels a pending revert.
<code>failSafeTimeout</code>	number (s)	config	REQUIRED	How long the endpoint waits without contact before its fail-safe behavior starts, from 5 s to 3600 s (Fail-Safe Behavior Section 7.2).
<code>powerOnBehavior</code>	enum: <code>fail-safe</code> , <code>last</code>	config	REQUIRED	What the endpoint does after power-up until its controller takes control: apply its fail-safe behavior, or restore the state it had before it lost power. If a revert was pending when power was lost, <code>last</code> restores the revert's target (Section 7.4). A light fades, and a pump ramps, to a level it restores (Section 8.3).
<code>powerOnDelay</code>	number (s)	config	OPTIONAL	How long the endpoint stays in its inactive state after power-up before applying <code>powerOnBehavior</code> . Default 0.
<code>loadCategory</code>	string	config	OPTIONAL	The equipment connected to or controlled by the endpoint, as a Data Model equipment category, for example <code>heater</code> .
<code>operatingHours</code>	number (h)	state	OPTIONAL	Total time the output has been active.

How endpoints use these attributes when they lose contact with their controller, lose power, or are under manual control is defined by the Fail-Safe Behavior specification [OAP-FS]. Every output and unit endpoint MUST implement it, with the rules of this section.

1. **Inactive state.** Each output and unit class gives its inactive state after its faults (Sections 9 and 10), as Fail-Safe Behavior Table 2 does, and a feature may add to it, as `local-program` adds `enabled false` to a dosing head's. A `failSafeState` MUST give a value for each attribute of its endpoint's inactive state. It MAY also give values of the other state attributes that the class's `set` command sets, such as a pump's `speed`, and MUST NOT give values of any other attributes. An attribute of the inactive state that `set` does not set, such as a doser's `dosing`, MUST have its inactive value. A device MUST reject a write of `failSafeState` that breaks these rules, or that has a value its `set` command would reject, and MUST keep the previous value (Section 7.3.4) [OAP-FS Section 8.1]. For example, a pump rejects `{}` and `{"speed": 70}`, and accepts `{"running": false}` and `{"running": true, "speed": 70}`.
2. **Continuing.** With `failSafeAction continue`, an endpoint that loses contact keeps everything its controller last commanded, including `enabled`, any hold, and any pending revert, which still falls due [OAP-FS Section 7.3]. A dosing head, an analyzer, or a top-off unit runs its program, or its own logic, only while it is enabled, so one that its controller held or disabled stays so. A light, a pump, or an outlet with `backup-thermostat` runs its program only if it was following it, and otherwise keeps the levels, the speed, or the state last set. A filter roller, which has no `enabled`, keeps running its own logic.
3. **Restoring a level.** An endpoint whose `powerOnBehavior` is `last`, and that restores a level or a speed above 0 % after power-up, MUST reach it evenly from 0 %, beginning when its `powerOnDelay` ends [OAP-FS Section 9.1]:
 - a `light` fades up over its program's restore fade [OAP-LP Section 13.4], if it returns to its program, and otherwise over 900 s (provisional);
 - a `pump` ramps up over its program's restore ramp [OAP-FP Section 12.3], if it returns to its program, and otherwise over 30 s (provisional); and
 - a `level-output` whose `loadCategory` is `light` fades up over 900 s, and one whose `loadCategory` is `return-pump` or `circulation-pump` ramps up over 30 s (both provisional). A command from its controller

replaces the fade or the ramp, and a stop ends it at once (Section 7.4.11). So a light that comes back after a power cut at midday does not jump from darkness to full light.

4. **Holds across a power cut.** A dosing head with `local-program` MUST keep `enabled` across power loss, as it keeps `dosedToday`, so that a head that was held when it lost power is still held after power-up, unless it enters a `failSafeState` that enables it [OAP-FS Section 9.1]. A dose that a power cut interrupted does not resume (Section 7.14.1): with `powerOnBehavior last`, a head restores `enabled`, never `dosing`.
5. **Running fail-safe states after power-up.** An endpoint whose `powerOnBehavior` is `fail-safe`, and that enters after power-up a `failSafeState` that runs a pump or lights a light, MUST reach that state evenly from 0 %, beginning when its `powerOnDelay` ends, over the time of item 3 for a level restored without a program [OAP-FS Section 9.1]:
 - a `light` fades up over 900 s, and a `pump` ramps up over 30 s (both provisional); and
 - a `level-output` whose `loadCategory` is `light` fades up over 900 s, and one whose `loadCategory` is `return-pump` or `circulation-pump` ramps up over 30 s (both provisional).

A command from its controller replaces the fade or the ramp, and a stop ends it at once (Section 7.4.11). So a return pump whose fail-safe state runs it at 70 % takes 30 s to reach 70 % after a power cut, rather than starting at once at full force.

6. **Running fail-safe states at other times.** An endpoint that enters a `failSafeState` that runs a pump or lights a light at any other time, such as on loss of contact, at a takeover or a release, or when no controller resumes control after power-up [OAP-FS Sections 7.3, 7.5, and 9.1], MUST reach each level or speed of that state evenly, from its value at that moment, over the time of item 5, whether it rises or falls. It MUST reach its inactive state, or any attribute of its inactive state that the `failSafeState` gives, at once, as it does on a stop (Section 7.4.11). A command from its controller replaces the fade or the ramp, and a stop ends it at once. So a pump that its controller paused for feeding ramps up to its fail-safe speed over 30 s when its controller fails, rather than starting at once at full force, and a light that its controller drove dark at night fades up to its fail-safe levels over 900 s.

8.4 Fault Codes

Code	Meaning
<code>communication-lost</code>	No contact with the controller for longer than the fail-safe timeout.
<code>hardware-failure</code>	An internal hardware fault.
<code>overcurrent</code>	Current above the rated or configured limit.
<code>stuck-on</code>	An output draws current or keeps running while it is commanded off, for example because a relay contact has welded shut.
<code>load-disconnected</code>	An output is on but its load draws no current, for example because equipment is unplugged or a solenoid coil has failed.
<code>overtemperature</code>	The device or a part of it is too hot.
<code>undervoltage</code>	Supply voltage too low.
<code>overvoltage</code>	Supply voltage too high.
<code>wiring-fault</code>	A supervised input detected an open or short circuit in its wiring.
<code>sensor-disconnected</code>	A sensor or probe is not connected.
<code>out-of-range</code>	A reading is outside the range the sensor can measure.
<code>calibration-due</code>	Calibration is due or overdue.
<code>calibration-failed</code>	The last calibration attempt failed.
<code>expired</code>	A consumable or probe has passed its end of life or expiry date.
<code>stall</code>	A motor is blocked or cannot turn.
<code>dry-run</code>	A pump is running without water.
<code>jam</code>	A mechanism is jammed.
<code>low</code>	A container, reservoir, supply, or consumable is low.
<code>empty</code>	A container, reservoir, supply, or consumable is empty.
<code>waste-full</code>	A waste container is full.

Code	Meaning
end-of-roll	A filter roll is used up.
timeout	An operation exceeded its maximum allowed time.
high-level	Water reached a high or overflow sensor.
implausible-result	A measurement result failed a plausibility check.
sample-failure	A sample could not be drawn or analyzed.
clock-invalid	The device's clock is not valid (Fail-Safe Behavior Section 10.5), so it runs no schedule or program that needs the time of day. <i>Added in 0.2.</i>
battery-weak	The device's battery is nearly exhausted, and needs charging or replacing soon. <i>Added in 0.2.</i>
leak	The endpoint has stopped because a leak was detected, and stays stopped until it is reset, such as a top-off unit locked out by its own leak sensor. <i>Added in 0.2.</i>
daily-limit	An operation has reached its daily limit, such as a top-off unit's fills reaching <code>maxDailyFillTime</code> , and waits until the limit allows it again. <i>Added in 0.2.</i>
cutoff	A backup thermostat's cutoff holds the output off (Power Module Section 11.4). <i>Added in 0.5.</i>

9 Output Classes

Output classes act on equipment.

9.1 Switch (switch)

An on/off output, such as a mains outlet, a switched DC port, a relay, or a dry contact.

Features

Feature	Description
power-monitoring	Measures the electrical supply to the load. An endpoint with this feature SHOULD use it to report <code>stuck-on</code> and <code>load-disconnected</code> .
overload-protection	Switches the output off when the load draws too much current.
backup-thermostat	The output has a thermostat of its own, which reads a temperature probe of its device, holds the output off beyond a cutoff, and can switch the output on and off to a setpoint as its program (Section 7.13), as the Power Module specification defines (its Section 11). <i>Added in 0.5.</i>

Attributes

Attribute	Type	Access	Presence	Feature	Description
on	boolean	state	REQUIRED	—	Whether the output is on.
ratedCurrent	number (A)	fixed	OPTIONAL	—	The most current the output may supply (Power Module Section 8.1). <i>Added in 0.5.</i>
switchingDevice	enum: relay, solid-state	fixed	OPTIONAL	—	What switches the output (Power Module Section 8.2). <i>Added in 0.5.</i>
unpoweredState	enum: off, on	fixed	OPTIONAL	—	The output's unpowered state, which its switching device takes by itself whenever the device's electronics cannot drive it (Fail-Safe Behavior Section 8.3; Power Module Section 8.3). <i>Added in 0.5.</i>
endurance	integer	fixed	OPTIONAL	—	The rated number of operations of the switching device at the output's <code>ratedCurrent</code> , with a resistive load (Power Module Section 8.2). <i>Added in 0.5.</i>
switchCount	integer	state	OPTIONAL	—	The operations of the switching device since manufacture, kept across power loss and firmware updates (Power Module Section 9.5). <i>Added in 0.5.</i>
voltage	number (V)	state	REQUIRED	power-monitoring	Supply voltage.

Attribute	Type	Access	Presence	Feature	Description
current	number (A)	state	REQUIRED	power-monitoring	Current drawn by the load.
power	number (W)	state	REQUIRED	power-monitoring	Real power drawn by the load.
powerFactor	number	state	OPTIONAL	power-monitoring	Power factor of the load.
energy	number (kWh)	state	OPTIONAL	power-monitoring	Energy used since the counter was last reset.
currentLimit	number (A)	fixed	REQUIRED	overload-protection	Current at which the output switches off.
tripped	boolean	state	REQUIRED	overload-protection	Whether the output has switched off because of overload.
backupThermostat	object	config	REQUIRED	backup-thermostat	The backup thermostat's settings, which are fail-safe settings (Power Module Sections 11.3 and 13.3). They ship with <code>enabled false</code> .
cutoffContact	boolean	fixed	REQUIRED	backup-thermostat	Whether the output has a cutoff contact, which opens when the cutoff holds the output off, so that a welded contact cannot defeat it (Power Module Section 11.6).
programRunning	boolean	state	REQUIRED	backup-thermostat	Whether the output follows its program, switching itself on and off by its backup thermostat to its <code>setpoint</code> . <code>false</code> while it follows <code>set</code> , a stop, or its fail-safe state.

Fields of `backupThermostat`:

Field	Type	Presence	Description
enabled	boolean	REQUIRED	Whether the backup thermostat acts.
probe	string	REQUIRED	The identifier of the device's temperature <code>sensor</code> endpoint that it reads.
mode	enum: <code>heat</code> , <code>cool</code>	REQUIRED	<code>heat</code> for a heater on the output; <code>cool</code> for a chiller.
cutoff	number (degC)	REQUIRED	In <code>heat</code> mode, the output is held off while the probe reads at or above it; in <code>cool</code> mode, at or below it (provisional range).
setpoint	number (degC)	OPTIONAL	The temperature the output regulates to as its program, at least <code>hysteresis</code> below <code>cutoff</code> in <code>heat</code> mode, and above it in <code>cool</code> mode.
hysteresis	number (degC)	OPTIONAL	The difference between switching on and off, and how far the reading must return before the cutoff releases. Default 0.5 °C (provisional).

Commands

Command	Parameters	Presence	Feature	Description
set	<code>on</code> (boolean); <code>revertAfter</code> (number, s, optional); <code>latch</code> (boolean, optional, added in 0.5)	REQUIRED	—	Turns the output on or off. With <code>revertAfter</code> , the change is temporary (Section 7.4). With <code>latch true</code> , the stop is latched (Section 7.4.13).
pulse	<code>duration</code> (number, s)	OPTIONAL	—	Turns the output on for a set time, then off.
resetEnergy	—	OPTIONAL	power-monitoring	Resets the energy counter to zero.
resetTrip	—	REQUIRED	overload-protection	Re-enables a tripped output.
runProgram	<code>revertAfter</code> (number, s, optional)	REQUIRED	backup-thermostat	Returns the output to its program: it switches itself on and off by its backup thermostat to its <code>setpoint</code> (Power Module Section 11.5). It cancels any state from <code>set</code> . Without <code>revertAfter</code> , it also cancels any pending revert; with <code>revertAfter</code> , a pending revert keeps its target (Section 7.4). The output MUST refuse it while its backup thermostat is not <code>enabled</code> with a <code>setpoint</code> (Section 7.13.9).

Faults: overcurrent, overtemperature, stuck-on, load-disconnected, hardware-failure, cutoff (added in 0.5).

Inactive state (Fail-Safe Behavior Table 2): {"on": false}.

9.2 Level Output (level-output)

A variable output from 0 to 100 %, such as a 0–10 V signal, a PWM output, or a DALI channel. Used for dimmable lights, variable-speed pumps, and fans that have no smarter interface.

Features

Feature	Description
analog-voltage	The voltage range of a 0–10 V or 1–10 V output can be adjusted.
pwm-frequency	The PWM frequency can be adjusted.

Attributes

Attribute	Type	Access	Presence	Feature	Description
level	number (%)	state	REQUIRED	—	Current output level. A level of 0 % turns the load off where the hardware allows it.
signal	enum: 0-10V, 1-10V, pwm, dali, other	fixed	REQUIRED	—	Type of control signal.
minLevel	number (%)	config	OPTIONAL	—	Lowest level the output will set, other than 0 %.
maxLevel	number (%)	config	OPTIONAL	—	Highest level the output will set.
minVoltage	number (V)	config	REQUIRED	analog-voltage	Output voltage at the lowest level above 0 %.
maxVoltage	number (V)	config	REQUIRED	analog-voltage	Output voltage at 100 %.
frequency	number (Hz)	config	REQUIRED	pwm-frequency	PWM frequency.

Commands

Command	Parameters	Presence	Feature	Description
set	level (number, %); fadeTime (number, s, optional); revertAfter (number, s, optional); latch (boolean, optional, added in 0.5)	REQUIRED	—	Sets the output level, optionally fading to it over fadeTime. With revertAfter, the change is temporary (Section 7.4). With latch true, the stop is latched (Section 7.4.13).

Faults: overcurrent, hardware-failure.

Inactive state (Fail-Safe Behavior Table 2): {"level": 0}.

9.3 Pump (pump)

A pump with its own controller, such as a DC return pump, a wave pump, or a cross-flow pump, or a pump output that drives a pump motor directly.

Features

Feature	Description
reverse	The pump can run in both directions. It comes to a stop, and stays stopped for at least reversePause, before it runs the other way, whatever it is commanded (Flow Programs Section 16.3).
wave	The pump can run a flow program, such as a wave pattern, by itself.
telemetry	The pump reports measurements from its motor.

Attributes

Attribute	Type	Access	Presence	Feature	Description
running	boolean	state	REQUIRED	—	Whether the pump is running.
speed	number (%)	state	REQUIRED	—	Commanded speed, as a percentage of maximum.
minSpeed	number (%)	fixed	OPTIONAL	—	Lowest speed at which the pump can run.
speedLimitLow	number (%)	config	OPTIONAL	—	Lowest speed the owner allows, other than stopped.
speedLimitHigh	number (%)	config	OPTIONAL	—	Highest speed the owner allows.

Attribute	Type	Access	Presence	Feature	Description
ratedFlowRate	number (L/h)	fixed	OPTIONAL	—	Rated maximum flow.
flowCurve	list of objects	fixed	OPTIONAL	—	The pump's flow with no head at each of several speeds, from its maker's measurements, so that a controller can estimate turnover (Flow Programs Section 7.4). <i>Added in 0.2.</i>
direction	enum: forward, reverse	state	REQUIRED	reverse	Current direction.
reversePause	number (s)	fixed	OPTIONAL	reverse	How long the pump stays stopped before it changes direction. <i>Added in 0.2.</i>
program	string	config	REQUIRED	wave	The pump program the pump runs, as JSON text without whitespace between tokens, or a reference to a file of one, as defined by the Flow Programs specification (its Sections 15.1 and 15.2). Empty when the pump follows <code>set</code> commands. After a reference is written, <code>program</code> keeps its previous value until the pump has read and checked the file.
syncGroup	string	config	OPTIONAL	wave	Identifies the group of pumps the pump runs its program with, such as the identifier of a flow function. Pumps of a group keep in step by their clocks and exchange no messages (Flow Programs Section 11); a pump that also keeps in step with others by means of its own MUST still keep to its program's timing.
programRunning	boolean	state	REQUIRED	wave	Whether the pump runs its program, or a pattern of it that <code>runProgram</code> gave it. <code>false</code> while it follows <code>set</code> , a stop, or its fail-safe state. <i>Added in 0.2.</i>
rpm	number (rpm)	state	OPTIONAL	telemetry	Motor speed.
power	number (W)	state	OPTIONAL	telemetry	Power drawn.
temperature	number (degC)	state	OPTIONAL	telemetry	Motor or driver temperature.
flowRate	number (L/h)	state	OPTIONAL	telemetry	Measured or estimated flow, as <code>flowRateSource</code> says.
flowRateSource	enum: measured, estimated	fixed	OPTIONAL	telemetry	Whether <code>flowRate</code> is measured, for example by a flow sensor in the pump, or estimated, for example from the motor's speed and power. <i>Added in 0.2.</i>

Fields of `flowCurve`:

Field	Type	Presence	Description
speed	number (%)	REQUIRED	Commanded speed.
flowRate	number (L/h)	REQUIRED	Flow forward with no head at that speed.
reverseFlowRate	number (L/h)	OPTIONAL	Flow in reverse with no head at that speed, for a pump with the <code>reverse</code> feature.

Commands

Command	Parameters	Presence	Feature	Description
set	<code>running</code> (boolean, optional); <code>speed</code> (number, %, optional); <code>direction</code> (enum: forward, reverse, optional); <code>rampTime</code> (number, s, optional, added in 0.2); <code>revertAfter</code> (number, s, optional); <code>latch</code> (boolean, optional, added in 0.5)	REQUIRED	—	Starts, stops, or changes the speed of the pump, optionally ramping evenly to the new speed over <code>rampTime</code> (Section 7.4). At least one of <code>running</code> , <code>speed</code> , and <code>direction</code> is required. <code>direction</code> requires the <code>reverse</code> feature. With <code>revertAfter</code> , the change is temporary (Section 7.4). With <code>latch</code> <code>true</code> , the stop is latched (Section 7.4.13).
runProgram	<code>pattern</code> (string, optional); <code>transition</code> (number, s,	REQUIRED	wave	Returns the pump to its program, or, with <code>pattern</code> , runs that pattern of its program in place of its schedule, changing over <code>transition</code> seconds, by

Command	Parameters	Presence	Feature	Description
	optional); <code>revertAfter</code> (number, s, optional)			default the program's restore ramp. It cancels any speed from <code>set</code> . Without <code>revertAfter</code> , it also cancels any pending revert; with <code>revertAfter</code> , a pending revert keeps its target (Section 7.4). A pump rejects a <code>pattern</code> its program does not have (Flow Programs Section 15.4). <i>Added in 0.2.</i>

Faults: `stall`, `dry-run`, `overtemperature`, `overcurrent`, `undervoltage`, `overvoltage`, `load-disconnected`, `hardware-failure`.

Inactive state (Fail-Safe Behavior Table 2): `{"running": false}`.

9.4 Light (light)

A light fixture with one or more independently dimmable channels.

Features

Feature	Description
<code>color-temperature</code>	The light can be set by correlated color temperature and intensity instead of by channel.
<code>local-program</code>	The light can run a lighting program by itself.
<code>fixture-temperature</code>	The light measures its own temperature and dims itself when too hot.
<code>power-monitoring</code>	The light reports the power it draws.

Attributes

Attribute	Type	Access	Presence	Feature	Description
<code>on</code>	boolean	state	REQUIRED	—	Whether the light is on.
<code>channels</code>	list of objects	fixed	REQUIRED	—	The fixture's channels.
<code>levels</code>	object	state	REQUIRED	—	Current level of each channel in %, keyed by channel id. A level is a share of the channel's full light output, to which it SHOULD be proportional (Lighting Programs Section 7.4).
<code>colorTemperature</code>	number (K)	state	REQUIRED	<code>color-temperature</code>	Current correlated color temperature.
<code>intensity</code>	number (%)	state	REQUIRED	<code>color-temperature</code>	Current overall intensity.
<code>minColorTemperature</code>	number (K)	fixed	REQUIRED	<code>color-temperature</code>	Lowest color temperature the light can produce.
<code>maxColorTemperature</code>	number (K)	fixed	REQUIRED	<code>color-temperature</code>	Highest color temperature the light can produce.
<code>program</code>	string	config	REQUIRED	<code>local-program</code>	The fixture program the light runs, as JSON text without whitespace between tokens, or a reference to a file of one, as defined by the Lighting Programs specification (its Sections 16.1 and 16.2). Empty when the light follows <code>set</code> commands. After a reference is written, <code>program</code> keeps its previous value until the light has read and checked the file.
<code>programRunning</code>	boolean	state	REQUIRED	<code>local-program</code>	Whether the light shows its program. <code>false</code> while it shows levels from <code>set</code> or <code>setColorTemperature</code> , a stop, or its fail-safe state. <i>Added in 0.2.</i>
<code>programMaxLevel</code>	number (%)	config	REQUIRED	<code>local-program</code>	The cap of the light's program: while the light shows its program, every channel, and the intensity, is at most this level. A controller writes it without changing what the light shows, a revert that falls due does not change it, and <code>runProgram</code>

Attribute	Type	Access	Presence	Feature	Description
					with <code>maxLevel</code> also sets it (Section 7.13.7). Default 100 %. <i>Added in 0.4.</i>
<code>temperature</code>	number (degC)	state	REQUIRED	<code>fixture-temperature</code>	Fixture temperature.
<code>thermalDimming</code>	boolean	state	REQUIRED	<code>fixture-temperature</code>	Whether the light is currently dimmed because it is too hot. While it is <code>true</code> , the endpoint MUST report <code>overtemperature</code> .
<code>thermalScale</code>	number (%)	state	OPTIONAL	<code>fixture-temperature</code>	How far the light has dimmed itself for heat: the share it shows of the levels it should show. 100 % while <code>thermalDimming</code> is <code>false</code> . <i>Added in 0.2.</i>
<code>power</code>	number (W)	state	REQUIRED	<code>power-monitoring</code>	Power drawn.

Fields of channels:

Field	Type	Presence	Description
<code>id</code>	string	REQUIRED	Identifies the channel within the endpoint.
<code>kind</code>	enum: <code>white</code> , <code>cool-white</code> , <code>warm-white</code> , <code>royal-blue</code> , <code>blue</code> , <code>cyan</code> , <code>green</code> , <code>red</code> , <code>deep-red</code> , <code>far-red</code> , <code>violet</code> , <code>uv</code> , <code>moon</code> , <code>full-spectrum</code> , <code>other</code>	REQUIRED	Color of the channel.
<code>peakWavelength</code>	number (nm)	OPTIONAL	Peak wavelength of the channel's LEDs: the wavelength at which their emission is greatest, not their dominant wavelength.
<code>name</code>	string	OPTIONAL	Name shown to the owner.
<code>maxPower</code>	number (W)	OPTIONAL	Power drawn by the channel at 100 %.
<code>minLevel</code>	number (%)	OPTIONAL	Lowest level above 0 % at which the channel's output is stable (Lighting Programs Section 7.4). <i>Added in 0.2.</i>
<code>resolution</code>	number (%)	OPTIONAL	Smallest change of level the channel shows. <i>Added in 0.2.</i>
<code>spectrum</code>	list of objects	OPTIONAL	The channel's spectral photon distribution: its photon flux at each of a list of wavelengths, relative to its highest (Section 7.15). <i>Added in 0.4.</i>
<code>photonFlux</code>	number (umol/s)	OPTIONAL	The channel's photon flux at 100 %: the photons it emits each second, at every wavelength (Section 7.15). <i>Added in 0.4.</i>

Fields of spectrum in channels:

Field	Type	Presence	Description
<code>wavelength</code>	number (nm)	REQUIRED	Wavelength.
<code>relativeFlux</code>	number	REQUIRED	Spectral photon flux at the wavelength, as a share of the channel's highest, which is 1.

Commands

Command	Parameters	Presence	Feature	Description
<code>set</code>	<code>on</code> (boolean, optional); <code>levels</code> (object, optional); <code>fadeTime</code> (number, s, optional); <code>revertAfter</code> (number, s, optional); <code>latch</code> (boolean, optional, <i>added in 0.5</i>)	REQUIRED	—	Turns the light on or off or sets channel levels, optionally fading evenly over <code>fadeTime</code> (Section 7.4). Channels not listed in <code>levels</code> keep their level. With <code>revertAfter</code> , the change is temporary (Section 7.4). With <code>latch</code> <code>true</code> , the stop is latched (Section 7.4.13).
<code>setColorTemperature</code>	<code>colorTemperature</code> (number, K); <code>intensity</code>	REQUIRED	<code>color-temperature</code>	Sets color temperature and intensity, optionally fading over <code>fadeTime</code> : the

Command	Parameters	Presence	Feature	Description
	(number, %); <code>fadeTime</code> (number, s, optional); <code>revertAfter</code> (number, s, optional, added in 0.2)			color temperature evenly in mireds, and the intensity evenly (Section 7.4.10). With <code>revertAfter</code> , the change is temporary (Section 7.4).
<code>runProgram</code>	<code>fadeTime</code> (number, s, optional); <code>maxLevel</code> (number, %, optional); <code>effects</code> (boolean, optional); <code>revertAfter</code> (number, s, optional)	REQUIRED	<code>local-program</code>	Returns the light to its program, fading evenly over <code>fadeTime</code> , by default the program's restore fade. <code>maxLevel</code> sets <code>programMaxLevel</code> , the cap of every channel and the intensity, and without <code>maxLevel</code> the cap stays as it is (Section 7.13.7). <code>effects false</code> stops the program's effects, such as clouds and lightning, until the light leaves its program or receives another <code>runProgram</code> , and a revert that returns the light to its program restores them (Section 7.13.8). It cancels any levels from set. Without <code>revertAfter</code> , it also cancels any pending revert; with <code>revertAfter</code> , a pending revert keeps its target (Section 7.4; Lighting Programs Section 16.4). Added in 0.2.

Faults: `overtemperature`, `hardware-failure`.

Inactive state (Fail-Safe Behavior Table 2): `{"on": false}`.

9.5 Heater (heater)

A heater, or a heater controller, that the controller can enable and, with the `thermostat` feature, give a setpoint.

Features

Feature	Description
<code>thermostat</code>	The heater regulates water temperature to a setpoint with its own sensor.
<code>power-monitoring</code>	The heater reports the power it draws.

Attributes

Attribute	Type	Access	Presence	Feature	Description
<code>enabled</code>	boolean	state	REQUIRED	—	Whether the heater is allowed to heat. Without the <code>thermostat</code> feature, an enabled heater heats continuously.
<code>heating</code>	boolean	state	REQUIRED	—	Whether the heating element is on now.
<code>setpoint</code>	number (degC)	config	REQUIRED	<code>thermostat</code>	Target water temperature.
<code>hysteresis</code>	number (degC)	config	OPTIONAL	<code>thermostat</code>	Difference between switching on and switching off.
<code>temperature</code>	number (degC)	state	REQUIRED	<code>thermostat</code>	Water temperature at the heater's sensor.
<code>cutoffTemperature</code>	number (degC)	config	OPTIONAL	<code>thermostat</code>	Temperature above which the heater will not heat, whatever its setpoint or commands.
<code>power</code>	number (W)	state	REQUIRED	<code>power-monitoring</code>	Power drawn.

Commands

Command	Parameters	Presence	Feature	Description
<code>set</code>	<code>enabled</code> (boolean); <code>setpoint</code> (number, degC, optional, added in 0.2); <code>revertAfter</code> (number, s,	REQUIRED	—	Enables or disables the heater. With <code>setpoint</code> , which requires the <code>thermostat</code> feature, it also sets <code>setpoint</code> . With <code>revertAfter</code> , the change is temporary, and <code>setpoint</code> returns with the rest

Command	Parameters	Presence	Feature	Description
	optional); <code>latch</code> (boolean, optional, added in 0.5)			(Section 7.4). With <code>latch</code> true, the stop is latched (Section 7.4.13).

Faults: `overtemperature`, `sensor-disconnected`, `stuck-on`, `load-disconnected`, `hardware-failure`.

Inactive state (Fail-Safe Behavior Table 2): `{"enabled": false}`.

9.6 Chiller (chiller)

A chiller, or a chiller controller, that the controller can enable and, with the `thermostat` feature, give a setpoint.

Features

Feature	Description
<code>thermostat</code>	The chiller regulates water temperature to a setpoint with its own sensor.
<code>power-monitoring</code>	The chiller reports the power it draws.

Attributes

Attribute	Type	Access	Presence	Feature	Description
<code>enabled</code>	boolean	state	REQUIRED	—	Whether the chiller is allowed to cool. Without the <code>thermostat</code> feature, an enabled chiller cools continuously.
<code>cooling</code>	boolean	state	REQUIRED	—	Whether the chiller is cooling now.
<code>setpoint</code>	number (degC)	config	REQUIRED	<code>thermostat</code>	Target water temperature.
<code>hysteresis</code>	number (degC)	config	OPTIONAL	<code>thermostat</code>	Difference between switching on and switching off.
<code>temperature</code>	number (degC)	state	REQUIRED	<code>thermostat</code>	Water temperature at the chiller's sensor.
<code>power</code>	number (W)	state	REQUIRED	<code>power-monitoring</code>	Power drawn.

Commands

Command	Parameters	Presence	Feature	Description
<code>set</code>	<code>enabled</code> (boolean); <code>setpoint</code> (number, degC, optional, added in 0.2); <code>revertAfter</code> (number, s, optional); <code>latch</code> (boolean, optional, added in 0.5)	REQUIRED	—	Enables or disables the chiller. With <code>setpoint</code> , which requires the <code>thermostat</code> feature, it also sets <code>setpoint</code> . With <code>revertAfter</code> , the change is temporary, and <code>setpoint</code> returns with the rest (Section 7.4). With <code>latch</code> true, the stop is latched (Section 7.4.13).

Faults: `overtemperature`, `sensor-disconnected`, `hardware-failure`.

Inactive state (Fail-Safe Behavior Table 2): `{"enabled": false}`.

9.7 Doser (doser)

One dosing head, such as one head of a peristaltic or stepper-motor dosing pump. A multi-head doser has one endpoint per head.

Features

Feature	Description
<code>reverse</code>	The head can run in reverse, for example to remove water.
<code>container</code>	The head tracks the volume left in its supply container.
<code>local-program</code>	The head can run a dosing program by itself.
<code>calibration-points</code>	The head is calibrated at several rates, and keeps a calibration point for each (Dosing Section 8.3). To give a dose at a rate, it runs at the speed whose flow, on straight lines between its points, is that rate, and it refuses a dose at a rate below its lowest point's flow or above its highest. It keeps at least 4 points (provisional). Added in 0.2.

Attributes

Attribute	Type	Access	Presence	Feature	Description
dosing	boolean	state	REQUIRED	—	Whether the head is running.
maxFlowRate	number (mL/min)	fixed	REQUIRED	—	Highest flow rate of the head.
minFlowRate	number (mL/min)	fixed	OPTIONAL	—	Lowest flow rate of the head.
calibratedFlowRate	number (mL/min)	config	REQUIRED	—	Measured flow rate at the default dosing speed, set by calibration. With <code>calibration-points</code> , a calibration at <code>doseRate</code> also sets it.
doseRate	number (mL/min)	config	OPTIONAL	—	Flow rate used for doses when a command does not give one.
maxDoseVolume	number (mL)	config	REQUIRED	—	Largest single dose the head will deliver. <i>REQUIRED from 0.2; OPTIONAL before.</i>
maxDailyVolume	number (mL)	config	REQUIRED	—	Most the head will deliver in one calendar day. <i>REQUIRED from 0.2; OPTIONAL before.</i>
maxContinuousRun	number (s)	fixed	REQUIRED	—	Longest the head can run without a rest. The head refuses a dose, prime, or calibration run that would run longer, and may refuse one that starts before it has rested for as long as its previous run lasted (Section 7.14). <i>REQUIRED from 0.5; OPTIONAL before.</i>
maxDailyPrimeTime	number (s)	config	OPTIONAL	—	The most time the head may spend priming in any 24 hours. The head refuses a prime that would exceed it, and stops a prime that reaches it (Section 7.14). <i>REQUIRED for a head that implements prime. Added in 0.3.</i>
dosedToday	number (mL)	state	REQUIRED	—	Volume delivered since local midnight, not counting primes and calibration runs. The head keeps it across power loss, and does not reset it while it does not know when local midnight is (Dosing Section 14.2). <i>REQUIRED from 0.2; OPTIONAL before.</i>
containerCapacity	number (mL)	config	REQUIRED	container	Volume of a full container.
containerRemaining	number (mL)	state	REQUIRED	container	Estimated volume left: the volume at the last refill, less every dose, prime, and calibration run since.
containerLowLevel	number (mL)	config	OPTIONAL	container	Volume at which the <code>low</code> fault is raised.
stopWhenEmpty	boolean	config	OPTIONAL	container	Whether the head refuses to dose when the container is estimated to be empty.
program	string	config	REQUIRED	local-program	The head program the head runs, as JSON text without whitespace between tokens, as defined by the Dosing specification (its Section 15.1). Empty when the head follows <code>dose</code> commands.
enabled	boolean	state	REQUIRED	local-program	Whether the head's program may start doses. Changed with <code>set</code> . The head keeps it across power loss, as it keeps <code>dosedToday</code> (Section 8.3). <i>Added in 0.2.</i>
calibrationPoints	list of objects	config	REQUIRED	calibration-points	The head's calibration points, in order of <code>rate</code> . A calibration replaces the point at its rate, or adds one.

Fields of `calibrationPoints`:

Field	Type	Presence	Description
rate	number (mL/min)	REQUIRED	The rate the head ran at, by its own reckoning.
flowRate	number (mL/min)	REQUIRED	The flow it delivered at that rate, as measured.
time	timestamp	OPTIONAL	When the point was calibrated.

Commands

Command	Parameters	Presence	Feature	Description
dose	volume (number, mL); rate (number, mL/min, optional); direction (enum: forward, reverse, optional)	REQUIRED	—	Delivers a volume. <code>direction</code> requires the <code>reverse</code> feature. With <code>calibration-points</code> , <code>rate</code> MUST be within the flows of the head's points.
stop	—	REQUIRED	—	Stops the head immediately. A dose, prime, or calibration run that it stops does not resume (Section 7.14).
prime	duration (number, s)	OPTIONAL	—	Runs the head for <code>duration</code> to fill its tubing. Primed volume is not counted as dosed, but comes out of the container. The head refuses a prime longer than <code>maxContinuousRun</code> , or than <code>maxDailyPrimeTime</code> leaves room for (Section 7.14). A controller sends it only when a person asks.
calibrate	duration (number, s); rate (number, mL/min, optional, added in 0.2)	REQUIRED	—	Runs the head for <code>duration</code> at the default dosing speed, or, with <code>calibration-points</code> , at <code>rate</code> , so the owner can measure the volume delivered. <code>rate</code> requires the <code>calibration-points</code> feature. The head refuses a <code>duration</code> longer than <code>maxContinuousRun</code> (Section 7.14). A controller sends it only when a person asks.
completeCalibration	measuredVolume (number, mL)	REQUIRED	—	Sets the head's calibration from the volume the owner measured during <code>calibrate</code> : <code>calibratedFlowRate</code> , or, with <code>calibration-points</code> , the point at the rate of that run.
refill	volume (number, mL, optional)	REQUIRED	container	Records that the container was refilled, to <code>volume</code> or to <code>containerCapacity</code> .
set	enabled (boolean); skip (boolean, optional); revertAfter (number, s, optional); latch (boolean, optional, added in 0.5)	REQUIRED	local-program	Holds or resumes the head's program. While <code>enabled</code> is <code>false</code> , the program starts no dose: the doses that fall due are delayed, or, with <code>skip</code> <code>true</code> , skipped, and a dose in progress stops unless <code>skip</code> is <code>true</code> (Dosing Section 15.3). A dose that the hold stops does not resume when the hold ends (Section 7.14). With <code>revertAfter</code> , the hold is temporary (Section 7.4). With <code>latch</code> <code>true</code> , the stop is latched (Section 7.4.13). Added in 0.2.

Events

Event	Fields	Feature	Description
doseCompleted	volume (number, mL); time (timestamp)	—	A dose finished. <code>volume</code> is the volume actually delivered.

Faults: stall, low, empty, overtemperature, calibration-due, load-disconnected, hardware-failure.

Inactive state (Fail-Safe Behavior Table 2): `{"dosing": false}`, or, with `local-program`, `{"dosing": false, "enabled": false}`.

9.8 Valve (valve)

A solenoid or motorized valve, such as an RO/DI feed or flush valve or a CO2 valve.

Attributes

Attribute	Type	Access	Presence	Description
<code>open</code>	boolean	state	REQUIRED	Whether the valve is open.
<code>normallyOpen</code>	boolean	fixed	REQUIRED	Whether the valve is open when unpowered.

Commands

Command	Parameters	Presence	Description
<code>set</code>	<code>open</code> (boolean); <code>revertAfter</code> (number, s, optional); <code>latch</code> (boolean, optional, added in 0.5)	REQUIRED	Opens or closes the valve. With <code>revertAfter</code> , the change is temporary (Section 7.4). With <code>latch true</code> , the stop is latched (Section 7.4.13).

Faults: `stuck-on`, `load-disconnected`, `hardware-failure`.

Inactive state (Fail-Safe Behavior Table 2): `{"open": false}`.

9.9 Feeder (feeder)

An automatic feeder.

Features

Feature	Description
<code>food-level</code>	The feeder measures how much food is left.

Attributes

Attribute	Type	Access	Presence	Feature	Description
<code>feeding</code>	boolean	state	REQUIRED	—	Whether the feeder is feeding now.
<code>foodLevel</code>	number (%)	state	REQUIRED	<code>food-level</code>	Food left, as a percentage of a full hopper.

Commands

Command	Parameters	Presence	Feature	Description
<code>feed</code>	<code>portions</code> (integer, optional)	REQUIRED	—	Feeds a number of portions (default 1).
<code>stop</code>	—	REQUIRED	—	Stops feeding at once. A feed that it stops does not resume (Section 7.14).

Events

Event	Fields	Feature	Description
<code>fed</code>	<code>portions</code> (integer); <code>time</code> (timestamp)	—	A feeding finished.

Faults: `jam`, `low`, `empty`, `hardware-failure`.

Inactive state (Fail-Safe Behavior Table 2): `{"feeding": false}`.

10 Unit Classes

Unit classes are equipment that runs its own control logic and safety checks, such as a top-off unit or an analyzer. A controller enables, configures, and monitors them rather than switching them directly. They have the output attributes of Section 8.3. For a unit class, a `failSafeAction` of `continue` keeps the unit's last commanded `enabled`, and the unit runs its own logic only while it is enabled (Section 8.3.2); a filter roller, which has no `enabled`, keeps running its own logic. The fill logic of top-off units is defined by the Water Level and Water Change specification [OAP-WL Section 8], and the testing of analyzers by the Water Testing specification [OAP-WT].

10.1 Auto Top-Off (auto-top-off)

A top-off unit that runs its own fill logic and safety checks. Its level sensors are separate `level-switch` endpoints on the same device. The fill logic and its safety requirements are defined by the Water Level and

Water Change specification. A unit with its own leak sensor stops at once when it detects a leak, and reports `Leak` until `resetFault`.

Features

Feature	Description
<code>reservoir</code>	The unit tracks its water reservoir.
<code>learning</code>	The unit learns how long a normal fill takes and treats much longer fills as faults.
<code>dry-run-detection</code>	The unit detects a pump running dry, for example from its power draw.

Attributes

Attribute	Type	Access	Presence	Feature	Description
<code>state</code>	enum: <code>idle</code> , <code>filling</code> , <code>disabled</code> , <code>fault</code>	state	REQUIRED	—	What the unit is doing.
<code>enabled</code>	boolean	state	REQUIRED	—	Whether the unit is allowed to fill.
<code>maxFillTime</code>	number (s)	config	REQUIRED	—	Longest a single fill may run. The unit stops and raises <code>timeout</code> when it is reached.
<code>minFillInterval</code>	number (s)	config	REQUIRED	—	Shortest time allowed between the end of one fill and the start of the next, however the fill ended. Greater than 0. <i>REQUIRED from 0.2; OPTIONAL before.</i>
<code>startDelay</code>	number (s)	config	OPTIONAL	—	How long the level must stay below the normal sensor before a fill starts (Water Level and Water Change Section 8.4). <i>Added in 0.2.</i>
<code>overrun</code>	number (s)	config	OPTIONAL	—	How long a fill continues after the normal sensor is reached. Default 0. <i>Added in 0.2.</i>
<code>maxDailyFillTime</code>	number (s)	config	OPTIONAL	—	The most fill time allowed in any 24 hours. When the fills of the last 24 hours reach it, the unit stops, reports <code>daily-limit</code> , and starts no fill until the window leaves room for one (Water Level and Water Change Section 8.9). It MUST NOT be less than <code>maxFillTime</code> . <i>Added in 0.2.</i>
<code>fillsToday</code>	integer	state	OPTIONAL	—	Fills since local midnight.
<code>volumeToday</code>	number (mL)	state	OPTIONAL	—	Volume added since local midnight, where the unit can measure or estimate it.
<code>reservoirCapacity</code>	number (mL)	config	REQUIRED	<code>reservoir</code>	Volume of a full reservoir.
<code>reservoirRemaining</code>	number (mL)	state	REQUIRED	<code>reservoir</code>	Estimated volume left.
<code>learnedFillTime</code>	number (s)	state	REQUIRED	<code>learning</code>	Typical duration of a fill.
<code>learningFactor</code>	number	config	OPTIONAL	<code>learning</code>	How many times the learned fill time a fill may run before it stops and reports <code>timeout</code> (Water Level and Water Change Section 8.10). Greater than 1. <i>Added in 0.2.</i>
<code>learningSamples</code>	integer	config	OPTIONAL	<code>learning</code>	How many of the latest normal fills the learned fill time is the median of. At least 2. <i>Added in 0.2.</i>

Commands

Command	Parameters	Presence	Feature	Description
<code>set</code>	<code>enabled</code> (boolean); <code>revertAfter</code> (number, s, optional); <code>latch</code> (boolean, optional, added in 0.5)	REQUIRED	—	Enables or disables filling. With <code>revertAfter</code> , the change is temporary (Section 7.4). With <code>latch true</code> , the stop is latched (Section 7.4.13).
<code>fill</code>	<code>duration</code> (number, s)	OPTIONAL	—	Fills manually for up to <code>duration</code> . The fill still stops at the high sensor and at <code>maxFillTime</code> . A unit MUST refuse it while it is not enabled, so that a manual fill never gets around a hold (Section 7.13).
<code>resetFault</code>	—	REQUIRED	—	Clears a latched fault, such as <code>timeout</code> , <code>high-level</code> , <code>dry-run</code> , or <code>leak</code> , after the owner has checked the system.

Command	Parameters	Presence	Feature	Description
refill	volume (number, mL, optional)	REQUIRED	reservoir	Records that the reservoir was refilled, to volume or to reservoirCapacity.

Events

Event	Fields	Feature	Description
fillCompleted	duration (number, s); volume (number, mL, optional); time (timestamp)	—	A fill finished.

Faults: timeout, high-level, low, empty, dry-run, wiring-fault, hardware-failure, daily-limit (added in 0.2), leak (added in 0.2).

Inactive state (Fail-Safe Behavior Table 2): {"enabled": false}.

10.2 Filter Roller (filter-roller)

An automatic filter that advances a roll of filter material.

Attributes

Attribute	Type	Access	Presence	Description
state	enum: idle, advancing, fault	state	REQUIRED	What the unit is doing.
rollRemaining	number (%)	state	OPTIONAL	Filter material left on the roll.

Commands

Command	Parameters	Presence	Description
advance	length (number, cm, optional)	REQUIRED	Advances the roll, by length or by the unit's normal step.
stop	—	REQUIRED	Stops advancing at once. An advance that it stops does not resume (Section 7.14).
newRoll	length (number, cm, optional)	OPTIONAL	Records that a new roll was fitted.

Events

Event	Fields	Feature	Description
advanced	length (number, cm, optional); time (timestamp)	—	The roll advanced.

Faults: jam, low, end-of-roll, hardware-failure.

Inactive state (Fail-Safe Behavior Table 2): {"state": "idle"}.

10.3 Analyzer (analyzer)

An automatic water tester. Results are reported with Data Model parameter identifiers and units. Scheduling, calibration against lab results, and plausibility checks are defined by the Water Testing specification.

Features

Feature	Description
reagents	The analyzer tracks its reagents.
waste	The analyzer tracks its waste container.
local-program	The analyzer can run a test schedule by itself.
correction	Results can be corrected to match a reference, such as a lab test.

Attributes

Attribute	Type	Access	Presence	Feature	Description
parameters	list of parameter identifiers	fixed	REQUIRED	—	The Data Model parameters the analyzer measures.
methods	object	fixed	OPTIONAL	—	The Data Model method type of each parameter's results, keyed by parameter identifier, for example {"alkalinity":

Attribute	Type	Access	Presence	Feature	Description
					"titration"} (Data Model Section 9.7). <i>Added in 0.2.</i>
state	enum: idle, testing, calibrating, cleaning, disabled, fault	state	REQUIRED	—	What the analyzer is doing.
enabled	boolean	state	REQUIRED	—	Whether the analyzer may test. Changed with <code>set</code> . While it is <code>false</code> , the analyzer starts no test of its own and refuses <code>test</code> , and it reports <code>state</code> <code>disabled</code> once any test in progress has ended. <i>Added in 0.3.</i>
lastResults	list of objects	state	REQUIRED	—	The most recent result for each parameter, from the system's water. Results of tests of a standard are not included.
reagents	list of objects	state	REQUIRED	reagents	Reagents and how much is left.
wasteLevel	number (%)	state	REQUIRED	waste	How full the waste container is.
wasteTests	integer	config	OPTIONAL	waste	How many tests fill an empty waste container. <i>Added in 0.2.</i>
program	string	config	REQUIRED	local-program	The analyzer program the analyzer runs, as JSON text without whitespace between tokens, as defined by the Water Testing specification (its Section 14.1). Empty when the analyzer only tests on command.
corrections	list of objects	config	REQUIRED	correction	Corrections applied to results before they are reported.

Fields of `lastResults`:

Field	Type	Presence	Description
parameter	parameter	REQUIRED	Data Model parameter.
value	number	REQUIRED	Result.
unit	unit	REQUIRED	Data Model unit of <code>value</code> .
qualifier	enum: <, >	OPTIONAL	As in Data Model observations.
raw	number	OPTIONAL	The result before the analyzer's corrections, in <code>unit</code> . Present when a correction changed the result. <i>Added in 0.2.</i>
time	timestamp	REQUIRED	When the sample was taken.

Fields of `reagents`:

Field	Type	Presence	Description
id	string	REQUIRED	Identifies the reagent within the analyzer.
name	string	REQUIRED	Name shown to the owner.
remaining	number (%)	REQUIRED	Amount left.
expires	date	OPTIONAL	Use-by date.
tests	integer	OPTIONAL	How many tests a full container gives. <i>Added in 0.2.</i>
parameters	list of parameter identifiers	OPTIONAL	The parameters whose tests use the reagent. <i>Added in 0.2.</i>

Fields of `corrections`:

Field	Type	Presence	Description
parameter	parameter	REQUIRED	Data Model parameter.
factor	number	OPTIONAL	Multiplier applied first. Default 1.
offset	number	OPTIONAL	Added after the factor, in <code>unit</code> , or, without <code>unit</code> , in the parameter's canonical unit. Default 0.
unit	unit	OPTIONAL	The unit of <code>offset</code> . It MUST be accepted for <code>parameter</code> . <i>Added in 0.2.</i>

Commands

Command	Parameters	Presence	Feature	Description
test	parameters (list of parameter identifiers, optional); standard (boolean, optional, added in 0.2)	REQUIRED	—	Starts a test of the listed parameters, or of all of them. With <code>standard true</code> , the sample is a standard solution of known value, not the system's water, and its results say so.
abort	—	REQUIRED	—	Stops the current test, calibration, or cleaning.
calibrate	—	OPTIONAL	—	Starts a calibration.
clean	—	OPTIONAL	—	Starts a cleaning cycle.
set	enabled (boolean); revertAfter (number, s, optional); latch (boolean, optional, added in 0.5)	REQUIRED	—	Enables or disables testing, as <code>enabled</code> reports. With <code>revertAfter</code> , the change is temporary (Section 7.4). With <code>latch true</code> , the stop is latched (Section 7.4.13).
replaceReagent	id (string); expires (date, optional)	REQUIRED	reagents	Records that a reagent was replaced.
emptyWaste	—	REQUIRED	waste	Records that the waste container was emptied.

Events

Event	Fields	Feature	Description
result	parameter (parameter); value (number); unit (unit); qualifier (enum: <, >, optional); raw (number, optional, added in 0.2); standard (boolean, optional, added in 0.2); time (timestamp)	—	A result is available. Once a controller has screened it (Water Testing Section 10.3), it can be recorded as a Data Model observation, with <code>raw</code> as its correction's <code>raw</code> (Data Model Section 8.4). <code>raw</code> is present when a correction changed the result. A result with <code>standard true</code> is of a standard solution, and is not an observation of the system's water.
testEnded	parameters (list of parameter identifiers); outcome (enum: completed, failed, aborted); fault (string, optional); standard (boolean, optional); started (timestamp, optional); time (timestamp)	—	A test ended, whatever its outcome, after the <code>result</code> events of the results it gave. <code>fault</code> is the fault code that made it fail. Every test ends with this event, so that a controller sees each test and its use of reagents and waste (Water Testing Section 9.1). <i>Added in 0.2.</i>

Faults: low, empty, expired, waste-full, implausible-result, sample-failure, calibration-failed, calibration-due, hardware-failure.

Inactive state (Fail-Safe Behavior Table 2): `{"enabled": false}`.

11 Input Classes

Input classes report what they sense.

11.1 Sensor (sensor)

A sensor that measures one Data Model parameter, such as a pH, ORP, conductivity, temperature, flow, or PAR sensor. A probe that measures two parameters, such as pH and temperature, is two endpoints.

Features

Feature	Description
calibration	The sensor can be calibrated against reference solutions or values. Procedures for each probe type are defined by the Probe Interface specification.
temperature-compensation	Readings are corrected for water temperature.
isolated	The sensor input is electrically isolated from other inputs, so several probes can share the same water without interfering. An endpoint declares it when, and only when, its port meets Probe Interface Section 12.
smart-probe	The probe carries its own identity and calibration, as a Probe Interface probe record (its Section 16), so it keeps its calibration when moved to another port or device.
pulse	The sensor counts pulses, as many flow sensors do.
selectable	The endpoint is one of several endpoints with the same <code>port</code> that report different parameters, such as a port that reads either a pH or an ORP electrode. One of them is active at a time. The

Feature	Description
	others report <code>quality</code> <code>stale</code> , keep their last <code>value</code> , and report no faults of the probe. <i>Added in 0.2.</i>

Attributes

Attribute	Type	Access	Presence	Feature	Description
<code>parameter</code>	parameter	fixed	REQUIRED	—	The Data Model parameter measured.
<code>unit</code>	unit	fixed	REQUIRED	—	The Data Model unit of <code>value</code> . MUST be accepted for <code>parameter</code> .
<code>value</code>	number	state	REQUIRED	—	The latest reading.
<code>qualifier</code>	enum: <code><</code> , <code>></code>	state	OPTIONAL	—	Present when the reading is beyond the measurable range, as in Data Model observations.
<code>quality</code>	enum: <code>good</code> , <code>stale</code> , <code>calibrating</code> , <code>fault</code>	state	REQUIRED	—	Whether <code>value</code> can be used. A sensor MUST NOT report <code>good</code> for a reading that fails its own checks, such as a fault in a probe it depends on or a value outside physical limits. Consumers MUST NOT use a value whose quality is not <code>good</code> for control.
<code>updated</code>	timestamp	state	REQUIRED	—	When <code>value</code> was measured.
<code>measurementInterval</code>	number (s)	state	OPTIONAL	—	The interval at which the sensor normally measures, which some sensors change. The sensor reports <code>quality</code> <code>stale</code> once its latest reading is older than three intervals (Fail-Safe Behavior Section 10.2). <i>Added in 0.3.</i>
<code>rangeMin</code>	number	fixed	OPTIONAL	—	Lowest value the sensor can measure, in <code>unit</code> .
<code>rangeMax</code>	number	fixed	OPTIONAL	—	Highest value the sensor can measure, in <code>unit</code> .
<code>resolution</code>	number	fixed	OPTIONAL	—	Smallest change the sensor reports, in <code>unit</code> .
<code>port</code>	string	fixed	OPTIONAL	—	Identifies the port, or the probe, the endpoint reads. Endpoints that read one probe, such as a conductivity cell's conductivity and salinity, have the same <code>port</code> , and share its calibration (Probe Interface Section 7.1). <i>Added in 0.2.</i>
<code>probeNominal</code>	object	config	OPTIONAL	—	The nominal characteristics of the probe connected, as the <code>nominal</code> member of a Probe Interface probe record (its Section 16.1), set for a probe that does

Attribute	Type	Access	Presence	Feature	Description
					not carry its own record. <i>Added in 0.2.</i>
tdsFactor	number	config	OPTIONAL	—	For total dissolved solids: the factor by which conductivity at 25 °C, in µS/cm, is multiplied to give mg/L (Probe Interface Section 9.6). <i>Added in 0.2.</i>
lastCalibrated	timestamp	state	REQUIRED	calibration	When the sensor was last calibrated successfully.
calibrationDue	timestamp	config	OPTIONAL	calibration	When the next calibration is due.
lastCalibration	object	state	OPTIONAL	calibration	The record of the latest calibration of the endpoint's probe, whatever its outcome, as a Data Model Calibration Record (Data Model Section 8.12), whose reason says why a calibration failed or was cancelled. Details that belong to the probe type, such as a conductivity port's compensation method, are in its extensions (Probe Interface Section 14.9). <i>Added in 0.2.</i>
compensationSource	enum: internal , external , fixed	config	REQUIRED	temperature-compensation	Where the compensation temperature comes from: the sensor's own temperature sensor, a value written by the controller, or a fixed value.
compensationTemperature	number (degC)	config	REQUIRED	temperature-compensation	The temperature used for compensation when the source is external or fixed . With external , a sensor that has not been written for 300 s (provisional) treats it as unavailable, and reports quality stale (Probe Interface Section 7.4).
compensationMethod	enum: pss-78 , linear	config	OPTIONAL	temperature-compensation	For conductivity, and the salinity and total dissolved solids computed from it: how readings are compensated to 25 °C (Probe Interface Section 9.4). <i>Added in 0.2.</i>
compensationCoefficient	number (%/degC)	config	OPTIONAL	temperature-compensation	For the linear method: the coefficient α . Default 2.0. <i>Added in 0.2.</i>
compensationSalinity	number (psu)	config	OPTIONAL	temperature-compensation	For dissolved oxygen in mg/L: the salinity of the water the probe is in (Probe Interface Section 11.1). <i>Added in 0.2.</i>

Attribute	Type	Access	Presence	Feature	Description
probeSerialNumber	string	state	REQUIRED	smart-probe	Serial number of the connected probe.
probeInstalled	date	state	OPTIONAL	smart-probe	When the probe was first used.
probeEndOfLife	date	state	OPTIONAL	smart-probe	Predicted end of the probe's useful life.
pulsesPerLitre	number	config	REQUIRED	pulse	Pulses per litre of flow.
active	boolean	config	REQUIRED	selectable	Whether the port reads this endpoint's parameter now. Writing <code>true</code> makes the port's other endpoints inactive. A write of <code>false</code> is rejected, so that one endpoint of the port is always active.

Commands

Command	Parameters	Presence	Feature	Description
beginCalibration	—	REQUIRED	calibration	Starts a calibration. <code>quality</code> becomes <code>calibrating</code> .
calibratePoint	reference (number); solution (string, optional, added in 0.2)	REQUIRED	calibration	Records a calibration point: the sensor is in a reference whose true value is <code>reference</code> , in <code>unit</code> . <code>solution</code> names the reference solution, as a Probe Interface solution identifier such as <code>technical-7</code> (its Table 20), so that the port can record it, and use its own table for it.
completeCalibration	—	REQUIRED	calibration	Finishes the calibration and applies it. Raises <code>calibration-failed</code> if the points are unusable.
cancelCalibration	—	REQUIRED	calibration	Abandons the calibration and keeps the previous one.
resetCalibration	—	OPTIONAL	calibration	Discards the probe's calibration, and restores the factory calibration of a smart probe, or the probe type's nominal values for another probe (Probe Interface Section 14.2.11).

Faults: `sensor-disconnected`, `out-of-range`, `calibration-due`, `calibration-failed`, `expired`, `hardware-failure`.

11.2 Level Switch (`level-switch`)

A sensor that detects whether water is present at one height, such as an optical level sensor, a float switch, or a conductive pin. The endpoint reports `wet` the same way whatever the sensor type.

Features

Feature	Description
supervised	The input detects broken or shorted wiring. While <code>wiring-fault</code> is active, consumers MUST treat <code>wet</code> as unknown.

Attributes

Attribute	Type	Access	Presence	Feature	Description
wet	boolean	state	REQUIRED	—	Whether water is present at the sensor.
position	enum: <code>low</code> , <code>normal</code> , <code>high</code> , <code>overflow</code> , <code>drain</code> , <code>reservoir-low</code> , <code>reservoir-high</code> , <code>reservoir-empty</code> , <code>other</code>	config	OPTIONAL	—	What the sensor is used to detect. <code>drain</code> is the level a water change drains to, and <code>reservoir-empty</code> a level near the bottom of a reservoir, just above its pump's intake. The values <code>drain</code> and <code>reservoir-empty</code> added in 0.2.
debounce	number (s)	config	OPTIONAL	—	How long a change must last before it is reported.

Attribute	Type	Access	Presence	Feature	Description
invert	boolean	config	OPTIONAL	—	Reverses the sensor's raw signal, for example for a float switch mounted upside down. wet is reported after inversion.

Faults: wiring-fault, sensor-disconnected, hardware-failure.

11.3 Leak Sensor (leak-sensor)

A sensor that detects water where it should not be.

Features

Feature	Description
supervised	The input detects broken or shorted wiring. While wiring-fault is active, consumers MUST treat leak as unknown.
water-type	The sensor tells fresh water, such as RO/DI water, from saltwater.

Attributes

Attribute	Type	Access	Presence	Feature	Description
leak	boolean	state	REQUIRED	—	Whether water is detected.
waterType	enum: fresh, salt, unknown	state	REQUIRED	water-type	Type of water detected.

Faults: wiring-fault, sensor-disconnected, hardware-failure.

11.4 Contact Input (contact-input)

A general-purpose on/off input, such as a dry contact, a door switch, or a button.

Features

Feature	Description
supervised	The input detects broken or shorted wiring. While wiring-fault is active, consumers MUST treat closed as unknown.
momentary	The input is a push button.

Attributes

Attribute	Type	Access	Presence	Feature	Description
closed	boolean	state	REQUIRED	—	Whether the contact is closed.
debounce	number (s)	config	OPTIONAL	—	How long a change must last before it is reported.
invert	boolean	config	OPTIONAL	—	Reverses the raw signal. closed is reported after inversion.

Events

Event	Fields	Feature	Description
pressed	duration (number, s, optional); time (timestamp)	momentary	The button was pressed and released.

Faults: wiring-fault, hardware-failure.

11.5 Analog Input (analog-input)

A general-purpose analog input: a voltage, such as a 0–10 V input, or, with the current-loop feature, a 4–20 mA current loop. It can scale its reading to a Data Model parameter.

Features

Feature	Description
scaling	The input converts its reading to a value of a Data Model parameter: value = voltage × scale + offset, or, with current-loop, value = current × scale + offset.
current-loop	The input measures a 4–20 mA current loop. A current below 3.6 mA or above 21 mA means that the loop is open or shorted, or that its transmitter signals a failure: the input reports wiring-fault, and consumers MUST treat current and value as unknown. A current from 3.6 mA to 3.8 mA, or from 20.5 mA to 21 mA, is beyond the transmitter's range, and the input reports out-of-range (provisional). Added in 0.2.

Attributes

Attribute	Type	Access	Presence	Feature	Description
voltage	number (V)	state	REQUIRED	—	Measured voltage. With <code>current-loop</code> , the voltage across the input's burden resistor.
maxVoltage	number (V)	fixed	REQUIRED	—	Highest voltage the input can measure.
updated	timestamp	state	REQUIRED	—	When <code>voltage</code> was measured.
parameter	parameter	config	REQUIRED	scaling	The Data Model parameter the input represents.
unit	unit	config	REQUIRED	scaling	The Data Model unit of <code>value</code> .
scale	number	config	REQUIRED	scaling	Multiplier from volts, or, with <code>current-loop</code> , from amperes, to <code>unit</code> .
offset	number	config	REQUIRED	scaling	Offset added after scaling, in <code>unit</code> .
value	number	state	REQUIRED	scaling	Scaled value.
current	number (A)	state	REQUIRED	current-loop	Measured loop current.
loopVoltage	number (V)	fixed	OPTIONAL	current-loop	The voltage the input supplies to power a two-wire transmitter. Absent when it supplies none.

Faults: `out-of-range`, `hardware-failure`, `wiring-fault` (added in 0.2).

12 Device Description

Every device **MUST** provide a device description. A binding **MAY** encode it differently, but the encoding **MUST** be convertible to the JSON form [RFC 8259] defined here without loss.

Table 7 — Device description members

Member	Type	Presence	Description
<code>oapDeviceClasses</code>	string	REQUIRED	The revision of this Specification the device implements: "0.5" for this revision, or "0.4", "0.3", "0.2", or "0.1" (Section 7.11).
<code>device</code>	object	REQUIRED	The device's fixed attributes, its <code>firmwareVersion</code> , and its <code>name</code> (Section 8.1), with the same presence as in Section 8.1; <code>optional</code> , REQUIRED from revision 0.2; and optional extensions.
<code>endpoints</code>	array	REQUIRED	One object per endpoint, with <code>id</code> , <code>class</code> , <code>features</code> , <code>optional</code> (REQUIRED from revision 0.2), an optional <code>name</code> , an <code>attributes</code> object, and optional <code>extensions</code> . From revision 0.2, it MAY be empty, for a device with no functions of its own, such as an adapter [OAP-DP Section 16].
<code>extensions</code>	object	OPTIONAL	Extensions, as in the Data Model.

1. An endpoint's `attributes` object **MUST** contain every REQUIRED fixed attribute of its class and of the features it declares. It **MAY** contain values of config attributes, which are the manufacturer's defaults, the values that a factory reset restores, and **MUST NOT** contain state attributes, as both bindings require [OAP-MB Section 10.2.5; OAP-DP Section 11.1.5]. A consumer reads current values of config and state attributes through the binding. A description of revision 0.1 or 0.2 could give state attributes, and a consumer ignores them.
2. `optional` is a list of the names of the optional elements the endpoint implements: OPTIONAL attributes and commands of its class, of the features it declares, and of Sections 8.2 and 8.3. The device's `optional` lists the OPTIONAL attributes of Section 8.1 that it implements. Either **MAY** be empty, and no name appears twice. An endpoint **MAY** also list the extension commands it implements, by their extension names (Section 7.9).
3. An OPTIONAL attribute whose value the description gives is implemented, whether or not `optional` lists it, and an OPTIONAL fixed attribute is implemented only when the description gives its value.
4. `specRevisions` **MUST** include `device-classes`, with the same revision as `oapDeviceClasses`.
5. A device description **MUST** be valid against `schema/device-description.schema.json` [JSON Schema], which accepts every revision, and **MUST NOT** use elements of a later revision than its `oapDeviceClasses`.
6. Consumers **SHOULD** read the device description again after a device's firmware changes.

For example, an endpoint of a two-outlet power strip that measures its load and flashes a light beside its socket for `identify`:

```
{ "id": "outlet_1", "class": "switch", "features": ["power-monitoring"],
  "optional": ["identify", "energy", "resetEnergy", "operatingHours"], "attributes": {} }
```

13 Conformance

13.1 Devices

A conforming device:

1. provides a valid device description, which, from revision 0.2, lists the optional elements it implements, from revision 0.3, gives no state attributes, and, from revision 0.4, gives its channels' spectra as Section 7.15 requires (Section 12);
2. presents its functions as endpoints, and implements, for each endpoint, every REQUIRED element of its class and of the features it declares (Sections 7.1 and 7.2);
3. uses the units defined for each attribute, and keeps and checks its settings (Section 7.3);
4. carries out commands, temporary commands, reverts, fades, ramps, and latched stops as Section 7.4 defines, and runs, caps, and returns to its local programs as Section 7.13 defines;
5. reports the faults of Section 7.6 that apply to it, such as `clock-invalid` and `battery-weak`;
6. enforces its own safety limits, and the limits on operations, primes, and calibration runs (Sections 7.8 and 7.14);
7. never resumes an operation by itself (Section 7.14);
8. implements the Fail-Safe Behavior specification [OAP-FS] on every output and unit endpoint, with the rules of Section 8.3;
9. accepts commands and configuration only as Section 14 allows; and
10. passes the tests in Table 8 that apply to it.

13.2 Controllers

A conforming controller:

1. ignores classes, features, attributes, events, and faults it does not understand (Section 7.9);
2. does not send commands that an endpoint's class and features do not define, and does not rely on optional elements that an endpoint does not list (Section 7.2);
3. does not use a sensor `value` for control unless its `quality` is `good`;
4. treats a supervised input, or a current-loop input, with an active `wiring-fault` as unknown;
5. operates devices of earlier revisions by the rules of their revisions (Section 7.11);
6. when it binds a replacement device, has the owner confirm each binding, and writes the owner's settings to it, and none of the old hardware's calibrations, before it resumes control (Section 7.12);
7. sends a dosing head's `prime` and `calibrate` only when a person asks (Section 7.14); and
8. passes the tests in Table 8 that apply to it.

13.3 Machine-Readable Definitions and Tools

`classes.json` is normative and contains the same definitions as Sections 8 to 11, which are generated from it by `tools/render_spec_tables.py`, as is Table 3. It also gives each output and unit class's inactive state, which the same tool shows after the class's faults. The JSON Schema for device descriptions is normative. The validator, `tools/validate_device_description.py`, is informative. It checks a description against the schema and against the class definitions of its revision: known classes and features, required fixed attributes, attribute types and ranges, Data Model parameters, units, and method types, and the optional elements listed. From revision 0.3, it also checks that endpoints give no state attributes, that a `failSafeState` follows Section 8.3.1, and that a head that implements `prime` implements `maxDailyPrimeTime`. From revision 0.4, it also checks that each `spectrum` of a light's channels follows Section 7.15.2.

13.4 Test Procedures

Each test names the requirements it checks, and whether it is for devices, controllers, or bindings. Every MUST and MUST NOT of this Specification, those of the class tables included, is checked by at least one test. A pass criterion that checks what a SHOULD asks applies to a product that does it. A requirement given in a class table is named by its class and element, as in “`sensor quality`”.

Table 8 — Test procedures

Test	For	Requirements	Method	Pass criteria
Description	Devices	7.1, 7.2.4, 7.9.1, 7.11.1, 7.11.2, 7.15.2, 12; <code>updateFeed</code> ; <code>sensor unit</code> ; <code>analyzer corrections</code>	Read the device’s description through each binding it supports, and validate it. Compare its endpoints with the device’s functions and labels. Restart the device, update its firmware, and read the description again.	Each binding gives the same description in JSON. It passes the validator, which checks the schema, the revision, the optional elements, units accepted for each parameter, from revision 0.3, that it gives no state attributes, and, from revision 0.4, the spectra of a light’s channels; its <code>oapDeviceClasses</code> and <code>specRevisions</code> agree. It has an endpoint for each function, and a <code>sensor</code> endpoint for each parameter measured, each of one class. Endpoint identifiers are unique, and do not change across the restart and the update. Names this Specification does not define are extension names.
Required elements	Devices	7.2.1, 7.2.2, 7.3.1	For each endpoint, read every REQUIRED attribute, and send every REQUIRED command, of its class and of the features it declares. Compare numeric readings with a reference instrument.	Each is implemented, every feature listed is implemented, and each numeric attribute is in its unit.
Optional elements	Devices	7.2.4	Send each OPTIONAL command of each endpoint that its description does not list, and write each OPTIONAL attribute that it does not list.	Each is refused as an element the endpoint does not implement.
Settings	Devices	7.3.3, 7.3.4; <code>auto-top-off</code> ; <code>maxDailyFillTime</code> ; <code>analyzer corrections</code>	Write each config attribute a value outside its permitted range, including a <code>maxDailyFillTime</code> less than <code>maxFillTime</code> and a correction whose <code>unit</code> is not accepted for its parameter; then write valid values, and cut the device’s power for 10 s.	The values outside their ranges are refused, and the previous values kept; the valid values are kept across the power cut.
Whole commands	Devices	7.4.2, 7.4.3	Send a command without a REQUIRED parameter, one with an unknown parameter, one with a value outside its range, and a <code>set</code> with one valid and one invalid value.	Each is rejected, and no part of it is carried out.
Revert target	Devices	7.4.5, 7.4.6	Send an output a temporary state with <code>revertAfter</code> 60. After 20 s, send another temporary state with <code>revertAfter</code> 60. Wait 70 s.	80 s after the first command, the output returns to the state it had before the first command.
Revert cancelled	Devices	7.4.7	Send a temporary state with <code>revertAfter</code> 60, then, after 20 s, the same state without <code>revertAfter</code> . Repeat with <code>runProgram</code> without <code>revertAfter</code> as the second command, on a light or a pump that runs its own program.	The output keeps the second command’s state after the first revert would have fallen due.

Test	For	Requirements	Method	Pass criteria
Reverts and operations	Devices	7.4.7	Hold the program of a dosing head with <code>local-program</code> with <code>set</code> , <code>enabled false</code> , and <code>revertAfter 120</code> , and send it <code>stop</code> 10 s later, then a dose. Repeat with the <code>stop</code> sent just before the hold.	Each time, the head's program resumes when the hold's revert falls due, 120 s after the hold.
A revert kept by <code>runProgram</code>	Devices	7.4.6	Stop a pump that runs its own program with <code>revertAfter 600</code> . After 60 s, send <code>runProgram</code> with a <code>pattern</code> and <code>revertAfter 300</code> .	The pump runs the pattern; 360 s after the first command, it returns to its program, and no revert is left pending.
Reverts and loss of contact	Devices	7.4.7, 8.3.2	With <code>failSafeAction state</code> and <code>failSafeState {"on": false}</code> , send an outlet that is on a <code>set</code> with <code>on false</code> and <code>revertAfter 600</code> , then stop all messages. Repeat with <code>failSafeAction continue</code> .	With <code>state</code> , the outlet enters its fail-safe state and is still off 600 s after the command. With <code>continue</code> , it switches on when the revert falls due.
Revert at power-up	Devices	7.4.8	With <code>powerOnBehavior last</code> , send a temporary state with <code>revertAfter 600</code> , then cut the device's power for 10 s.	After power-up, the output restores the state it had before the temporary state.
Even fades	Devices	7.4.10	Send a light or a level output a change of 50 percentage points with <code>fadeTime 10</code> , and sample its level every 0.5 s. Send a light with <code>color-temperature</code> at 2700 K and 20 % <code>setColorTemperature</code> with 6500 K, 80 %, and <code>fadeTime 60</code> , and sample its <code>colorTemperature</code> and <code>intensity</code> every 0.5 s.	Each sample of a level or of the intensity is within 2 percentage points (provisional) of the straight line from the old value to the new. Each color temperature is within 5 mireds (provisional) of the even change in mireds, such as 3815 K after 30 s.
Stops during fades	Devices	7.4.11	During a fade up of a light over 60 s, send <code>set</code> with <code>on false</code> . Then, from full, send <code>set</code> with every level 0 and <code>fadeTime 60</code> , and after 20 s, <code>set</code> with <code>on false</code> . Repeat with a pump and <code>rampTime</code> .	The first stop takes effect at once. The fade to 0 follows its straight line until the stop, which takes effect at once.
Latched stops	Devices	7.4.13; <code>latched</code>	Give a pump <code>failSafeAction state</code> and a <code>failSafeState</code> that runs it. Send it <code>set</code> with <code>running true</code> and <code>latch true</code> , then <code>set</code> with <code>running false</code> , <code>latch true</code> , and <code>revertAfter 60</code> , then <code>set</code> with <code>running false</code> and <code>latch true</code> , and read <code>latched</code> . Then run the test "Latched stops" of Fail-Safe Behavior Table 4. Repeat with a dosing head with <code>local-program</code> , held with <code>set</code> , <code>enabled false</code> , and <code>latch true</code> , while a dose of its program falls due during each part; and read <code>latched</code> of a feeder.	The first two commands are rejected, and the third is carried out, with <code>latched true</code> . Each endpoint passes the Fail-Safe Behavior test: the pump stays stopped, and the head starts no dose, until a command that changes its output ends the latch, after which <code>latched</code> is <code>false</code> . The feeder reports <code>latched false</code> .
Fading back	Devices	7.4.12, 7.13.4	Send a light that runs its own program <code>set</code> with new levels, <code>fadeTime 20</code> , and <code>revertAfter 60</code> . Then send the same without <code>fadeTime</code> . Repeat with a pump, <code>rampTime</code> , and its restore ramp.	When the first revert falls due, the light fades back to its program over 20 s; when the second does, it fades back over its program's restore fade.
Faults	Devices	7.6.2, 7.6.3	Cause and clear each fault the device can detect, once under its controller and once with no controller; cause a fault that the device keeps until a command clears it.	Each fault is reported when its condition begins and removed when it ends, in both cases; the kept fault stays until the command clears it.

Test	For	Requirements	Method	Pass criteria
No valid time	Devices	7.6.4	Start a device that runs a local program without giving it the time.	It reports <code>clock-invalid</code> , and runs no program that needs the time, until it has the time.
Weak battery	Devices	7.6.5	Discharge a battery-powered device, or simulate its battery.	It reports <code>battery-weak</code> at the latest when <code>batteryLevel</code> reaches 10 %.
Device limits	Devices	7.8	Send a head a dose larger than <code>maxDoseVolume</code> , and doses that would take <code>dosedToday</code> past <code>maxDailyVolume</code> ; start a manual fill of a top-off unit sooner than <code>minFillInterval</code> after the last; and lower a limit while an operation runs. Repeat with every other limit the device documents.	Each command that would exceed a limit is rejected; an operation in progress stops when it reaches a limit.
Held outputs	Devices	7.8.3	While a top-off unit reports <code>daily-limit</code> , send <code>set</code> with <code>enabled true</code> . While a backup thermostat's cutoff holds two outlets off, send the first <code>set</code> with <code>on true</code> , and the second <code>set</code> with <code>on true</code> and then <code>set</code> with <code>on false</code> . End each condition. Repeat with a third outlet, with <code>failSafeAction state</code> and <code>failSafeState {"on": false}</code> , send <code>set</code> with <code>on true</code> during the cutoff, then stop all messages, and end the cutoff after the outlet's timeout.	Each command is accepted. When its condition ends, the top-off unit fills again as its level calls for, the first outlet switches on, and the second stays off. The third outlet enters its fail-safe state, and stays off when the cutoff ends.
Operations do not resume	Devices	7.14.1, 7.14.2, 8.3.4	Stop a dose with <code>stop</code> , a feed, and an advance. Hold a head's program during a dose, with <code>revertAfter 60</code> . Over the Device Protocol, send <code>device.stop</code> with <code>revertAfter 60</code> during a dose. Cut a head's power during a dose, once with <code>powerOnBehavior last</code> and once with <code>fail-safe</code> .	No operation resumes: not when a revert falls due, and not after power-up. After the power cut with <code>last</code> , the head restores <code>enabled</code> , and is not dosing.
Long runs	Devices	7.14.3	Give a head a <code>maxContinuousRun</code> of 60 s. Send a dose that takes 90 s at its rate, a prime of 90 s, and a calibration run of 90 s; then a dose that takes 50 s.	The first three are refused; the last is carried out.
Prime limit	Devices	7.14.4	Give a head that implements <code>prime</code> a <code>maxDailyPrimeTime</code> of 60 s. Send primes of 40 s, then 30 s, then 20 s. Cut the head's power, restore it, and send a prime of 5 s. Then, 24 hours after the first prime, start a prime of 30 s and lower <code>maxDailyPrimeTime</code> to 10 s while it runs.	The prime of 30 s is refused, and the others before the power cut are carried out; the prime after the power cut is refused; the last prime stops when it reaches the new limit.
Manual fills	Devices	<code>auto-top-off fill</code>	With the water at the normal sensor of a top-off unit, disable the unit with <code>set</code> and <code>enabled false</code> , and send <code>fill</code> ; then enable it, wait for <code>minFillInterval</code> , and send <code>fill</code> again.	The first <code>fill</code> is refused, and no fill starts; the second is carried out.
Primes on request	Controllers	7.14.5	Configure every kind of automation, program, mode, and water change the controller offers on a head, and run them for a day. Then, as a person, ask for a prime and a calibration run.	The controller sends <code>prime</code> and <code>calibrate</code> only when the person asks.

Test	For	Requirements	Method	Pass criteria
Programs	Devices	7.13.2, 7.13.3	Write an endpoint a program it cannot run in full. Then write a reference to a program file that the controller serves, where the program's specification allows it.	The first is refused, and the previous program kept. <code>program</code> reports the previous program until the endpoint has read and checked the file.
Returning to a program	Devices	7.13.5; <code>runProgram</code>	Give a light or a pump that runs its program a <code>set</code> without <code>revertAfter</code> , then <code>runProgram</code> .	<code>programRunning</code> becomes <code>false</code> , then <code>true</code> , and the endpoint shows its program again.
Effects after a revert	Devices	7.13.8	On a day whose program shows clouds, send a light that runs its program <code>runProgram</code> with <code>effects false</code> and <code>revertAfter 600</code> , then <code>runProgram</code> with <code>effects false</code> and no <code>revertAfter</code> , and then <code>set</code> with other levels and <code>revertAfter 60</code> .	The light shows no clouds until the first revert falls due, and shows them again after it. After the second <code>runProgram</code> , it shows none, until the revert of the <code>set</code> returns it to its program, which then shows them.
Outlet programs	Devices	7.13.9; <code>switch runProgram</code>	With <code>failSafeAction state</code> , send an outlet with <code>backup-thermostat runProgram</code> while its backup thermostat is disabled; enable it with a <code>setpoint</code> , send <code>runProgram</code> again, and cool and warm its probe past the <code>setpoint</code> . Send <code>set</code> with <code>on false</code> , and cool the probe. Send <code>runProgram</code> , and disable the backup thermostat.	The first <code>runProgram</code> is refused. After the second, <code>programRunning</code> is <code>true</code> , and the outlet switches on and off at the <code>setpoint</code> less and more half the hysteresis. After <code>set</code> , <code>programRunning</code> is <code>false</code> , and the outlet stays off whatever the probe reads. When the backup thermostat is disabled, the outlet switches off, and <code>programRunning</code> becomes <code>false</code> .
Program caps	Devices	7.13.7; <code>programMaxLevel</code>	Run a light's program where it shows more than 60 %. Send <code>set</code> with other levels and <code>revertAfter 120</code> , and then write <code>programMaxLevel 60</code> . After the revert has fallen due, send <code>runProgram</code> without <code>maxLevel</code> , then <code>runProgram</code> with <code>maxLevel 40</code> and <code>revertAfter 60</code> . After that revert has fallen due, cut the light's power for 10 s, and read <code>programMaxLevel</code> .	Writing the cap changes no level, and the light shows the levels of <code>set</code> until the revert falls due; it then shows its program with every channel, and the intensity, at most 60 %. <code>runProgram</code> without <code>maxLevel</code> keeps the cap at 60 %. <code>maxLevel 40</code> sets <code>programMaxLevel</code> to 40, which stays after its revert and the power cut.
Holding a head's program	Devices	7.13	While a dose of a head's program is due, send <code>set</code> with <code>enabled false</code> and <code>revertAfter 300</code> .	No dose starts while it is held, and the head's program resumes when the revert falls due.
Calibration points	Devices	<code>doser dose</code>	Calibrate a head with <code>calibration-points</code> at two rates, then dose at a rate between them, and at a rate above both.	The first dose delivers its volume within the head's documented accuracy; the second is refused.
Identify and silence	Devices	<code>identify</code> , <code>silence</code>	With every output on, send <code>identify</code> to each endpoint. Then disconnect the controller until an endpoint is in its fail-safe behavior and its device sounds, reconnect, and send <code>silence</code> .	<code>identify</code> changes no output. After <code>silence</code> , <code>sounding</code> is <code>false</code> , no output changes, and <code>controlSource</code> stays fail-safe.
Fail-Safe Behavior	Devices	8.3	Run the tests of Fail-Safe Behavior Table 4 for devices on every output and unit endpoint.	Each passes.
Fail-safe states	Devices	8.3.1	For an endpoint of each class, write as <code>failSafeState: {}</code> ; a value without an attribute of its inactive state; one with an attribute that is not a state attribute that <code>set</code> sets, such as a heater's <code>setpoint</code> ; one that gives an attribute that <code>set</code> does not set a value other than its inactive value, such as <code>{"dosing": true}</code> ; one with a value that <code>set</code> would reject; and a valid state.	The first five are rejected, and the previous value kept; the valid state is accepted.

Test	For	Requirements	Method	Pass criteria
Continuing	Devices	8.3.2	With <code>failSafeAction</code> <code>continue</code> , disable a top-off unit and an analyzer, and hold a head's program, each without <code>revertAfter</code> , and stop a light, a pump, and an outlet with <code>backup-thermostat</code> that ran their programs; then stop all messages. Lower the water below the top-off unit's normal sensor, cool the outlet's probe below its <code>setpoint</code> , and let a test and a dose fall due. Separately, let a filter roller with <code>continue</code> lose contact, and make its advance trigger true.	After the timeout, no fill, test, or dose starts, and the light, the pump, and the outlet stay stopped; the filter roller advances.
Restoring a level	Devices	8.3.3	With <code>powerOnBehavior</code> <code>last</code> , run a light at 80 % from <code>set</code> , and a pump at 70 % from <code>set</code> , and cut their power for 10 s. Repeat with a light and a pump that run their own programs, and with a level output whose <code>loadCategory</code> is <code>light</code> .	Each reaches its level from 0 %, evenly within 2 percentage points (provisional), from the end of <code>powerOnDelay</code> : a light over 900 s, a pump over 30 s, and one that returns to its program over its program's restore fade or ramp.
Holds across a power cut	Devices	8.3.4	Hold a head's program, cut its power, and restore it with no controller present, once with <code>powerOnBehavior</code> <code>last</code> and once with <code>fail-safe</code> and <code>failSafeAction</code> <code>continue</code> .	The head reports <code>enabled</code> <code>false</code> , and no dose of its program starts.
Running fail-safe states after power-up	Devices	8.3.5	With <code>powerOnBehavior</code> <code>fail-safe</code> , <code>failSafeAction</code> <code>state</code> , and <code>powerOnDelay</code> 10 s, give a pump the <code>failSafeState</code> <code>{"running": true, "speed": 70}</code> , a light one that lights each channel at 80 %, and a level output whose <code>loadCategory</code> is <code>return-pump</code> <code>{"level": 70}</code> , and cut their power for 10 s with no controller present. Repeat, and send each a stop 10 s after <code>powerOnDelay</code> ends.	Each stays in its inactive state for <code>powerOnDelay</code> , then reaches its fail-safe state from 0 %, evenly within 2 percentage points (provisional): the pump and the level output over 30 s, and the light over 900 s. Each stop takes effect at once.
Running fail-safe states at other times	Devices	8.3.6	With <code>failSafeAction</code> <code>state</code> , give a pump the <code>failSafeState</code> <code>{"running": true, "speed": 70}</code> , and a light one that lights each channel at 80 %. Stop the pump with <code>set</code> and <code>revertAfter</code> 600, set each channel of the light to 20 %, and stop all messages. Repeat with the pump running at 100 %, and again with the <code>failSafeStates</code> <code>{"running": false}</code> and <code>{"on": false}</code> . Then release the device from its controller while the pump is stopped.	After the timeout, the pump ramps from 0 % to 70 % over 30 s, and the light fades from 20 % to 80 % over 900 s, each evenly within 2 percentage points (provisional). From 100 %, the pump ramps down to 70 % over 30 s. The pump and the light reach their inactive states at once. At the release, the pump ramps up as on loss of contact.
Thermal dimming	Devices	<code>light</code> <code>thermalDimming</code>	Heat a light with <code>fixture-temperature</code> until it dims itself, then let it cool.	It reports <code>overtemperature</code> whenever <code>thermalDimming</code> is <code>true</code> .
Sensor quality	Devices	<code>sensor</code> <code>quality</code>	Disconnect a probe that a sensor's reading depends on, then make the reading go outside physical limits.	<code>quality</code> is not <code>good</code> in either case.
Keeping in step	Devices	<code>pump</code> <code>syncGroup</code>	Run two pumps of one group that also keep in step by means of their own, and block those means.	Each keeps to its program's timing within Flow Programs Section 7.9.

Test	For	Requirements	Method	Pass criteria
Current loop	Devices	analog-input current-loop	Open the loop of a current-loop input, then drive 22 mA through it, then 12 mA.	wiring-fault is reported in the first two cases, and cleared in the third.
Selectable endpoints	Devices	sensor selectable	Write active true to the inactive endpoint of a port, then false to it.	The other endpoint of the port reports quality stale; the write of false is refused.
Who may command and configure	Devices	14.2	Send commands, and write config attributes, from a party that is not the device's controller but is authorized to configure it, and from one that is authorized for neither; on a network binding, send a command without authentication.	Commands from anyone but the controller are refused, except identify where the binding allows it; writes are carried out only for a party that the Security and Access specification authorizes; the command without authentication is refused.
Replacing a device	Controllers	7.12.3 to 7.12.5	Replace a device with one of another make, and decline one binding that the controller proposes.	The controller binds only the endpoints the owner confirms, writes the owner's settings, the fail-safe configuration included, before it commands each one, and copies no calibration, counter, or learned value.
Optional and unknown elements	Controllers	7.2.5, 7.9.3	Give a controller a device of revision 0.3 whose description lists no optional elements, with an extension class, feature, attribute, event, and fault.	The controller operates the rest of the device, uses no optional element, and ignores the extensions.
Earlier revisions	Controllers	7.11.3	Have a controller send a pump of revision 0.1 a temporary state that follows another, and resume control of an analyzer of revision 0.2 after a loss of contact, once while it is disabled and once while it is not.	It sends the pump the second temporary state as Automation Section 17.5 describes, and resumes the analyzer with set whose enabled is false while its state is disabled, and true otherwise.
Unusable readings	Controllers	sensor quality; supervised; analog-input current-loop	Make a sensor used for control report quality stale, then fault; make a supervised input and a current-loop input report wiring-fault.	The controller uses none of these values for control, and treats each input as unknown.
Bindings	Bindings	7.5, 7.10, 12	Run the test procedures of the Module Bus [OAP-MB Section 14] and Device Protocol [OAP-DP Section 17] specifications.	Each binding reports state as Section 7.5 requires, and carries names, types, units, and descriptions unchanged.

14 Safety and Security Considerations

1. Fail-safe behavior [OAP-FS] and device-enforced limits (Section 7.8) protect livestock when a controller, network, or binding fails, and when a controller sends a wrong command.
2. A device MUST accept commands only from its controller, except identify, which its binding may let other parties send [OAP-SA Table 6]. A device MUST accept configuration only from the parties that the Security and Access specification authorizes, which include its controller and may include apps of users with the role it requires [OAP-SA Table 6]. Network bindings MUST NOT accept unauthenticated commands.
3. Manual controls on a device take effect immediately and are reported through controlSource, so that a controller never mistakes a manual override for its own command.
4. Names set by the owner may contain personal information and SHOULD be treated as the owner's data.
5. Revert targets (Section 7.4) make a failed controller's last temporary state end in the state before it, rather than in an earlier temporary state that was never meant to last. Entering a fail-safe state cancels a pending revert, so that a temporary state never undoes a fail-safe state.

6. A replacement device receives the owner's settings, and the owner confirms each binding (Section 7.12), so that a new device never runs with a guessed fail-safe configuration, or with the calibration of the hardware it replaces.
7. A device reports its own firmware information (Section 8.1); a controller checks update images and manifests against it as the Security and Access specification requires (its Section 15).
8. Operations that are stopped or cut short never resume by themselves, and primes and calibration runs, which add product outside the limits of doses, are limited by the head and started only by a person (Section 7.14).
9. A cap on a light's program stays when a revert falls due and when the power is cut, so that a cap that protects livestock, such as one that dims a light while the water is too warm, holds when the controller that set it fails (Section 7.13.7).
10. A stop with no known end, such as an interlock's while a leak sensor is wet, can be latched, so that the endpoint keeps it when its controller fails and through power cuts, rather than entering a fail-safe state that runs its equipment (Section 7.4.13). Only stops can be latched, so a latch never starts anything.
11. An interlock's stop takes an outlet with a backup thermostat off its program, so that a heater outlet stopped for low water does not heat a drained sump when its controller fails (Section 7.13.9), and a top-off unit that is not enabled refuses a manual fill, so that a hold of it cannot be got around (Section 10.1).

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 been shown to work together.	Not started	
Every normative reference is publicly available.	Done	
The class definitions, schema, and examples are published, and the examples pass the validator.	Done	
Every requirement has a test procedure (Section 13.4).	Done	
The Open Issues appendix is empty, or each open issue is explicitly deferred.	In progress	Issue 32 waits on the bench tests of the breadboard prototype, and on makers' and owners' experience.
No unresolved Exclusion Notices for this Specification are recorded in <code>Notices.md</code> .	Done	
The Maintainer has recorded consensus to approve.	Not started	

Appendix B — Equipment Mapping (Informative)

How common aquarium equipment is presented as endpoints.

Table 9 — Equipment and endpoints

Equipment	Endpoints
Controller	One endpoint per built-in port: sensor, level-switch, leak-sensor, contact-input, analog-input, switch, and level-output.
Power strip	One switch per outlet, usually with power-monitoring and overload-protection, and with backup-thermostat for an outlet with a thermostat of its own; a sensor for any probe port.

Equipment	Endpoints
Probe module	One sensor per parameter, with calibration, temperature-compensation, and isolated as appropriate, and port naming the port each reads. A combined pH and temperature probe is two endpoints. A port that reads either a pH or an ORP electrode is two endpoints with the same port and the selectable feature.
DC return pump	pump with telemetry, and loadCategory set to return-pump.
Wave or cross-flow pump	pump with wave, telemetry, and, for cross-flow pumps, reverse.
Pump driver for other makers' pumps	One pump per pump output.
0–10 V, PWM, or DALI dimmer	One level-output per channel.
LED light	light, often with local-program and fixture-temperature, and with each channel's spectrum and photonFlux where its maker gives them.
Heater with a built-in controller	heater with thermostat; a controller changes its setpoint for a while with set and revertAfter.
Chiller	chiller with thermostat.
Dosing pump	One doser per head, with calibration-points for a head calibrated at several rates; one level-switch per container sensor, with position reservoir-low or reservoir-empty.
Top-off unit	auto-top-off, one level-switch per level sensor, and a leak-sensor if it has one, on which the unit stops and reports leak.
Automatic water change	doser heads or pumps and valves, with level-switch sensors, such as one with position drain. The sequence is defined by the Water Level and Water Change specification, and runs on a controller. A self-contained water changer presents its parts this way until a class exists for it (Appendix D, issue 7).
RO/DI system	valve endpoints for feed and flush, a sensor for TDS, and level-switch sensors for the reservoir.
Analyzer	analyzer, usually with reagents, waste, and local-program.
Feeder	feeder, with food-level if it measures food.
Filter roller	filter-roller, with a level-switch for the advance trigger.
DC protein skimmer	pump with loadCategory set to protein-skimmer, and a level-switch with position high for the collection cup.
Calcium reactor	pump or switch for the recirculation pump, valve for CO2, and sensor for the reactor pH.
Leak detector	leak-sensor, with supervised and, where possible, water-type.
Flow sensor	sensor for flow-rate, with pulse if it counts pulses.
UV sterilizer	switch with loadCategory uv-sterilizer, with power-monitoring where possible, which shows a failed lamp as load-disconnected. Lamp life is counted from operatingHours since the lamp was replaced (Appendix D, issue 6).
Ozone generator	switch, or level-output for one with a variable output, with loadCategory ozone-generator, driven from an ORP sensor by automation.
Air pump	switch or level-output with loadCategory air-pump.
Fan	switch or level-output with loadCategory fan.
Sensor with a 4–20 mA output	analog-input with current-loop and scaling.
Battery-powered leak sensor	leak-sensor, with the device attributes powerSource and batteryLevel, and the fault battery-weak.

Appendix C — Examples (Informative)

A two-head doser that tracks one container, runs for up to 600 s without a rest, and can be primed for up to 60 s a day, as a device description:

```
{
  "oapDeviceClasses": "0.5",
  "device": {
    "manufacturer": "Example Aquatics", "model": "D2", "firmwareVersion": "1.4.0",
    "specRevisions": { "device-classes": "0.5" }, "optional": []
  }
}
```

```

    },
    "endpoints": [
      { "id": "head_1", "class": "doser", "features": ["container"], "optional": ["prime", "maxDailyPrimeTime"],
        "attributes": { "maxFlowRate": 60, "maxContinuousRun": 600, "maxDailyPrimeTime": 60 } },
      { "id": "head_2", "class": "doser", "features": [], "optional": ["prime", "maxDailyPrimeTime"],
        "attributes": { "maxFlowRate": 60, "maxContinuousRun": 600, "maxDailyPrimeTime": 60 } }
    ]
  }
}

```

The `examples` directory contains complete device descriptions:

- `power-strip.json`: four monitored outlets with their ratings, one with a backup thermostat that reads the strip's temperature probe, and one with its maker's default fail-safe configuration for a return pump.
- `probe-module.json`: isolated pH, ORP, conductivity, salinity, and temperature inputs, a port that reads either a pH or an ORP electrode, and a flow-sensor input.
- `doser.json`: a reversible two-head doser with container tracking, calibration at several rates, head programs, a daily limit on primes, and container level sensors.
- `wave-pump.json`: a reversible cross-flow pump that runs its own wave programs, with its flow at each speed.
- `led-light.json`: a six-channel light with color-temperature control, firmware information, a cap on its own program, and channels that declare their lowest stable levels, and their spectra and photon flux.
- `analyzer.json`: an alkalinity, calcium, and magnesium analyzer, with its methods and its waste container's capacity.
- `auto-top-off.json`: a top-off unit with its level sensors, a leak sensor, and a sounder.
- `io-module.json`: a 4–20 mA input scaled to a dissolved oxygen transmitter's range, a 0–10 V input, a contact input, and two switched outputs.
- `leak-sensor.json`: a battery-powered leak sensor.
- `heater.json`: a heater controller with its own probe.

The examples are of revision 0.5, and descriptions of revisions 0.1 to 0.4 stay valid. The module descriptions of the [breadboard firmware](#) and of the [module bus simulator](#) are of revision 0.3. The breadboard controller implements revisions 0.1 to 0.3, so it refuses descriptions of revisions 0.4 and 0.5 until its firmware implements them (Section 7.11.4). The simulator also keeps one of revision 0.1, with which its tests check how a controller operates devices of earlier revisions (Section 7.11.3).

Appendix D — Open Issues (Informative)

Issues 1 to 23 were open in revision 0.1, and issues 24 to 33 arose in revision 0.2. The Project Lead decided issues 34 to 39 on 15 October 2026, and every other open issue later that day, accepting the recommendation in each. Issues 44 to 46 arose from revision 0.2 of the Lighting Programs and Flow Programs specifications, and the Project Lead decided them, and issue 24, on 16 October 2026, accepting the recommendation in each; revision 0.4 applies them. Issues 47 to 49 arose in revision 0.4, and the Project Lead decided them on 17 October 2026, accepting the recommendation in each. Issues 50 to 53 record the Project Lead's decisions of 17 October 2026, which revision 0.5 applies with issues 33 and 42. Issue 54 records the choices that revision 0.5 made where those decisions left them open, which the Project Lead accepted on 18 October 2026. Each issue now says how a revision resolves it, which specification is to apply it, or what it waits for; only issue 32 is open. Issue numbers do not change, because other specifications refer to them.

1. **Multi-speed calibration.** *Resolved in revision 0.2.* The `doser` feature `calibration-points` keeps a calibration point, the rate a head ran at and the flow measured at it, for each rate the head is calibrated at, and `calibrate` takes a `rate` (Section 9.7), as the Dosing specification proposed (its Section 8.3). `calibratedFlowRate` stays the calibration at `doseRate`, so heads without the feature work as before.
2. **Standalone backup control.** *Resolved in revision 0.3: no rule format is needed here.* Whether a device may run a simple rule of its own as its fail-safe behavior, such as a power strip acting as a thermostat with its own probe, was Fail-Safe Behavior Appendix C, issue 1. That specification's revision 0.2 decides that a device runs such a rule as a local program, defined by the specification of its kind of equipment, with `failSafeAction continue`. The [Power Module](#) specification defines one such rule, a backup thermostat, which revision 0.5 makes the `backup-thermostat` feature of `switch` (issue 33).

3. **Roles and replacement.** *Resolved in revision 0.2.* Endpoints carry no role. The Automation specification already gives outputs and inputs their roles, and binds them to endpoints (its Section 8.5), and endpoints describe their own use with `loadCategory`, `position`, `parameter`, and now `port`. Section 7.12 says how a controller binds a replacement in its predecessor's place, which settings it copies, and which it must not.
4. **State reporting.** *Resolved in revision 0.2.* Reporting rates belong to bindings, which define them (Module Bus Section 11.1; Device Protocol Section 11.5), because they depend on what each connection can carry. Section 7.5 now says what every binding must do: report changes of control, faults, and outputs at once, report every endpoint within an interval it defines, and carry every state attribute in each report. This Specification sets no change thresholds for readings.
5. **Device types.** *Resolved in revision 0.2: none are defined.* A device description already lists a device's endpoints, and Appendix B shows the usual combinations. Formal device types would add a second, more rigid layer that products would have to fit, for no gain to a controller, which works endpoint by endpoint. The specifications of each kind of equipment instead require the combinations they depend on, as the Water Level and Water Change specification requires a top-off unit's normal and high sensors (its Section 8.14). A future compatibility mark could name such combinations as profiles.
6. **Other equipment.** *Resolved in revision 0.2: no new classes.* Air pumps, UV sterilizers, and ozone generators are switched or dimmed loads, which `switch` and `level-output` endpoints with a `loadCategory` describe (Appendix B). A failed UV lamp shows as `load-disconnected` on a `switch` with `power-monitoring`, and a lamp's life is counted from the output's `operatingHours` since the controller recorded the lamp's replacement as a `maintenance` event, with the Data Model Equipment's `endOfLife` for its expected end. Maintenance that falls due by hours of running, rather than by date, is issue 28.
7. **Water changers.** *Resolved in revision 0.2: no class yet.* Three of the four commercial controller ecosystems examined change water through their controllers, with dosing heads, or pumps and valves (Water Level and Water Change Section 9), and no self-contained water changer with an open interface was found. A unit class defined now would be guessed. Until one exists, a self-contained water changer presents its pumps, valves, and sensors as endpoints, which a controller's water change function drives. A water changer class is deferred beyond revision 1.0 (issue 27).
8. **Current-loop inputs.** *Resolved in revision 0.2.* The `analog-input` feature `current-loop` measures a 4–20 mA loop [IEC 60381-1], reports its `current`, scales it with `scaling`, and reports a broken or failed loop as `wiring-fault`, at the signal levels that NAMUR recommends [NE 43] (provisional). A separate class was considered, but many inputs take either a voltage or a current, and a feature lets them keep one class.
9. **Groups.** *Resolved in revision 0.2.* A group of lights or pumps is the outputs of one lighting or flow function, which keep in step by their clocks and exchange no messages (Lighting Programs Section 15; Flow Programs Section 11), so no grouping construct is needed here. `syncGroup` now names a pump's group, and a device can report how closely its clock follows its controller's (`clockUncertainty`).
10. **Cameras.** *Resolved in revision 0.3: no camera class.* The apps of two of the four commercial controller ecosystems examined show camera views. A camera is not equipment that a controller operates, no safety function depends on one, and a camera view is a feature of apps, which can use existing video standards. The Project Lead decided to keep cameras out of the scope of device classes until a function needs one, such as image analysis that feeds automation.
11. **Optional elements.** *Resolved in revision 0.2.* A device description lists, in `optional`, the OPTIONAL attributes and commands that the device and each endpoint implement (Section 12), and the validator checks the lists. A controller does not rely on an optional element that an endpoint does not list (Section 7.2).
12. **Revert targets.** *Resolved in revision 0.2.* A command with `revertAfter` sent while a revert is pending keeps that revert's target (Section 7.4), so a series of temporary states ends in the state before the first of them. An endpoint that loses power with a revert pending, and whose `powerOnBehavior` is `last`, restores the revert's target, which also answers Automation Appendix D, issue 6, and Fail-Safe Behavior Appendix C, issue 8. Raised by the `Automation` specification (Appendix D, issue 22).
13. **Returning to a local program.** *Resolved in revision 0.2.* `runProgram` returns a light or a pump to its program, and `programRunning` says whether it follows it; `set` with `enabled` holds and resumes a dosing head's program, as it does an analyzer's (Section 7.13). Section 7.13 also says how a controller resumes

- control without changing what an endpoint does. Raised by the [Automation](#) specification (Appendix D, issue 5).
14. **Temporary setpoints.** *Resolved in revision 0.2.* The `set` commands of `heater` and `chiller` take a `setpoint`, with the `thermostat` feature, and `revertAfter` returns it with the rest of the state. Raised by the [Automation](#) specification (Appendix D, issue 9).
 15. **Firmware information.** *Resolved in revision 0.2.* The members of Security and Access Table 11 are device attributes: `updateChannel`, `securityVersion`, `firmwareSigner`, `updateFeed`, `updateKeys`, `supportEnds`, and `automaticInstall` (Section 8.1). `lastUpdate` reports the outcome of the latest update, including one that was refused or reverted. It is a state attribute rather than an event, because both bindings carry a device's own state, and neither carries events of a whole device. Raised by the [Security and Access](#) specification (Section 15.9; Appendix D, issue 7).
 16. **Device sounders.** *Resolved in revision 0.2.* An endpoint reports `sounding` while its device sounds or shows an alarm for its faults, and `silence` silences it (Section 8.2). Like `identify`, `silence` does not resume control. Raised by the [Alarms and Notifications](#) specification (Section 14.4; Appendix D, issue 12).
 17. **Top-off units and level sensors.** *Resolved in revision 0.2.* `minFillInterval` is REQUIRED; `startDelay`, `overrun`, `maxDailyFillTime`, `learningFactor`, and `learningSamples` are added; a unit locked out by its own leak sensor reports the fault `leak`, and a unit at its daily limit `daily-limit`; and `level-switch` has the positions `drain`, for a water change's drain sensor, and `reservoir-empty`. A water changer class is issue 7. Raised by the [Water Level and Water Change](#) specification (Appendix D, issue 9).
 18. **Dosing heads.** *Resolved in revision 0.2.* Calibration points (issue 1); `set` with `enabled`, `skip`, and `revertAfter`, and the state attribute `enabled`, for heads with `local-program`; `maxDoseVolume`, `maxDailyVolume`, and `dosedToday` REQUIRED for every head, because a head cannot know whether a program will drive it, with `dosedToday` kept across power loss; primes and calibration runs counted in `containerRemaining`; and `program` defined as the Dosing specification's head program. Raised by the [Dosing](#) specification (Appendix D, issue 9).
 19. **Analyzers.** *Resolved in revision 0.2.* `raw` in `result` and `lastResults`; the event `testEnded`, which ends every test, whatever its outcome; `methods`; reagents' `tests` and `parameters`, and `wasteTests`; `test` with `standard`, for a standard solution, whose results say so and stay out of `lastResults`; a `unit` for the offset of `corrections`; and `program` defined as the Water Testing specification's analyzer program. Results of tests run while the controller was away are kept by the Device Protocol's recent events (its Section 11.6); the module bus does not keep events, which is Module Bus Appendix C, issue 8. Raised by the [Water Testing](#) specification (Appendix D, issue 9).
 20. **Lights.** *Resolved in revision 0.2, and spectra in revision 0.4 (issue 24).* `program` is a fixture program or a reference to one, and a light reports a new one only once it has loaded and checked it; `runProgram` and `programRunning` (issue 13); fades are even (Section 7.4); `setColorTemperature` takes `revertAfter`; levels are a share of light output, and channels declare `minLevel` and `resolution`; a light reports `overtemperature` while `thermalDimming` is `true`, and `thermalScale` says how far it has dimmed; `peakWavelength` is the peak, not the dominant, wavelength; and devices report `clock-invalid`. Spectra are issue 24. Raised by the [Lighting Programs](#) specification (Appendix D, issues 1 and 3).
 21. **Pumps.** *Resolved in revision 0.2.* `program` is a pump program or a reference to one, reported once loaded and checked; `runProgram`, with `pattern`, `transition`, and `revertAfter`, and `programRunning`; `syncGroup` names the pump's group, which keeps in step by its clocks; `set` takes `rampTime`; a reversible pump stops, for at least `reversePause`, before it changes direction; `flowRateSource` says whether `flowRate` is measured or estimated, and `flowCurve` gives the flow at each speed; `clockUncertainty` reports how closely a device's clock follows its controller's; and devices report `clock-invalid`. Raised by the [Flow Programs](#) specification (Appendix D, issue 3).
 22. **Probes.** *Resolved in revision 0.2.* A port that reads either of two parameters, such as pH or ORP, presents an endpoint for each, with the same `port` and the `selectable` feature, of which one is `active` (issue 25 says why); `lastCalibration` reports the latest calibration as a Data Model Calibration Record, with its reason; `probeNominal`, `compensationMethod`, `compensationCoefficient`, `tdsFactor`, and `compensationSalinity` replace the provisional extensions (Table 5); `calibratePoint` takes a `solution`; an external `compensationTemperature` goes stale after 300 s; and `isolated` and `smart-probe` mean what Probe Interface Sections 12 and 16 say. Raised by the [Probe Interface](#) specification (Appendix D, issue 7).

23. **Batteries.** *Resolved in revision 0.2.* The device attributes `powerSource`, `batteryLevel`, and `batteryVoltage`, and the fault `battery-weak`, which Matter's Power Source cluster could show; and `optional` says whether an endpoint implements `identify` (issue 11). Raised by the [Matter Mapping](#) specification (Appendix D, issue 10).

The following issues arose in revision 0.2:

24. **Spectra.** *Resolved in revision 0.4, as the Project Lead decided on 16 October 2026.* The Lighting Programs specification defines no standard spectra: channel kinds stay names of colors, with informative bands (its Appendix D, issue 1). Revision 0.4 lets each channel of a light give, as OPTIONAL fixed fields of `channels`, its spectral photon distribution, `spectrum`, as the relative photon flux at each of a list of wavelengths, and its photon flux at 100 %, `photonFlux`, in `umol/s`, the unit of photon flux that Data Model revision 0.3 adds (Section 7.15). From them, controllers can estimate the photosynthetic photon flux and the daily light integral of a light, and show color temperature programs on any light, which the next revision of the Lighting Programs specification is to do.
25. **Selectable parameters.** *Resolved in revision 0.3: one endpoint per parameter stays.* The Probe Interface specification proposed that a port that reads pH or ORP change one endpoint's `parameter` and `unit` (its Section 7.2.4). Revision 0.2 gives each parameter its own endpoint instead, because a controller of revision 0.1, or one that does not know a new feature, would read millivolts of ORP as pH; an inactive endpoint reports `stale`, which no controller uses for control. The Project Lead kept this design; makers of such ports can raise any difficulty with it during the public review.
26. **Large values on the module bus.** *Resolved here: the Module Bus specification carries it.* A member of an object, or an item of a list, longer than 256 bytes cannot be carried on the module bus (Module Bus Section 11.6), and `lastCalibration` with several points can be. Module Bus Appendix C, issue 11, now covers state values as well as configuration.

Issue 27 is deferred beyond revision 1.0, and is tracked outside this Specification.

28. **Maintenance by running time.** *Resolved by Data Model revision 0.3.* UV lamps and dosing tubing wear with running time, but Data Model Tasks fell due only by date (its Section 8.11). The Project Lead decided on 15 October 2026 that a Task also fall due after a number of hours of running, counted from the `operatingHours` of the outputs that drive its equipment, which serves both, as Dosing Appendix D, issue 22, also asks. Data Model revision 0.3 adds the Task member `everyHours` for it.
29. **Bridged devices.** *Resolved in revision 0.3: they stay in the Device Protocol.* The Device Protocol defines `via`, `failSafe`, and `reachable` for the devices an adapter bridges (its Section 16), and asks whether they belong in the device model (its Appendix C, issue 12). Revision 0.2 lets a description have no endpoints, for an adapter with none of its own. The Project Lead decided to keep the bridged-device members in the Device Protocol, because only that binding has adapters, which answers its Appendix C, issue 12.
30. **Units without codes.** *Resolved in revision 0.3, with Data Model revision 0.3.* `compensationCoefficient` is in % per °C, for which the Data Model had no unit code. The Project Lead decided on 15 October 2026 that the Data Model add one: Data Model revision 0.3 adds `%/degC`, and `compensationCoefficient` names it as its unit.
31. **Reference firmware.** *Resolved in the reference firmware, as the Project Lead decided on 15 October 2026.* The breadboard controller accepted descriptions of revision 0.1 only, so it refused those of other makers' devices of later revisions. It now accepts every revision, checks each by the rules of its revision, as the validator does, and operates each device by them (Section 7.11.3); it relies only on the optional elements a device shows (Section 7.2.5), and sends primes and calibration runs only when a person asks (Section 7.14.5). The breadboard modules present revision 0.3. Revision 0.4 came later: the breadboard controller refuses its descriptions until its firmware implements it (Section 7.11.4).
32. **Numbers to confirm.** *Open: waits on the bench tests of the breadboard prototype, and on makers' and owners' experience.* The battery level at which `battery-weak` is required, the current-loop signal levels, the four calibration points a dosing head keeps, the 300 s after which an external compensation temperature goes stale, and the tolerance of even fades are provisional. So are the numbers that revision 0.3 adds (issue 41), and the 5 mireds within which a fade of color temperature passes its test, which revision 0.4 adds (Table 8). Revision 0.5 adds the 1 s within which a sleepy device reports a change of its faults, which

is the Device Protocol's (its Section 14.3), and the ranges of a backup thermostat's settings, which are the Power Module's (its Appendix E, issue 12).

33. **Power modules.** *Resolved in revision 0.5, as the Project Lead decided.* The [Power Module](#) specification carries what a mains outlet needs as provisional extensions of [switch](#) and of the device (its Tables 5 and 6): an outlet's rated current, switching device, unpowered state, and endurance ([org.openaquariumproject.outlet](#)); a backup thermostat's settings, and the fault [org.openaquariumproject.cutoff](#) while it holds an outlet off; a count of switching operations; and whether mains is present at a module whose electronics stay powered without it. A condition that holds an output off, such as that cutoff or a top-off unit's [daily-limit](#), now accepts commands and makes them wait, rather than refusing them as commands that would exceed a limit (Section 7.8.3). The Project Lead decided that a later revision is to make the extensions attributes, and a [backup-thermostat](#) feature of [switch](#), once revision 0.2 of the Power Module specification has settled them, so that they are not defined twice while it changes. Power Module revision 0.2 settles them as its Tables 2 to 6 and schema define them, adds that a module refuses to disable a backup thermostat while its outlet uses [continue](#), and cites Section 7.8.3 for the commands that a cutoff holds back (its Appendix E, issue 8). Revision 0.5 makes them attributes of [switch](#) and of the device, a fault, and a feature, as that issue lists them: the fixed attributes [ratedCurrent](#), [switchingDevice](#), [unpoweredState](#), and [endurance](#), the state attribute [switchCount](#), the device attribute [mainsPresent](#), the fault [cutoff](#), and the feature [backup-thermostat](#), with its settings, [backupThermostat](#), its [cutoffContact](#), and the program of issue 52. The Power Module's declarations of the module as a whole, its outlets' sockets, inrush ratings, and fuses, which that issue leaves out, stay its own. Raised by the Power Module specification (Appendix E, issue 8).

Decided by the Project Lead

34. **continue for units, and pending reverts.** *Resolved in revision 0.3.* A unit with [failSafeAction](#) [continue](#) keeps its last commanded [enabled](#), and runs its own logic only while enabled (Section 10; Section 8.3.2), and every endpoint with [continue](#) keeps everything last commanded, a pending revert included, as Fail-Safe Behavior Section 7.3 says. Entering [failSafeState](#) cancels a pending revert (Section 7.4.7), as Fail-Safe Behavior Appendix C, issue 13, decides.
35. **Holds across a power cut.** *Resolved in revision 0.3.* A dosing head keeps [enabled](#) across power loss, as it keeps [dosedToday](#), so its hold survives a power cut (Section 8.3.4). A dose that a power cut interrupts never resumes by itself, and nor does any other operation that ends before it finishes (Section 7.14.1).
36. **Fades and ramps after power-up.** *Resolved in revision 0.3.* A [light](#) that restores a level after power-up fades up to it, and a [pump](#) ramps up to it, from 0 %, over its program's restore fade or ramp when it returns to its program, and otherwise over 900 s for a light and 30 s for a pump (Section 8.3.3). The fade and the ramp begin when [powerOnDelay](#) ends, and a [level-output](#) whose [loadCategory](#) is a light or a pump does the same. The lengths are provisional (issue 41).
37. **Primes and calibration runs.** *Resolved in revision 0.3.* A head refuses a prime or a calibration run longer than its [maxContinuousRun](#), and a head that implements [prime](#) has [maxDailyPrimeTime](#), a daily limit on its primes' running time, which it keeps across power loss (Section 7.14). Controllers send [prime](#) and [calibrate](#) only when a person asks. The default limit is provisional (issue 41).
38. **Measurement intervals.** *Resolved in revision 0.3.* [sensor](#) has the OPTIONAL state attribute [measurementInterval](#), the interval in s at which the sensor normally measures, which some sensors change (Section 11.1). Fail-Safe Behavior Section 10.2 names it, so that a controller can read it and set maximum ages from it. Raised by the [Fail-Safe Behavior](#) specification (Appendix C, issue 17).
39. **Analyzers' enabled.** *Resolved in revision 0.3.* [analyzer](#) has the REQUIRED state attribute [enabled](#), which [set](#) changes, so that a controller can resume control of an analyzer with its current [enabled](#) (Section 7.13.5). An analyzer's inactive state is `{"enabled": false}`, with which it reports [state disabled](#) once any test in progress has ended (Section 10.3), as Fail-Safe Behavior Table 2 now says. Raised by the [Fail-Safe Behavior](#) specification (Appendix C, issue 20).

The following issues arose in revision 0.3:

40. **Other specifications to follow.** *Resolved: each of those specifications follows revision 0.3 in its revision 0.2.* Revision 0.3 changed what several specifications relied on, and each has followed:

- the Flow Programs and Lighting Programs specifications, whose Sections 15.4 and 16.4 said that `runProgram` cancels any pending revert, while it now keeps the target of a pending revert when it carries `revertAfter` (Section 7.4.6), and a revert that falls due now fades or ramps (Section 7.4.12);
 - the Lighting Programs specification, whose Section 17.1.2 made `overtTemperature` while a light dims for heat a SHOULD, which `thermalDimming` now requires;
 - the Water Level and Water Change specification, whose Section 7.5.2 described the revert of revision 0.1 as current, and whose water changes by dosing head relied on heads resting by themselves, while a head now refuses a run longer than `maxContinuousRun` (Section 7.14.3);
 - the Alarms and Notifications specification, whose Section 14.4.2 said that devices have no command to silence their sounders, which `silence` and `sounding` now are (issue 16);
 - the Device Protocol and Security and Access specifications, which send `device.stop` and `stop-all` with `revertAfter`, and now say which stops carry it (Section 7.14.2); and
 - the Dosing and Water Testing specifications, which now use `maxDailyPrimeTime`, primes only at a person's request, and an analyzer's `enabled`.
41. **Lengths of restoring fades and ramps, and the prime limit.** *Resolved in revision 0.3: the values stay, provisional, with those of issue 32.* Revision 0.3 proposes, provisionally, that a light that restores levels without a program of its own fades up over 900 s, and a pump that restores a speed ramps up over 30 s (Section 8.3.3), and that a head ship with a `maxDailyPrimeTime` of no more than 60 s (Section 7.14.4). 900 s and 30 s are the default restore fade and restore ramp of the Lighting Programs and Flow Programs specifications (their Sections 13.4 and 12.3), so a light or a pump comes back the same way whether or not it runs a program. At 60 mL/min, 60 s of priming delivers 60 mL, which fills 2 m of tubing with a 2 mm bore about ten times. The Project Lead kept these values until bench tests and owners' experience confirm them.
42. **Heads without `maxContinuousRun`.** *Resolved in revision 0.5, as the Project Lead decided on 17 October 2026: `maxContinuousRun` is REQUIRED. `maxContinuousRun` was OPTIONAL, so a head without it had no limit on the length of a calibration run, and its primes were limited only by `maxDailyPrimeTime`.* Revision 0.3 kept it so, because a person starts every calibration run and watches it, but the Dosing and Water Level and Water Change specifications bound calibration runs and the doses of a water change by it, which needs every head to have one (Section 7.14.3). Revision 0.5 makes it REQUIRED for every dosing head; a head of an earlier revision still need not give it. A head may still refuse a run during its rest (Section 7.14.3), but none has to.
43. **Choices of revision 0.3.** *Resolved in revision 0.3.* The Project Lead accepted the choices that revision 0.3 made where the decisions it applies left them open:
- a `level-output` whose `loadCategory` is a light or a pump fades or ramps up after power-up, as a `light` or a `pump` does (Section 8.3.3);
 - a revert that falls due fades or ramps as the latest command with `revertAfter` did, or, back to a program, with the program's restore fade or ramp (Section 7.4.12);
 - a disabled analyzer refuses `test`, as well as starting no test of its own (Section 10.3);
 - `maxDailyPrimeTime` counts the priming of any 24 hours, as `maxDailyFillTime` counts fills, so that it needs no valid clock (Section 7.14.4); and
 - descriptions of revisions 0.1 and 0.2 that give state attributes stay valid, and consumers ignore those attributes (Section 12.1).

The following issues arose from other specifications after revision 0.3:

44. **A light's cap without `runProgram`.** *Resolved in revision 0.4, as the Project Lead decided on 16 October 2026.* `runProgram` was the only way to change the cap of a light's program, and, without `revertAfter`, it cancels a pending revert (Section 7.4.7), so a cap from dimming on a condition would end a scene that the light shows (Lighting Programs Section 12.3.5). Revision 0.4 gives the `local-program` feature of `light` the config attribute `programMaxLevel`, which holds the cap of the light's program, on every channel and the intensity, which a controller writes without changing what the light shows, and which `runProgram`'s `maxLevel` also sets (Section 7.13.7). A revert that falls due does not change it, so that a cap reaches a light during a scene, and stays after the scene's revert. Its name is issue 47. Raised by the `Lighting Programs` specification (Appendix D, issue 19).

45. **Fades of color temperature.** *Resolved in revision 0.4, as the Project Lead decided on 16 October 2026.* Section 7.4.10 said that a fade changes each level evenly, but not how `setColorTemperature` changes the color temperature. It now says that a fade of a light's color temperature changes it evenly in mireds, one million divided by the color temperature in kelvin, and the intensity evenly, as the Lighting Programs specification computes them (its Sections 8.4 and 14.2), so that a scene of a color temperature program fades in the same way on a light as in its program. The test "Even fades" checks it. Raised by the Lighting Programs specification (Appendix D, issue 20).
46. **Running fail-safe states after power-up.** *Resolved in revision 0.4, with Fail-Safe Behavior revision 0.3 (its Appendix C, issue 22), as the Project Lead decided on 16 October 2026.* Section 8.3.3 made a light fade, and a pump ramp, up to a level it restores after power-up, but an endpoint whose `powerOnBehavior` is `fail-safe` entered its `failSafeState` at once [OAP-FS Section 9.1], so a pump with the recommended configuration of `run` stepped from a stop to its fail-safe speed after every power cut. Section 8.3.5 now says that an endpoint that enters a `failSafeState` that runs a pump, or lights a light, after power-up ramps or fades up to it from 0 %, over the times of Section 8.3.3 for a level it restores without a program, beginning when its `powerOnDelay` ends; a stop still ends the ramp or the fade at once (Section 7.4.11). The test "Running fail-safe states after power-up" checks it. Issue 48 extends this to such a state entered at any other time, such as on loss of contact. Raised by the Flow Programs specification (Appendix D, issue 18).

The following issues arose in revision 0.4:

47. **The name of a program's cap.** *Resolved in revision 0.4: `programMaxLevel`, as the Project Lead decided on 17 October 2026.* The decision of issue 44 left the name of the cap of a light's program open, and `programCap` and `maxProgramLevel` were also considered. `programMaxLevel` begins with `program`, as `program` and `programRunning`, the other attributes of a light's `local-program` feature, do, and it ends with `maxLevel`, the name of the parameter of `runProgram` that also sets it, and of the caps of the Lighting Programs specification (its Section 12), so that one value has one name in each place (Section 7.13.7).
48. **Running fail-safe states on loss of contact.** *Resolved in revision 0.4, with Fail-Safe Behavior revision 0.3 (its Appendix C, issue 24), as the Project Lead decided on 17 October 2026.* Section 8.3.5 ramps or fades an endpoint up to a running fail-safe state after power-up, but one that entered such a state on loss of contact, a takeover, or a release [OAP-FS Sections 7.3 and 7.5] changed to it at once. So a pump that its controller stopped for a while, as for feeding, stepped from a stop to its fail-safe speed when its controller failed, and a light that its controller drove dark at night stepped to its fail-safe levels. Section 8.3.6 now says that an endpoint that enters a `failSafeState` that runs a pump or lights a light at any other time reaches it evenly, from what it shows or runs at that moment, over the times of Section 8.3.5, whether that is up or down, and that it still reaches its inactive state, or any attribute of it, at once, as it does on a stop [OAP-FS Section 8.2]. The ramp delays a pump's full flow by 30 s at most, which is short beside a `failSafeTimeout` of 30 s or more, and lights mostly use `continue` rather than a fail-safe state [OAP-FS Table 3], so little else changes. The test "Running fail-safe states at other times" checks it.
49. **Choices of revision 0.4.** *Resolved in revision 0.4: the Project Lead accepted each choice on 17 October 2026.* Revision 0.4 made these choices where the decisions it applies left them open:
- `runProgram` without `maxLevel` leaves `programMaxLevel` as it is, rather than setting the cap to 100 % as it did before, so that a `runProgram` that ends a scene or a hold never lifts a cap from dimming on a condition (Section 7.13.7);
 - `programMaxLevel` is REQUIRED of every light of revision 0.4 with `local-program`, as `runProgram` is from revision 0.2, so that a controller can rely on it, while a controller caps a light of an earlier revision as before (Section 7.11.3);
 - Section 8.3.5 applies to an endpoint whose `powerOnBehavior` is `fail-safe`, and Section 8.3.6 to one whose `powerOnBehavior` is `last`, and that enters its fail-safe state when no controller resumes control [OAP-FS Section 9.1, step 4], because it has already restored a level;
 - `spectrum` is a list of points, each a wavelength and a relative flux, the highest 1, between which the flux changes in a straight line, rather than a first wavelength and a step, or two lists, so that the points can be closer where the flux changes quickly; and `photonFlux` counts the photons at every wavelength,

- rather than only from 400 nm to 700 nm, so that a controller finds from them the share in any band, and a channel of ultraviolet or far red has a value too (Section 7.15); and
- a fade of color temperature passes its test within 5 mireds, about the smallest change of color that people notice, rather than 2 mireds, 10 mireds, or a share of the fade (provisional, issue 32).

The following issues record the Project Lead's decisions of 17 October 2026, which revision 0.5 applies:

50. **Latched stops.** *Resolved in revision 0.5, with Fail-Safe Behavior revision 0.4, as the Project Lead decided on 17 October 2026.* A stop with no known end, such as an interlock's while a leak sensor is wet, or an emergency mode's, left an output to its fail-safe behavior when its controller failed, so a return pump whose fail-safe state runs it pumped water onto the floor. Revision 0.5 adds `latch` to the `set` of every output and unit class, and `latched` to the output attributes: a controller latches a stop with no known end, and the endpoint keeps it, in place of its fail-safe behavior and its `powerOnBehavior`, through loss of contact and power cuts, until a command from its controller that changes its output ends it (Section 7.4.13; Fail-Safe Behavior Section 8.4). Only a `set` that puts attributes of the inactive state in their inactive values can be latched, so a latch never starts anything; the Automation specification says which stops a controller latches. Raised by the Automation specification (Appendix D, issue 17).
51. **Stops and pending reverts.** *Resolved in revision 0.5, as the Project Lead decided on 17 October 2026.* A dosing head's `stop`, which carries no `revertAfter`, cancelled a pending revert, as every command that changes the output did. A controller that sent a head both a `stop` and a temporary hold of its program, as `stop-all` and emergency modes did, left the head held for good if the `stop` arrived second and the controller then failed. Section 7.4.7 now has a command that starts or ends an operation leave a pending revert as it is, so that other specifications need not prescribe the order of such messages.
52. **Outlets with a backup thermostat.** *Resolved in revision 0.5, with Fail-Safe Behavior revision 0.4, as the Project Lead decided on 17 October 2026.* An outlet whose backup thermostat was enabled with a `setpoint` regulated the water whenever it followed `continue` [OAP-PM Section 11.5], but a `switch` had no hold, so a heater outlet that a low-water interlock had stopped switched on in a drained sump as soon as its controller was lost. The `backup-thermostat` feature gives such an outlet a program, as lights and pumps have: `runProgram` returns the outlet to regulation by its backup thermostat, `programRunning` reports it, and a `set`, such as an interlock's stop, takes it away, so that with `continue` the outlet keeps the stop (Section 7.13.9).
53. **What revision 0.4 left.** *Resolved in revision 0.5, as the Project Lead decided on 17 October 2026.* `maxContinuousRun` is REQUIRED for every dosing head (issue 42). A revert that returns a light to its program restores the program's effects, so that a scene that holds the light still never leaves clouds and storms off for good (Section 7.13.8). `resetCalibration` restores a smart probe's factory calibration, or the probe type's nominal values for another probe, as the Probe Interface specification defines it (its Section 14.2.11). A sleepy device of the Device Protocol reports a change of its faults within 1 s, as it reports a change of `leak`, `wet`, or `closed` (Section 7.5.1).
54. **Choices of revision 0.5.** *Resolved in revision 0.5: the Project Lead accepted each choice on 18 October 2026.* Revision 0.5 made these choices where the decisions it applies left them open:
 - `latch` is a parameter of `set` only, because the `stop` of a doser, a feeder, or a filter roller ends an operation that nothing starts again by itself, a dosing head's program is held with `set`, and a filter roller's own logic has no hold to latch (Section 7.4.13);
 - an outlet with `backup-thermostat` refuses `runProgram` unless its backup thermostat is `enabled` with a `setpoint`, and leaves its program, and switches off, when a write disables its backup thermostat or removes its `setpoint` while it follows its program, so that it never follows a program it cannot run (Section 7.13.9);
 - only the members of the Power Module's extensions that its Appendix E, issue 8, lists become attributes, with `cutoffContact`, which belongs with the backup thermostat, and the Power Module keeps its own declarations of sockets, inrush ratings, fuses, and the module as a whole (issue 33); and
 - a top-off unit that is not enabled refuses a manual `fill`, so that a manual fill never gets around a hold (Section 10.1).