



OPEN AQUARIUM PROJECT SPECIFICATION

Lighting Programs

PRE-DRAFT

Revision	0.3
Status	Pre-Draft. A candidate document submitted for consideration.
Date	2026-10-18
Identifier	lighting-programs
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 `lighting-programs` 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 checks file, channel kinds file, JSON Schema, test vectors, 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 checks file, channel kinds file, schema, test vectors, and examples under the Apache License 2.0. Every computation a program needs is written out in full: the times of its points, its curves, the sun and the moon, its random clouds and storms, and how the light’s settings combine them (Sections 8 to 14). A reference implementation of all of it, with test vectors, is published with the validator, so that makers can check that their products agree, and owners can see why a light does what it does (Section 21.4). Every requirement has a published test procedure (Section 21.5).

2.2 Interoperability

A program is written for kinds of channel, such as white and royal blue, rather than for one product, so the same program runs on lights with different channels, and from different makers (Section 8.6). The same program runs the same way on a controller and on a light that runs it by itself, to within a second and half a percentage point (Sections 7.10 and 16). Programs written by one product can be shared with owners of another, as program documents (Section 18). Lighting is a function type of the Automation specification, `lighting`, so it moves between controllers with the rest of an owner’s configuration. It uses the Device Classes `light`, `level-output`, and `switch` classes, their channel kinds, and the spectra that lights publish, the Automation clock, astronomy, conditions, and modes, Data Model parameters, Fail-Safe Behavior terms, and Alarms and Notifications condition codes, and adds no parallel vocabulary.

2.3 Local Control

Programs run on the controller or on the light itself, with no network service, account, or vendor. The sun and the moon are computed from a location or a table, never fetched (Section 9). A light that runs its own program keeps the day, the night, the moon, and its weather when its controller or the network is down (Section 16.3). Owners can export every program they run, and the programs of their lights, as files that any conforming product reads (Section 18).

2.4 Safety

Light is not as quick a danger as heat or dosing, but corals take days to weeks to adjust to more light, light that changes suddenly startles fish into jumping, flashing light can trigger seizures in people who are sensitive to it, light at night can stop corals spawning, and a light can overheat. This Specification limits each of these:

- a channel that a program does not describe stays off, rather than being guessed at (Section 8.6);
- the owner's caps and acclimation apply to everything a light shows, including scenes and lightning, and a program that is brighter than the one it replaces is brought in gradually (Sections 12 and 18.2);
- scenes end by themselves, even on a light whose controller has failed, every change of state fades, a scene starts from what the light shows, storm or not, and a light that dims returns to its program with a fade after a power cut, a stop, or a new program, rather than jumping from darkness to midday (Sections 13 and 17.4);
- lightning is off unless the owner turns it on for a light, after a warning about photosensitive epilepsy, which a controller gives again before it turns lightning on for a second light; on each light, it flashes no more than three times in any second, at least 360 ms apart, in bursts at least 5 s apart, never in red, and only during a storm (Section 11.5);
- a light's own thermal limits override every level it is asked for, and a controller never tries to make up for them (Section 17.1); and
- every failure has a defined outcome: a lost controller, a lost clock, a lost input, a lost light, and a failed program file (Section 17).

2.5 Sustainability

Because programs are portable, owners keep their lights when they change controllers, and keep their programs when they change lights. Owners' caps, scales, and dimming on a condition, such as a warm room, let lights run below their maximum, which saves energy and extends LED life (Section 12). Measured light, recorded as PPFD and daily light integral, and the light that lights publish, estimated in photons, let owners give their livestock the light it needs rather than the most a light can give (Section 19).

Revision History

Revision	Date	Description	Authors
0.1	2026-10-14	Initial Pre-Draft.	Trent Scholl
0.2	2026-10-16	Scenes fade in from what a light shows, its clouds and storms included, and fade out to what the program then shows, with caps applied before the scene; on a light that runs its own program, a scene ends with a fade even when its controller has failed, because a revert of Device Classes revision 0.3 fades; a cap from <code>dim</code> that changes during a scene no longer ends it; only one light of a lighting function flashes for lightning; lights the controller drives keep <code>last</code> , and every light fades up to the levels it restores after power-up; clouds and storms should fade over at least 10 s; <code>runProgram</code> and <code>programRunning</code> of Device Classes revision 0.2, which keep a pending revert's target, with the provisional names kept for lights of revision 0.1; <code>overtemperature</code> required while a light dims for heat; a light's clock valid as Fail-Safe Behavior defines it, with <code>clock-invalid</code> while it is not; <code>program-not-loaded</code> raised for each light, as a condition whose delay is in <code>conditions.json</code> ; faults that a light reports not raised again for its function; a <code>method</code> that this revision does not define unsupported rather than invalid; the Data Model's <code>daily-light-integral</code> ; the limit of one command a second stated as this Specification's own; the claim about program files swapped on the way limited to the Device Protocol; examples of Device Classes revision 0.2, with lightning on one light and a two-hour maintenance mode; requirement keywords for every safety check; a	Trent Scholl

Revision	Date	Description	Authors
		test procedure, mapped to its requirements, for every requirement; test vectors for scenes during a storm; references cited by tag; one sentence per line; the decided issues resolved, with spectra left to makers, which are to publish them, and one conformance level for lights, and every other issue given a status.	
0.3	2026-10-18	The Project Lead's decisions of 17 October 2026 applied: a controller warns before it turns lightning on for a second light of any of its lighting functions, a validator warns about a configuration with more than one, and no text reads as a promise that a group never flashes more than three times a second; the function type <code>lighting</code> , with its saved state as <code>state.functions["<function_id>"]</code> , in place of the provisional names, which controllers still read; the configuration's <code>noonAt</code> and <code>shiftDays</code> , which Automation revision 0.3 adds, read and written into fixture programs, and <code>noonAt</code> allowed with a seasonal table that has a <code>location</code> ; the schema's \$id https://openaquariumproject.org/schemas/lighting-programs/0.3 . The decisions of 16 October 2026 that Device Classes revision 0.4 applies, used: the cap from <code>dim</code> as the cap of a light's program, <code>programMaxLevel</code> , which stays after a scene's revert; fades of color temperature evenly in mireds; mixes that a controller proposes from the <code>spectrum</code> and <code>photonFlux</code> of a light's channels, as a light member's new <code>mixes</code> , so that a color temperature program without mixes runs on a light without the <code>color-temperature</code> feature; a light's photosynthetic photon flux and daily photons, a vessel's daily light integral, and programs compared by their photons, estimated from what lights publish; and fades to fail-safe states that light a light. Also: <code>runProgram</code> when a scene's fade-out begins, so that a scene fades back over its own fade after a cap, and a light of Device Classes revision 0.4 or earlier shows its effects again; every <code>runProgram</code> to a light of Device Classes revision 0.3 or earlier carries the cap in force; no lightning during a restore fade; latched stops; a light driven once <code>program-not-loaded</code> has lasted its delay; the stops that reach a light under local control as Automation revision 0.3 lists them; a restore fade of at least 60 s (provisional), with a warning; switched lights left out of the promise that no light jumps from darkness; power-up fades limited to lights that were following their program; scene triggers required with MUST; the Module Bus's program files and the Device Protocol's time cited; the Flow Programs, Water Testing, and Matter Mapping specifications listed as references; examples of Device Classes revision 0.5, with spectra; "A pass criterion that checks what a SHOULD asks applies to a product that does it"; corrected tests; references to the final revisions of the other specifications; issues 9 to 13, 15, and 16 deferred beyond revision 1.0.	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	3
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	9
5.1	Requirements Language	9
5.2	Terms and Definitions	9
5.3	Notation	10
6	Introduction	10
6.1	Purpose	10
6.2	Design Principles	11
6.3	Overview	11
6.4	Example	13
7	Common Requirements	13
7.1	Where Programs Run	13
7.2	The Lighting Function	14
7.3	Lights and Outputs	14
7.4	Levels	15
7.5	Local Time and the Clock	15
7.6	Holds	15
7.7	Saved State	16
7.8	Faults and Alarms	16
7.9	Visibility	16
7.10	Determinism	17
8	Programs	17
8.1	Program Members	17
8.2	Times	18
8.3	Curves	18
8.4	Color Temperature	19
8.5	Mixes	19
8.6	Channel Kinds	20
8.7	Curves for One Fixture	20
8.8	Safety Checks	21
9	The Sun and the Moon	21
9.1	Sources and Method	21
9.2	The Sun's Position	22
9.3	Sunrise, Solar Noon, and Sunset	22
9.4	The Moon	22
9.5	Seasonal Tables	23
9.6	Accuracy	23
9.7	Checks	24
10	Moonlight	24
10.1	The Lunar Cycle	24
10.2	Timing and Level	25
10.3	Moon Channels	25
10.4	Safety Checks	25
11	Effects	26
11.1	Overview	26
11.2	Random Numbers	26

11.3	Clouds	26
11.4	Storms	27
11.5	Lightning	27
11.6	Where Effects Run	28
11.7	Safety Checks	28
12	Light Settings	29
12.1	Settings of Each Light	29
12.2	Acclimation	29
12.3	Dimming on a Condition	30
12.4	Safety Checks	30
13	Scenes	30
13.1	Scenes	30
13.2	Starting and Ending a Scene	31
13.3	Photo Mode	32
13.4	Fades	32
13.5	Safety Checks	33
14	Computing Levels	33
14.1	Programs of Curves	33
14.2	Color Temperature Programs	34
14.3	Level Outputs and Switches	34
14.4	Driving Lights from the Controller	34
15	Groups	34
15.1	Lights That Run Together	34
15.2	Staggering	35
15.3	Clocks	35
15.4	Safety Checks	35
16	Lights That Run Their Own Programs	35
16.1	Fixture Programs	35
16.2	Writing a Fixture Program	36
16.3	Running a Fixture Program	36
16.4	Under a Controller	37
16.5	Safety Checks	38
17	Heat and Failures	38
17.1	Fixture Temperature	38
17.2	Fail-Safe Configuration	38
17.3	Lost Inputs	39
17.4	Power, Restarts, and the Clock	39
17.5	A Failed Controller	40
17.6	Safety Checks	40
18	Sharing	40
18.1	Program Documents	40
18.2	Importing and Exporting	41
18.3	Licenses, Credit, and Privacy	41
19	History	41
20	Recommended Values	42
21	Conformance	43
21.1	Controllers	43
21.2	Lights	44
21.3	Documents	44
21.4	Schema and Tools	44
21.5	Test Procedures	44
22	Security Considerations	52
Appendix A — Approval Checklist		52
Appendix B — Examples (Informative)		53
Appendix C — Commercial Features (Informative)		53
Appendix D — Open Issues (Informative)		54

3 Scope

This Specification defines:

- lighting programs: curves of level over the day for each kind of channel, or of color temperature and intensity, with defined interpolation;
- how a program runs on a light with different channels from those it was written for;
- the sun's timing, from a location or a seasonal table, and an astronomical method that gives the same result on every product;
- moonlight that follows the moon's phase, or a lunar cycle moved or set for breeding;
- clouds, storms, and lightning, with randomness that repeats exactly;
- acclimation over days, owners' scales and caps, and dimming on a condition;
- scenes, such as photo mode;
- groups of lights that run one program together, or staggered;
- the programs that lights run by themselves, and how a controller works with them;
- how lighting interacts with fixture over-temperature dimming and with fail-safe behavior;
- program documents, in which owners and products share programs; and
- a test procedure for every safety check.

This Specification does not define:

- how a controller evaluates a configuration, or the elements it hosts lighting in, which the Automation specification defines;
- the `light`, `level-output`, and `switch` classes, which the Device Classes specification defines [OAP-DC];
- standard spectra of channel kinds: each light may publish its own [OAP-DC Section 7.15], which Sections 8.5 and 19 use (Appendix D, issue 1);
- how much light a system needs, or the values to aim for;
- the optical, thermal, and electrical design of lights, or their safety certification;
- how alarms are delivered, which the Alarms and Notifications specification defines; or
- cameras and image capture during photo mode.

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.
- **[RFC 3339]** Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.
- **[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>.
- **[FIPS 180-4]** National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS PUB 180-4, August 2015, <https://doi.org/10.6028/NIST.FIPS.180-4>.
- **[TZDB]** Internet Assigned Numbers Authority, "Time Zone Database", <https://www.iana.org/time-zones>.
- **[POSIX]** IEEE Std 1003.1-2024, "The Open Group Base Specifications Issue 8", the TZ environment variable, <https://pubs.opengroup.org/onlinepubs/9799919799/>.
- **[SPDX]** The Linux Foundation, "SPDX License List" and "SPDX License Expressions", <https://spdx.org/licenses/>.
- **[OAP-DM]** Open Aquarium Project, "Open Aquarium Data Model", Revision 0.4, [specs/data-model/spec.md](https://github.com/open-aquarium/specs/blob/main/data-model/spec.md).
- **[OAP-DC]** Open Aquarium Project, "Open Aquarium Device Classes", Revision 0.5, [specs/device-classes/spec.md](https://github.com/open-aquarium/specs/blob/main/device-classes/spec.md).

- **[OAP-FS]** Open Aquarium Project, “Open Aquarium Fail-Safe Behavior”, Revision 0.4, [specs/fail-safe-behavior/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-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-SA]** Open Aquarium Project, “Open Aquarium Security and Access”, Revision 0.3, [specs/security-and-access/spec.md](#).

4.2 Informative References

- **[OAP-FP]** Open Aquarium Project, “Open Aquarium Flow Programs”, Revision 0.3, [specs/flow-programs/spec.md](#).
- **[OAP-WT]** Open Aquarium Project, “Open Aquarium Water Testing”, Revision 0.3, [specs/water-testing/spec.md](#).
- **[OAP-MM]** Open Aquarium Project, “Open Aquarium Matter Mapping”, Revision 0.3, [specs/matter-mapping/spec.md](#).
- **[NOAA]** NOAA Global Monitoring Laboratory, “Solar Calculation Details”, <https://gml.noaa.gov/grad/solcalc/calcdetails.html>. The sun’s equations of Section 9.2 are those of its calculator.
- **[Meeus]** Meeus, J., “Astronomical Algorithms”, second edition, 1998. The moon’s equations of Section 9.4 are the series of its Chapter 47, shortened to their largest terms, with its formulas for sidereal time, for right ascension and declination, and for the phase angle (formula 48.4).
- **[SplitMix]** Vigna, S., “splitmix64.c”, public domain, <https://prng.di.unimi.it/splitmix64.c>. The generator of Section 11.2 is this SplitMix64, the generator of Steele, Lea, and Flood, “Fast Splittable Pseudorandom Number Generators”, OOPSLA 2014, with these constants.
- **[WCAG]** World Wide Web Consortium, “Web Content Accessibility Guidelines (WCAG) 2.2”, Success Criterion 2.3.1, Three Flashes or Below Threshold, <https://www.w3.org/TR/WCAG22/>.
- **[ITU-R BT.1702]** International Telecommunication Union, Recommendation ITU-R BT.1702-3, “Guidance for the reduction of photosensitive epileptic seizures caused by television”, November 2023, <https://www.itu.int/rec/R-REC-BT.1702-3-202311-I/en>.
- **[CIE]** International Commission on Illumination, “ILV: International Lighting Vocabulary”, CIE S 017:2020, term 17-23-068, “correlated colour temperature”, <https://cie.co.at/eilvterm/17-23-068>.
- **[CIE CMF]** International Commission on Illumination, “CIE 1931 colour-matching functions, 2 degree observer”, data table, <https://doi.org/10.25039/CIE.DS.xvudnb9b>. Section 8.5 finds the color of a channel’s light with these functions.
- **[Priest]** Priest, I. G., “A Proposed Scale for Use in Specifying the Chromaticity of Incandescent Illuminants and Various Phases of Daylight”, Journal of the Optical Society of America 23(2), 41, 1933, <https://doi.org/10.1364/JOSA.23.000041>.
- **[Kyba]** Kyba, C. C. M., et al., “How bright is moonlight?”, Astronomy & Geophysics 58(1), 1.31–1.32, 2017, <https://doi.org/10.1093/astrogeo/atx025>.
- **[Craggs]** Craggs, J., et al., “Inducing broadcast coral spawning ex situ: Closed system mesocosm design and husbandry protocol”, Ecology and Evolution 7(24), 11066–11078, 2017, <https://doi.org/10.1002/ece3.3538>.
- **[Koukoumaftsis]** Koukoumaftsis, L., et al., “Controlled out-of-season spawning of reef-forming corals using offset environmental cues”, Proceedings of the Royal Society B 292(2055), 20251558, 2025, <https://doi.org/10.1098/rspb.2025.1558>.
- **[Lin]** Lin, C.-H., et al., “Moonrise timing is key for synchronized spawning in coral *Dipsastraea speciosa*”, Proceedings of the National Academy of Sciences 118(34), e2101985118, 2021, <https://doi.org/10.1073/pnas.2101985118>.
- **[Kaniewska]** Kaniewska, P., et al., “Signaling cascades and the importance of moonlight in coral broadcast mass spawning”, eLife 4, e09991, 2015, <https://doi.org/10.7554/eLife.09991>.

- **[DiPerna]** DiPerna, S., et al., "Effects of variability in daily light integrals on the photophysiology of the corals *Pachyseris speciosa* and *Acropora millepora*", PLoS ONE 13(9), e0203882, 2018, <https://doi.org/10.1371/journal.pone.0203882>.
- **[CCAC]** Canadian Council on Animal Care, "CCAC guidelines: Zebrafish and other small, warm-water laboratory fish", December 2020, https://ccac.ca/Documents/Standards/Guidelines/CCAC_Guidelines-Zebrafish_and_other_small_warm-water_laboratory_fish.pdf.
- **[DLC]** DesignLights Consortium, "Technical Requirements for LED-Based Horticultural Lighting", Version 4.0, March 2025, https://designlights.org/dlc_hort_technical_requirements_v4-0_finalpolicy/.
- **[DALI]** DALI Alliance, "DALI Colour Control", <https://www.dali-alliance.org/dali/colour.html>.

5 Terms, Definitions, and Conventions

5.1 Requirements Language

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

5.2 Terms and Definitions

Terms defined in the Device Classes specification [OAP-DC], such as device, endpoint, feature, and controller, in the Fail-Safe Behavior specification [OAP-FS], such as fail-safe behavior, inactive state, and unavailable input, in the Automation specification [OAP-AU], such as output, input, condition, function, mode, override, resolved state, saved state, and temporary state, and in the Alarms and Notifications specification [OAP-AN], such as condition code, have the same meaning here.

Light: One output that a lighting function drives: an endpoint of the `light` class, or a `level-output` or `switch` endpoint that carries one kind of light (Section 7.3).

Channel: One independently dimmable group of LEDs of a light, as in the `channels` attribute of the `light` class, or the single channel of a `level-output` or `switch` output.

Kind: The color of a channel, one of the channel kinds of the `light` class, such as `cool-white` or `royal-blue` [OAP-DC Section 9.4].

Correlated color temperature: The temperature of the black body whose color is nearest that of a light [CIE], in kelvin; "color temperature" in this Specification.

Level: How bright a channel is, from 0 % (off) to 100 % (its full output).

Program: What a group of lights shows over the day and night: its sun, curves or color temperature, moonlight, effects, and scenes (Section 8).

Program of curves: A program that gives a curve for each of some kinds of channel.

Color temperature program: A program that gives a correlated color temperature and an intensity over the day, instead of curves (Section 8.4).

Point: A time of day and a level, or a color temperature and an intensity, in a curve.

Curve: The points of one or more kinds, and how levels change between them.

Mired: One million divided by a color temperature in kelvin. Equal steps in mireds look like equal changes of color.

Mix: The level of each of some kinds that shows one color temperature at full intensity on a light without the `color-temperature` feature (Section 8.5).

Photosynthetic photon flux: The photons from 400 nm to 700 nm that a light emits each second, in $\mu\text{mol/s}$, estimated from the `spectrum` and `photonFlux` of its channels [OAP-DC Section 7.15.4].

Daylight: What a program's curves or color temperature show, with its effects, as distinct from moonlight.

Sun source: Where a program's sunrise, solar noon, and sunset come from: a location, a seasonal table, or the configuration (Section 9.1).

Solar day: The sunrise, solar noon, and sunset that a program uses for one local date (Section 9.3).

Shift: A time added to every sun and moon event, so that a distant location's day falls at a convenient time where the aquarium is (Section 9.3).

Effect: A cloud, a storm, or lightning (Section 11).

Burst: One to three lightning flashes in quick succession (Section 11.5).

Scene: A fixed look that a light shows for a while instead of its program, such as photo mode (Section 13).

Acclimation: Daylight that rises from a share of the program to all of it over some days (Section 12.2).

Scale: A share by which one light's daylight is multiplied, so that one program suits lights of different strengths (Section 12.1).

Cap: The highest level the owner allows a channel, whatever the program asks for (Section 12.1).

Group: The lights of one lighting function, which run one program together, each with its own offset (Section 15).

Offset: The time by which one light of a group runs behind the program (Section 15.2).

Restore fade: The fade with which a light returns to its program after a power-up, a stop, or a change of program (Section 13.4).

Lighting function: An Automation function that runs one program on a group of lights (Section 7.2).

Fixture program: A program as a light runs it by itself, with that light's settings, written to its `program` attribute (Section 16.1).

Program document: A file of programs that owners and products share (Section 18).

Thermal dimming: A light's own reduction of its output because it is too hot, shown by its `thermalDimming` attribute [OAP-DC Section 9.4].

5.3 Notation

JSON member names, function types, condition codes, channel kinds, and the names of attributes, commands, and faults appear in `monospace`. Durations are in seconds, and levels and intensities in %. Color temperatures are in kelvin (K), angles in degrees, and latitudes and longitudes in degrees (WGS 84), north and east positive. Units are Data Model unit codes [OAP-DM]. Timestamps are Data Model Timestamps: RFC 3339 dates and times with a UTC offset [RFC 3339]. References to other Open Aquarium specifications name them, as in "Automation Section 17.3", or use their tags from Section 4, as in "[OAP-AU Section 17.3]". A section number with a third part refers to a numbered item of that section: "Section 8.3.3" means item 3 of Section 8.3. Numbers marked provisional are to be confirmed with the breadboard controller, prototype lights, and owners' experience (Appendix D). In equations, `sin`, `cos`, and `tan` take degrees, `asin`, `acos`, and `atan2` return degrees, and "mod" gives a result from 0 up to, but not including, its divisor.

6 Introduction

6.1 Purpose

Commercial controllers and aquarium lights run lighting programs, and their programs are much alike: points through the day for each channel, a sunrise and sunset, a moon, clouds, an acclimation for new lights, and a photo mode (Appendix C). Yet none of them can run another's program. A program is written for one light's channels, its timing comes from one product's idea of the sun, and its clouds are different on every product, even when the settings are the same. An owner who replaces a light, or whose lights come from two makers, rebuilds their program by hand, and lights from two makers over one tank never quite agree.

This Specification defines a lighting program once, so that it means the same on every product. It is written for kinds of channel rather than for one light, its sun and moon come from one published method, and its clouds and storms come from a published random number generator, keyed by the date, so that every light in a tank, and every product, shows the same sky.

6.2 Design Principles

1. **Kinds, not products.** Programs name kinds of channel; a light maps them to its own channels, and a channel the program does not describe stays off (Section 8.6).
2. **The same sky everywhere.** Sun, moon, and weather are computed by published methods from the program, the date, and a seed, so that every conforming product shows the same thing at the same moment (Sections 9 and 11).
3. **The owner's limits are the most.** Caps, scales, and acclimation apply to everything a light shows, from any source (Section 12).
4. **Nothing sudden.** Changes fade, flashes are limited, and a light returns to its program gradually (Sections 11.5 and 13.4).
5. **The light protects itself.** A light's thermal limits override every request, and nobody makes up for them (Section 17.1).
6. **The same program on the light.** A light that runs its own program follows the same rules as a controller, so that it can keep the day when the controller is gone (Section 16).

6.3 Overview

Figure 1 shows how a light's levels are made from its program and its settings.

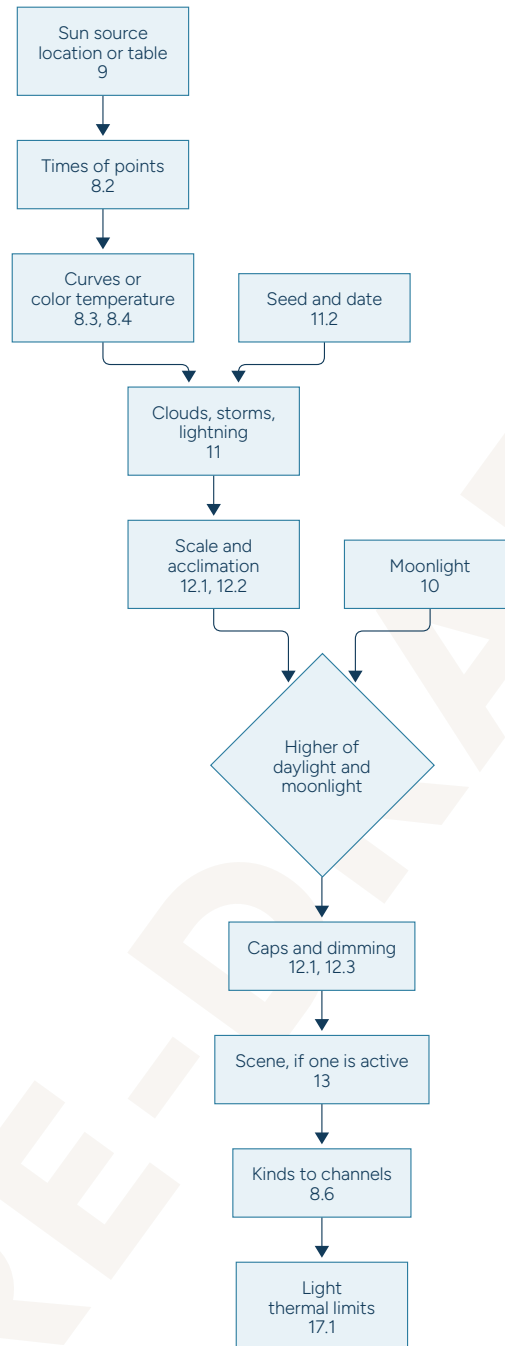


Figure 1 — How a light's levels are made

Table 1 — Parts of this Specification

Part	What it does	Section
Common requirements	Where programs run, the lighting function, the outputs it drives, and how lighting fits Automation.	7
Programs	Points, curves, color temperature, and how kinds map to a light's channels.	8
Sun and moon	Where sun times come from, and the astronomical method.	9
Moonlight	The lunar cycle, moved or set for breeding, and when moonlight shows.	10
Effects	Clouds, storms, and lightning that repeat exactly.	11
Light settings	Scales, caps, acclimation, and dimming on a condition.	12
Scenes	Fixed looks for a while, such as photo mode, and fades.	13
Computing levels	How the parts combine into each channel's level.	14

Part	What it does	Section
Groups	Lights that run together, or staggered.	15
Fixture programs	Lights that run their own programs, under a controller or without one.	16
Heat and failures	Thermal dimming, fail-safe behavior, lost inputs, power, and the clock.	17
Sharing	Program documents, importing, licenses, and privacy.	18

A controller runs programs as a function type that the Automation specification hosts [OAP-AU Section 11.10]. Interlocks, modes, and overrides act on lights as on any other output.

6.4 Example

An owner in Munich keeps a reef with two six-channel lights from one maker, and buys a program that a reefkeeper in Brisbane wrote for a four-channel light from another. The program follows the sun of Lizard Island, on the Great Barrier Reef, moved so that its mean solar noon falls at 13:00, and has curves for `white`, `royal-blue` and `blue`, and `violet` and `uv`, a moon, light clouds, and an afternoon storm on about one day in twelve.

1. The owner's lights have a `cool-white` channel, which follows the `white` curve, and a `moon` channel, which carries the moonlight (Sections 8.6 and 10.3). The controller shows this before the program runs.
2. On 14 October, Lizard Island's sun rises at 05:52 local time there. Moved to Munich, it rises over the owner's tank at 06:34, crosses the meridian at 12:46, and sets at 18:58 (Section 9.3).
3. The right-hand light runs 300 s behind the left, so that the sun rises across the tank, and its clouds 20 s behind, so that they pass from left to right (Section 15.2).
4. Both lights compute the same clouds from the program's seed and the date, so their clouds match without any message between them (Section 11.2).
5. The lights are new, so daylight starts at 50 % of the program, and rises to all of it over 30 days (Section 12.2).
6. At 13:15 the owner presses the photo button. Both lights fade in 10 s from what they show, a passing cloud included, to the program's photo scene, and after 30 minutes they fade back to the program, even if the controller has failed meanwhile (Section 13).
7. In August the room warms, and the water rises above 28.5 °C. The controller caps both lights' programs at 50 % until it has cooled, and the cap stays on each light if the controller fails meanwhile (Section 12.3).
8. One afternoon the right-hand light's fan fails, and it dims itself and reports `overtemperature`. The controller raises an alarm, shows that the light is dimmed for heat, and does not try to raise its level (Section 17.1).
9. The owner turns on lightning for the right-hand light, after the controller warns that flashing light can trigger seizures in people who are sensitive to it. The owner leaves it off on the left-hand light, as the controller advises, because the flashes of two lights can fall a little apart and together pass three in a second (Section 11.5).
10. Overnight the Wi-Fi router fails. Both lights keep running the program by themselves, moon and all, and at 07:00 their clouds still match (Section 17.5).

7 Common Requirements

7.1 Where Programs Run

1. A program runs either on a controller, which sends each light its levels, or on a light with the `local-program` feature, which runs a fixture program written to its `program` attribute [OAP-DC Section 9.4]. A light member's `runOn` says which (Table 17).
2. The same rules apply to both, so a light that runs a program by itself shows what a controller would have sent it (Sections 7.10 and 16.3).
3. Only `light` endpoints run programs by themselves; a `level-output` or `switch` output is always driven by its controller.

4. A controller MUST run lighting functions in its control process, as Automation Section 17.1.3 requires, so that if they stop, the lights apply their fail-safe behavior.
5. A controller SHOULD let lights that have the `local-program` feature run their own programs, so that they keep the day and night when the controller is gone (Section 17.5).

7.2 The Lighting Function

1. A lighting function is a function of the type `lighting`, which the Automation specification hosts [OAP-AU Section 11.10].
2. A lighting function runs one program on a group of lights (Section 15). It has the members of Table 2, and each of its lights the members of Table 17.

Table 2 — Members of a lighting function

Member	Type	Presence	Description
<code>id</code> , <code>type</code> , <code>name</code> , <code>notes</code> , <code>preset</code>			As for every function [OAP-AU Section 11.1]. <code>preset</code> is <code>light</code> .
<code>outputs</code>	array	REQUIRED	The output of every light.
<code>program</code>	object	REQUIRED	The program the lights run (Section 8).
<code>lights</code>	array	REQUIRED	The lights, one for each output (Section 12.1).
<code>acclimation</code>	object	OPTIONAL	An acclimation of every light of the function (Section 12.2).
<code>sceneTriggers</code>	array	OPTIONAL	Conditions that start scenes, each a <code>scene</code> and a <code>when</code> condition (Section 13.2).
<code>dim</code>	array	OPTIONAL	Caps while a condition holds, each a <code>when</code> condition and a <code>maxLevel</code> (Section 12.3).
<code>restoreFade</code>	integer (s)	OPTIONAL	The restore fade, from 0 to 1800 s. Default 900 s (Section 13.4).
<code>extensions</code>	object	OPTIONAL	Extensions, as in the Automation specification [OAP-AU Section 7.7].

3. `outputs` MUST list the `output` of every light, and nothing else, so that a controller that does not support lighting functions can still find their outputs, and treat them as Automation Section 19.2 requires for an unsupported function.
4. A lighting function defines its own behavior when an input is unavailable (Section 17.3), so it MUST NOT have `onUnavailable`.
5. A member described as a condition names a condition of the configuration [OAP-AU Section 10].
6. A configuration document that contains lighting functions MUST be valid against the Automation schema, and each lighting function MUST be valid against `$defs/lighting` in `schema/lighting-programs.schema.json`, a JSON Schema [JSON Schema].
7. A configuration MAY have several lighting functions, for example one for the display and one for a refugium. An output belongs to at most one function [OAP-AU Section 11.1].
8. Revisions 0.1 and 0.2 named the function type `org.openaquariumproject.lighting`, and kept the saved state of every lighting function under `state.extensions["org.openaquariumproject.lighting"]`, keyed by function identifier. A controller reads them as this revision's type and saved state (Section 7.7) [OAP-AU Section 7.8], and writes those of this revision.

7.3 Lights and Outputs

Table 3 — Outputs a lighting function drives

Class	How it shows a program	Light member
<code>light</code>	Each channel follows the curve for its kind, through <code>set</code> with <code>level</code> s, or a color temperature program through <code>setColorTemperature</code> with the <code>color-temperature</code> feature (Section 14).	Its channels give their kinds; <code>channels</code> may change them.
<code>level-output</code>	One channel, through <code>set</code> with <code>level</code> .	<code>kind</code> is REQUIRED.
<code>switch</code>	One channel, on while its level is above <code>threshold</code> , through <code>set</code> with <code>on</code> (Section 14.3).	<code>kind</code> is REQUIRED.

1. Each light of a lighting function is one output of a class in Table 3, or of an extension class, named by its output identifier.
2. A `level-output` or `switch` output carries one kind of light, which its light member gives as `kind`, for example the `royal-blue` channel of a light that is dimmed by a 0–10 V output. A light whose channels are each driven by a `level-output`, such as a light with a separate dimmer for each channel, is one light member for each output.
3. A `light` output's channels give their kinds in the endpoint's `channels` attribute; a light member MUST NOT have `kind`.
4. In a fixture program, channels are named by their `id` in the light's `channels` attribute, because a light knows nothing of the configuration (Section 16.1).

7.4 Levels

1. A level is a percentage of a channel's full output, from 0 to 100, and an intensity is a percentage of a light's full output at the color temperature it shows.
2. A light SHOULD make its output proportional to the level, as nearly as its drivers allow, so that 50 % gives half the light of 100 %, and MUST document how its output depends on the level. Programs then mean the same on every light, and measured light can be scaled (Section 19).
3. A light that cannot show a level above 0 but below its lowest stable output SHOULD show its lowest stable output, rather than switching the channel off or flickering.
4. Levels are numbers, and are not rounded by this Specification. A light shows them as finely as its drivers allow.
5. A level of 0 % means the channel is off.

7.5 Local Time and the Clock

1. Times of day are local times in the configuration's time zone, and follow Automation Section 7.6, including its rules for local times that do not exist or occur twice.
2. A fixture program carries the time zone as `posixTimeZone`, a POSIX TZ string [POSIX], because a light has no other way to know it (Section 16.1).
3. A light that runs its own program MUST implement POSIX TZ strings with rules (the M, J, and plain-number forms) and transition times, and MUST follow them as the Time Zone Database [TZDB] would for every date for which the configuration's `posixTimeZone` is valid [OAP-AU Section 7.2].
4. Durations, such as fades and offsets, are elapsed time.
5. While a controller's clock is not valid [OAP-FS Section 10.5.1], its lighting functions treat the time as an unavailable input (Section 17.3).
6. A light's clock is valid as Fail-Safe Behavior Section 10.5.4 defines it: while it has been set from its controller's time since the light started, or kept by a battery-backed clock, and cannot have drifted by more than 60 s since. It stays valid when the light loses contact with its controller, and when a binding reports the light's timestamps as unknown, until it could have drifted that far [OAP-FS Section 10.5.5], so a light keeps its program when its controller fails. While a light's clock is not valid, it runs no program (Section 17.4).

7.6 Holds

1. Interlocks, modes, overrides, and step programs act on lights as on any other output, in the order of Automation Table 25. A lighting function is the light's control function, at layer 10.
2. The lighting function's demand on a light is its program: the levels of Section 14 for a light the controller drives, and the light's own program for a light that runs one.
3. A higher layer can stop a light (`set on false`, or `level 0`) or run it (`set on true`); the `light` class has no level in Automation [OAP-AU Table 5]. A higher layer's demand replaces the program for as long as it lasts, and the lighting function then returns the light to its program with the restore fade (Section 13.4). A stop with no known end that an interlock or an emergency mode demands is latched [OAP-AU Section 17.5.4]: the light keeps it, and its program shows nothing, even when its controller fails, until a controller ends the stop [OAP-DC Section 7.4.13].

4. Scenes and dimming on a condition are part of the lighting function's demand, so a mode, override, or interlock that stops a light also stops its scene.
5. The Automation presets leave lights to their function in feed, maintenance, water-change, and emergency modes, and stop them in power-loss mode [OAP-AU Table 29]. A mode that should change the lights, such as maintenance, does so through a scene trigger (Section 13.2).
6. A mode that a scene trigger follows acts on the lights through it, so a Matter bridge counts the trigger when it decides whether it may offer the mode to home apps [OAP-MM Section 13.5.2].

7.7 Saved State

1. A controller keeps the saved state of each lighting function in the configuration's saved state [OAP-AU Section 17.8], as `state.functions["<function id>"]`, in the form of `$defs/savedState` of `schema/lighting-programs.schema.json`.
2. It holds `scene`: the scene that the owner started, with `id`, `started`, and `until`, while it lasts.
3. Scenes started by `sceneTriggers` are not saved; they follow their condition.
4. A controller MUST save the state whenever a scene starts or ends, and restores it after a restart (Section 17.4). An acclimation needs no saved state, because it follows the date.

7.8 Faults and Alarms

1. A controller raises the faults that a lighting function detects, which Table 4 lists, with the function as their subject [OAP-AN Section 8.2], naming the light. `program-not-loaded` concerns one light, so its subject is the light's output, as for `command-refused` [OAP-AU Section 17.9], and each light has its own alarm [OAP-AN Section 8.3].
2. Faults that a light reports itself, such as `overtemperature` while it dims for heat (Section 17.1), `hardware-failure`, and `communication-lost` (Section 17.5), raise alarms with the light's endpoint or device as their subject [OAP-AN Section 8.4]. The controller does not raise them again for the function, and shows them with it (Section 7.9).
3. Table 5 gives the code this Specification adds, which is listed with its default severity and its delay in `conditions.json`. Pumps that run their own programs [OAP-FP Section 15.2] and analyzers [OAP-WT Section 14.3] raise the same code. A condition of this Specification has no time in it: the delay in `conditions.json` is how long it lasts before its alarm is raised [OAP-AN Section 12.2.5].

Table 4 — Codes lighting raises

Code	Defined by	When	Subject	Latches
<code>input-unavailable</code>	Automation	A condition of <code>dim</code> or <code>sceneTriggers</code> is unknown, or the controller's clock is not valid (Section 17.3).	The function	No
<code>command-refused</code>	Automation	A light refused a command, such as a scene's <code>set</code> (Section 13.2).	The light's output	No
<code>unsupported</code>	Automation	The controller does not support lighting functions, or a member of one, such as its <code>method</code> (Section 21.1).	The function	No
<code>program-not-loaded</code>	This Specification	A light does not report the fixture program its controller wrote (Section 16.2).	The light's output	No

Table 5 — Conditions this Specification adds

Code	Meaning	Default	Critical for	Notes
<code>program-not-loaded</code>	A light, a pump, or an analyzer that should run its own program does not report the program its controller wrote (Section 16.2; Flow Programs Section 15.2; Water Testing Section 14.3).	<code>warning</code>	—	Raised after 60 s. Once the condition has lasted its delay, the controller drives the light or pump itself, where it can, and holds the analyzer, until the device reports the program. A program file that failed its checks, or a device that cannot run the program, is the usual cause.

7.9 Visibility

A controller MUST show the owner, for each lighting function:

1. its program's name, and, for each light, whether it runs the program itself or the controller drives it;
2. today's sunrise, solar noon, and sunset, as the program uses them, and the moon's phase;
3. for each light, the kind each channel follows, including substitutions, and the channels that stay off (Section 8.6);
4. the level of each channel now, and what holds it below the program, if anything: a cap, acclimation, dimming on a condition, thermal dimming, or a higher layer, such as "held at 50 % by 'Water hot'" or "dimmed for heat by the light";
5. an active scene, with the time it has left;
6. an acclimation, with its day and share, such as "day 12 of 30, 69 %";
7. today's clouds and storm, if any, for an owner who asks; and
8. the faults its lights report, such as `overtemperature` (Section 7.8.2).

7.10 Determinism

1. Given the same program, light settings, acclimation, scene, saved state, and clock, two conforming products MUST compute the same level for each channel at every moment, within 0.5 percentage points (provisional).
2. The exceptions are the times of sun and moon events, which may differ within the tolerances of Section 9.6, and moments within 2 s of a point, an effect, a flash, or a scene's start, fade-out, or end, where a level may be taken a little earlier or later.
3. A light that runs its own program MUST meet item 1 against the reference implementation, so that a light and its controller agree (Section 16.3).

This makes programs portable: the same program looks the same on every product, and lights from two makers over one tank agree.

8 Programs

8.1 Program Members

1. A program has the members of Table 6, and MUST be valid against `$defs/program` in `schema/lighting-programs.schema.json`.
2. A program has either `curves`, which makes it a program of curves, or `colorTemperature`, which makes it a color temperature program, and never both.
3. Members whose values are their defaults MAY be left out.

Table 6 — Members of a program

Member	Type	Presence	Description
<code>id</code>	identifier	REQUIRED	Identifies the program, as an Automation identifier [OAP-AU Section 7.3].
<code>name</code>	string	REQUIRED	Name shown to the owner.
<code>description</code>	string	OPTIONAL	What the program is for, and what it suits.
<code>author</code>	string	OPTIONAL	Who wrote the program, as they wish to be named.
<code>license</code>	string	OPTIONAL	An SPDX license expression [SPDX], under which others may use the program, such as <code>CC0-1.0</code> (Section 18.3).
<code>revision</code>	string	OPTIONAL	The author's revision of the program.
<code>created</code>	timestamp	OPTIONAL	When the program was written.
<code>water</code>	array	OPTIONAL	The Data Model water types the program suits: <code>marine</code> , <code>brackish</code> , <code>freshwater</code> .
<code>tags</code>	array	OPTIONAL	Data Model system tags the program suits, such as <code>reef</code> or <code>planted</code> [OAP-DM Section 9.7].
<code>method</code>	string	OPTIONAL	The astronomical method: <code>oap-1</code> , the method of Section 9, or a method that a later revision defines (Section 9.1.4). Default <code>oap-1</code> .
<code>seed</code>	integer	OPTIONAL	The seed of the program's effects, from 0 to 4294967295 (Section 11.2). Default 0.
<code>sun</code>	object	REQUIRED	Where sunrise, solar noon, and sunset come from (Section 9.1).

Member	Type	Presence	Description
curves	array	One of	The curves of a program of curves, 1 to 15 (Section 8.3).
colorTemperature	object	One of	The points and mixes of a color temperature program (Sections 8.4 and 8.5).
moon	object	OPTIONAL	Moonlight (Section 10).
clouds	object	OPTIONAL	Clouds (Section 11.3).
storms	object	OPTIONAL	Storms, with lightning (Sections 11.4 and 11.5).
scenes	array	OPTIONAL	Scenes, up to 16 (Section 13).
fixture	object	OPTIONAL	The <code>manufacturer</code> and <code>model</code> of the light that curves for channels are written for (Section 8.7).
extensions	object	OPTIONAL	Extensions, as in the Data Model [OAP-DM Section 7.6].

8.2 Times

Each point has a `time`, which says when it falls on each local date.

Table 7 — Times of points

Form	Example	On a local date, the moment
Time of day	"08:30"	That local time on the date, as Automation Section 7.6 defines it.
Sun event	{"event": "sunrise", "offset": 1800}	The <code>sunrise</code> , <code>solar-noon</code> , or <code>sunset</code> of the date's solar day (Section 9.3), plus <code>offset</code> , a whole number of seconds from -43200 to 43200. Default <code>offset</code> 0.
Share of the day	{"dayFraction": 0.25}	The moment that share, from 0 to 1, of the way from the date's sunrise to its sunset.

1. A time of day suits a program whose day should not move, such as a planted tank's day set around its owner's evenings.
2. A sun event keeps a ramp the same length all year, and moves the day with the seasons.
3. A share of the day stretches and shrinks with the day, so that a peak stays in the middle of it, and a ramp lasts a fixed share of it.
4. A window, such as a night or a storm's window, from a `start` to an `end`, that starts on a local date runs from `start` on that date to the first moment after it at which `end` falls on that date or the next, and lasts at most 24 hours. So a night from {"event": "sunset"} to {"event": "sunrise"} ends at the next morning's sunrise.

8.3 Curves

1. A curve has `kinds`, the channel kinds it describes, or, in a program with `fixture`, `channels`, the channels of that fixture it describes (Section 8.7); 1 to 48 `points`, each a `time` and a `level`; and an optional `interpolation`.
2. Each kind, and each channel, appears in at most one curve of a program.
3. On each local date, a curve's points fall at the moments of Section 8.2, in the order they are listed, and the points of all dates follow each other in date order. A product MUST move a point that falls earlier than the point before it in this sequence to that point's moment, so that every product resolves such points the same way.
4. Points SHOULD be listed in the order in which they fall, and the points of one date SHOULD span less than 23 hours, so that no point is moved on any date of the year. Points can still be moved where sun events and times of day are mixed, for example a fixed "20:00" after {"event": "sunset"} in midsummer, and moving them gives a defined result, which a validator reports.
5. The level of a curve at a moment is found from the last point at or before it, *a*, and the next point after it, *b*. With *f* the share of the time from *a* to *b* that has passed:
 - `linear` (the default): *a*'s level plus (*b*'s level - *a*'s level) × *f*;
 - `cosine`: as `linear`, with *f* replaced by $(1 - \cos(180^\circ \times f)) / 2$, which starts and ends each change gently; and
 - `step`: *a*'s level, until *b*'s moment. Where *a* and *b* fall at the same moment, the level is *b*'s.

6. A curve repeats every day: after the last point of one date, it continues towards the first point of the next. A curve with one point gives a constant level.

For example, a white curve with points at {"event": "sunrise", "offset": 1800} (0 %), {"event": "sunrise", "offset": 10800} (30 %), {"dayFraction": 0.5} (45 %), {"event": "sunset", "offset": -10800} (30 %), and {"event": "sunset", "offset": -1800} (0 %), with cosine interpolation, begins half an hour after sunrise, reaches 30 % two and a half hours later, peaks at midday, and ends half an hour before sunset, on every day of the year.

8.4 Color Temperature

1. A color temperature program has `colorTemperature`, with 1 to 48 points, each a time, a `colorTemperature`, in K, from 1000 to 100 000, and an `intensity`, in %; an optional `interpolation`; and optional `mixes` (Section 8.5).
2. The points, their order, and the weight f of each interpolation are as in Section 8.3.
3. Between two points, the color temperature changes evenly in mireds: with m_a and m_b the two points' mireds, the color temperature is $1\,000\,000 / (m_a + (m_b - m_a) \times f)$. The intensity changes as a level does.
4. A light with the `color-temperature` feature shows a color temperature program with `setColorTemperature`, with the color temperature kept within its `minColorTemperature` and `maxColorTemperature`.
5. A light without it, and a `level-output` or `switch` output, shows the program through `mixes` (Section 8.5): its light member's `mixes`, or else the program's. A controller MUST NOT run a color temperature program on such a light without either, and MUST tell the owner why.

Changing evenly in mireds makes a sunrise from 2700 K to 6500 K look even, because equal steps in mireds are about equally visible; changing evenly in kelvin would race through the warm colors and linger among the cool ones [Priest]. Matter and DALI carry color temperature in mireds for the same reason. A light of Device Classes revision 0.4 or later fades its color temperature in the same way when it is sent `setColorTemperature` with a `fadeTime` [OAP-DC Section 7.4.10], so that it shows a fade between two points, or a scene's fade, as Section 14.2 computes it; a light of an earlier revision may take another path between the same two ends.

8.5 Mixes

1. `mixes` gives, for each of 1 to 16 color temperatures, the level of each of some kinds at full intensity, as `colorTemperature` and `levels`.
2. Mixes are listed in order of increasing color temperature, each color temperature once.
3. At a color temperature below the first mix, the first mix applies, and above the last, the last. Between two mixes, each kind's level changes evenly in mireds between them, a kind missing from a mix counting as 0 %.
4. Each channel's level is its kind's level in the mix, found as in Section 8.6, times the intensity, divided by 100.
5. A light member MAY have `mixes`, in the form of the program's, which the light shows in place of the program's (Table 17). They suit a program to one light's channels, and let a color temperature program without mixes run on a light without the `color-temperature` feature.
6. When a color temperature program is to run on a light without the `color-temperature` feature whose channels each give their `spectrum` and `photonFlux` [OAP-DC Section 7.15], a controller SHOULD propose the light member's `mixes` from them, for the owner to confirm, as it proposes an acclimation (Section 12.2.5).

A program's mixes are an approximation that its author makes for typical channels. Mixes made from a light's own spectra fit that light, and a light with the `color-temperature` feature mixes its channels itself, which Section 8.4.4 prefers (Appendix D, issue 1).

One way to make a mix from spectra: for each channel, take its spectral photon flux at 100 %, `photonFlux` times its `spectrum` divided by the area under it; divide it by the wavelength, which gives its spectral power, apart from a constant; and find from that its tristimulus values X , Y , and Z with the CIE 1931 colour-matching functions [CIE CMF]. Then choose for each channel a level from 0 to 100 % such that the channels' summed X , Y , and Z have the chromaticity of a black body at the mix's color temperature, and the greatest Y , which is the most light at that color. Where the channels cannot reach that chromaticity, the controller proposes no mix at that color temperature, and tells the owner which of the program's color temperatures the light shows only as the

nearest mix (item 3). A mix gives one level for each kind, so the channels of one kind share a level, found from their spectra together.

8.6 Channel Kinds

1. A channel of kind *k* follows the curve for *k*. If the program has none, it follows the curve for the first of *k*'s substitutes in Table 8 that has one. If none of them has one, the channel MUST show no daylight.
2. The same rule finds a channel's level in a scene's `levels` (Section 13.1) and in a mix (Section 8.5).
3. A `moon` channel has no substitutes for daylight: it carries moonlight (Section 10.3), and daylight only from a curve for `moon`.
4. A light member's `channels` gives another kind for some of a light's channels, keyed by channel `id`, or `none` for a channel that is always off. For example, `{"6": "royal-blue"}` lets a light's spare channel, which its maker calls `other`, follow the royal blue curve.
5. Several channels of the same kind, such as two `royal-blue` channels, each follow the curve for that kind.
6. A controller MUST show the owner, before it runs a program on a light, which curve each channel follows, and which channels stay off (Section 7.9).

Table 8 — Channel kinds

Kind	Substitutes, in order	Clouds and storms dim it	Flashes for lightning	Typical peak (informative)
white	full-spectrum, cool-white, warm-white	Yes	Yes	—
cool-white	white, full-spectrum, warm-white	Yes	Yes	—
warm-white	white, full-spectrum, cool-white	Yes	Yes	—
full-spectrum	white, cool-white, warm-white	Yes	Yes	—
royal-blue	blue	Yes	No	440 to 460 nm
blue	royal-blue	Yes	No	460 to 490 nm
cyan	blue, royal-blue	Yes	No	490 to 520 nm
violet	uv, royal-blue	Yes	No	400 to 430 nm
uv	violet	Yes	No	380 to 400 nm
green	None	Yes	No	520 to 550 nm
red	deep-red	Yes	No	620 to 645 nm
deep-red	red	Yes	No	650 to 670 nm
far-red	None	Yes	No	720 to 740 nm
moon	None	No	No	—
other	None	Yes	No	—

The substitutes are close in color, so a channel never shows a color the program's author did not intend. A channel with no curve and no substitute stays off rather than being given a guess: too much of the wrong light, such as ultraviolet or far red, does more harm than too little. `channel-kinds.json` holds Table 8 in machine-readable form.

8.7 Curves for One Fixture

Programs exported from one maker's light often describe its channels one by one.

1. A program with `fixture`, a `manufacturer` and a `model`, MAY have curves with `channels`, which name channels of that fixture by their `id`.
2. On a light whose device's `manufacturer` and `model` attributes [OAP-DC Section 8.1] are exactly those of `fixture`, a channel named in such a curve follows it, rather than the curve for its kind.
3. On any other light, curves with `channels` are ignored, and channels follow their kinds as in Section 8.6.
4. A program with curves for channels SHOULD also have curves for kinds, so that it runs on other lights.

8.8 Safety Checks

Table 9 — Checks on programs

Check	Presence	Applies to	What it requires	Section
Channels without a curve stay off	REQUIRED	Controllers, lights	A channel whose kind has no curve, and no substitute with one, shows no daylight. A controller shows the owner which curve each channel follows before the program runs.	8.6
Points in order	REQUIRED	Controllers, lights	A curve's points fall in the order they are listed on every date. A point that falls before the one before it is moved to it, the same way on a controller and on a light.	8.3
Color temperature without mixes	REQUIRED	Controllers	A controller does not run a color temperature program on a light that lacks the <code>color-temperature</code> feature without mixes, its light member's or the program's, and tells the owner why.	8.4
Programs checked before use	REQUIRED	Controllers, lights	A product rejects a program document or a fixture program that is not valid, without changing what runs, and a light rejects a program it cannot run in full.	18.2
Brighter programs come in gradually	RECOMMENDED	Controllers	When a program replaces another on a light, the controller compares their daily light, and proposes an acclimation when the new one gives more than 20 % more.	18.2

9 The Sun and the Moon

9.1 Sources and Method

Table 10 — Members of `sun`

Member	Type	Presence	Description
<code>source</code>	string	REQUIRED	<code>location</code> , <code>table</code> , or <code>configuration</code> .
<code>location</code>	object	REQUIRED for <code>location</code> ; OPTIONAL for <code>table</code>	A <code>latitude</code> , from -60 to 60, a <code>longitude</code> , from -180 to 180, and an optional name. With <code>table</code> , it is used only for the moon (Section 10.2).
<code>table</code>	array	REQUIRED for <code>table</code>	A seasonal table: entries, each a month and day (<code>date</code>) and a local <code>sunrise</code> and <code>sunset</code> (Section 9.5).
<code>shift</code>	integer (s)	OPTIONAL	Added to every sun and moon event, from -86400 to 86400. Default 0. Not with <code>noonAt</code> .
<code>noonAt</code>	time of day	OPTIONAL, with a <code>location</code>	The local time at which the location's mean solar noon falls; the shift that puts it there is computed for each date (Section 9.3).
<code>shiftDays</code>	integer	OPTIONAL, <code>location</code> and <code>table</code> only	Moves the sun's seasons, and the moon with them, by whole days, from -366 to 366. Default 0.

1. With `location`, the sun is computed for that place, by the method of Sections 9.2 and 9.3. The location need not be the owner's own: it is often a reef or river that the program follows.
2. With `table`, the sun follows a seasonal table (Section 9.5).
3. With `configuration`, the program uses the configuration's `astronomy` [OAP-AU Section 10.7]: its `location`, `sunTable`, `shift`, `noonAt`, and `shiftDays`, with `sunTable` used for the sun when both are given. A configuration that runs such a program MUST have `astronomy`. Programs that follow the owner's own sun use this source, so that a shared program does not carry the owner's location (Section 18.3).
4. `method` names the method of this section, `oap-1`. A later revision may define other methods, under other names, so a program with another `method` is valid, but unsupported by a product that does not implement its method. A product MUST NOT run a program whose method it does not implement: a controller treats its lighting function as unsupported (Section 21.1.4), and a light rejects it as a program it cannot run in full (Section 16.1.4).
5. With `shiftDays` *n*, each date has the sun of the date *n* days before it, at the same time of day, and the moon follows it (Section 10.1). Coral spawning research moves a reef's seasons this way, by advancing them six months, and moves its clock with `shift` or `noonAt` [Koukoumftsis] [Craggs].

- Latitudes are limited to 60° north and south. Nearer the poles the sun may not rise or set for days, and days are longer or shorter than aquarium life needs; a program that follows such a place uses a seasonal table.

9.2 The Sun's Position

For a moment u , in POSIX time in seconds, the Julian day is $JD = u / 86400 + 2440587.5$, and $T = (JD - 2451545) / 36525$, in Julian centuries. These are the equations of the NOAA solar calculator [NOAA]:

- $L_0 = (280.46646 + T \times (36000.76983 + 0.0003032 \times T)) \bmod 360$
- $M = 357.52911 + T \times (35999.05029 - 0.0001537 \times T)$
- $e = 0.016708634 - T \times (0.000042037 + 0.0000001267 \times T)$
- $C = \sin M \times (1.914602 - T \times (0.004817 + 0.000014 \times T)) + \sin 2M \times (0.019993 - 0.000101 \times T) + 0.000289 \times \sin 3M$
- $\Omega = 125.04 - 1934.136 \times T$
- $\lambda = L_0 + C - 0.00569 - 0.00478 \times \sin \Omega$
- $\epsilon_0 = 23 + (26 + (21.448 - T \times (46.815 + T \times (0.00059 - 0.001813 \times T))) / 60) / 60$
- $\epsilon = \epsilon_0 + 0.00256 \times \cos \Omega$
- The sun's declination, $\delta = \arcsin(\sin \epsilon \times \sin \lambda)$.
- $y = \tan^2(\epsilon / 2)$
- The equation of time, in minutes, $E = 4 \times (180 / \pi) \times (y \times \sin 2L_0 - 2e \times \sin M + 4ey \times \sin M \times \cos 2L_0 - 0.5y^2 \times \sin 4L_0 - 1.25e^2 \times \sin 2M)$, where π is 3.14159...
- At latitude ϕ , the hour angle of sunrise, $H_0(\delta) = \arccos(\cos 90.833^\circ / (\cos \phi \times \cos \delta) - \tan \phi \times \tan \delta)$.

The 0.833° below the horizon allows for refraction and the size of the sun's disc, as Automation Section 10.7.1 defines sunrise.

9.3 Sunrise, Solar Noon, and Sunset

For a local date d , at the location's longitude λ_0 , with t , N , and M_0 counted in days from 1970-01-01 00:00 UTC, and $E(t)$ and $\delta(t)$ the equation of time and declination of Section 9.2 at the moment t :

- The shift, s , in seconds, is:
 - with `noonAt`, the moment `noonAt` falls on d in local time, less the moment $(n - \text{shiftDays} + 0.5 - \lambda_0 / 360) \times 86400$, where n is the number of days from 1970-01-01 to d ;
 - otherwise `shift + 86400 × shiftDays`, each 0 when it is not given.
- The reference moment, R , is the moment 12:00 falls on d in local time, less s , in POSIX time.
- The solar day's mean day number is $N = \text{floor}(R / 86400 + \lambda_0 / 360)$, and its mean noon is $M_0 = N + 0.5 - \lambda_0 / 360$.
- Solar noon: start from $t = M_0$, and twice set $t = M_0 - E(t) / 1440$.
- Sunrise: start from $t = \text{noon} - H_0(\delta(\text{noon})) / 360$, and twice set $t = M_0 - E(t) / 1440 - H_0(\delta(t)) / 360$.
Sunset: the same, with $+$ in place of $-$ before H_0 .
- Each event is $t \times 86400 + s$, in POSIX time.

The solar day of a date is the one whose mean noon, after the shift, is nearest local noon on that date. For a location in the aquarium's own time zone, it is that date's own sunrise and sunset. This is the day that Automation Section 10.7.4 takes, whenever the shifted sunrise and sunset fall on the same local date (Appendix D, issue 6).

For example, Lizard Island (14.668° S, 145.463° E) with `noonAt 13:00`, run in Munich (`Europe/Berlin`), gives on 14 October 2026 a shift of 8 h 41 min 51 s, and a sunrise at 06:34:13, solar noon at 12:46:05, and sunset at 18:58:09, local time in Munich. Solar noon falls before 13:00 because `noonAt` places the mean solar noon, and the sun runs 14 minutes ahead of its mean in October.

9.4 The Moon

For a moment u , with JD and T as in Section 9.2, these are the moon's mean elements, the largest terms of its longitude, latitude, and distance, and its phase angle, from [Meeus]:

- $L' = 218.3164477 + 481267.88123421 T - 0.0015786 T^2 + T^3 / 538841 - T^4 / 65194000$
- $D = 297.8501921 + 445267.1114034 T - 0.0018819 T^2 + T^3 / 545868 - T^4 / 113065000$

3. $M = 357.5291092 + 35999.0502909 T - 0.0001536 T^2 + T^3 / 24490000$
4. $M' = 134.9633964 + 477198.8675055 T + 0.0087414 T^2 + T^3 / 69699 - T^4 / 14712000$
5. $F = 93.2720950 + 483202.0175233 T - 0.0036539 T^2 - T^3 / 3526000 + T^4 / 863310000$
6. The moon's longitude, $\lambda = L' + 6.288774 \sin M' + 1.274027 \sin(2D - M') + 0.658314 \sin 2D + 0.213618 \sin 2M' - 0.185116 \sin M - 0.114332 \sin 2F + 0.058793 \sin(2D - 2M') + 0.057066 \sin(2D - M - M') + 0.053322 \sin(2D + M') + 0.045758 \sin(2D - M) - 0.040923 \sin(M - M') - 0.034720 \sin D - 0.030383 \sin(M + M')$
7. Its latitude, $\beta = 5.128122 \sin F + 0.280602 \sin(M' + F) + 0.277693 \sin(M' - F) + 0.173237 \sin(2D - F) + 0.055413 \sin(2D - M' + F) + 0.046271 \sin(2D - M' - F)$
8. Its distance, in km, $\Delta = 385000.56 - 20905.355 \cos M' - 3699.111 \cos(2D - M') - 2955.968 \cos 2D - 569.925 \cos 2M'$
9. Its phase angle, $i = 180 - D - 6.289 \sin M' + 2.100 \sin M - 1.274 \sin(2D - M') - 0.658 \sin 2D - 0.214 \sin 2M' - 0.110 \sin D$
10. **The illuminated fraction**, $k = (1 + \cos i) / 2$, from 0 at new moon to 1 at full moon.
11. With ϵ_0 of Section 9.2, its right ascension, $\alpha = \text{atan2}(\sin \lambda \times \cos \epsilon_0 - \tan \beta \times \sin \epsilon_0, \cos \lambda)$, and its declination, $\delta = \text{asin}(\sin \beta \times \cos \epsilon_0 + \cos \beta \times \sin \epsilon_0 \times \sin \lambda)$.
12. The Greenwich mean sidereal time, $\theta_0 = 280.46061837 + 360.98564736629 \times (\text{JD} - 2451545) + 0.000387933 T^2 - T^3 / 38710000$.
13. At latitude ϕ and longitude λ_0 , the hour angle $H = \theta_0 + \lambda_0 - \alpha$, and the altitude of the moon's centre, $h = \text{asin}(\sin \phi \times \sin \delta + \cos \phi \times \cos \delta \times \cos H)$.
14. With the horizontal parallax $\pi_M = \text{asin}(6378.14 / \Delta)$, **the moon's altitude** seen from the location is $a = h - \pi_M \times \cos h$.

9.5 Seasonal Tables

1. A seasonal table gives local sunrise and sunset times for some days of the year, as Automation Section 10.7.2 does for `sunTable`.
2. Each date MUST appear only once, and each entry's **sunrise** MUST be earlier than its **sunset**.
3. On a local date, sunrise and sunset are interpolated linearly, in seconds after midnight, by day of the year, counted in a year of 365 days with 29 February taken as 28 February, between the entries before and after it, wrapping from the last entry to the first. With one entry, every day is the same. With `shiftDays` n , the table is read for the date n days before.
4. They are local times on the date, as Automation Section 7.6 defines them, moved by the date's shift (Section 9.3.1) less $86400 \times \text{shiftDays}$, whose whole days item 3 has already applied: by `shift`, or, with `noonAt`, by the time that puts the location's mean solar noon at `noonAt` on the date. Solar noon is halfway between them.
5. An entry MAY give `dayLight`, the share of the program's daylight for that day of the year, from 0 to 100 %, interpolated in the same way, with a default of 100 %. It multiplies the curves, or the intensity, before effects, so that a season can be duller or brighter, as researchers who spawn corals follow a reef's monthly light [Craggs]. It does not change moonlight, lightning, or scenes.
6. A table gives no position for the moon, so a program with a table and moonlight that follows the moon in the sky also needs a `location` (Section 10.2).

A table suits a program whose day should follow the seasons without following any one place, a program that follows a reef's recorded months, and a program for a place nearer the poles than 60°.

9.6 Accuracy

1. A product MUST compute with at least the precision of IEEE 754 binary64 floating point, because Julian days need it.
2. For latitudes from 60° S to 60° N, and dates from 2000 to 2100, a product MUST compute sunrise, solar noon, and sunset within 2 s, the moon's illuminated fraction within 0.001, and its altitude within 0.01°, of the reference implementation (Section 21.4) (provisional).
3. `test-vectors.json` gives results of the reference implementation to check against.

Checked for this Specification against PyEphem 4.2.1, an independent ephemeris, the method gives sunrise and sunset within 6 s, the illuminated fraction within 0.0025, and the moon's altitude within 0.11°, at latitudes

up to 60° and dates from 2023 to 2030. That is well within the tolerances of Automation Section 10.7.5, so a controller can use this method for Automation's sun and moon events as well; found from the moon's altitude of Section 9.4.14, with the top of the disc and refraction allowed for, moonrise falls within a minute of the ephemeris.

9.7 Checks

Table 11 — Checks on the sun and moon

Check	Presence	Applies to	What it requires	Section
Sun and moon by the method	REQUIRED	Controllers, lights	Sunrise, solar noon, sunset, and the moon's illuminated fraction and altitude come from the method of Sections 9.2 to 9.5, within the tolerances of Section 9.6.	9.6
Local time on a light	REQUIRED	Lights	A light follows POSIX TZ strings as the Time Zone Database would.	7.5

10 Moonlight

Table 12 — Members of `moon`

Member	Type	Presence	Description
<code>level</code>	number (%)	REQUIRED	Moonlight at full moon.
<code>minLevel</code>	number (%)	OPTIONAL	Moonlight at new moon. Default 0.
<code>kinds</code>	array	OPTIONAL, programs of curves only	The kinds that carry moonlight. Default <code>["moon"]</code> (Section 10.3).
<code>colorTemperature</code>	number (K)	REQUIRED in color temperature programs	The color temperature of moonlight.
<code>timing</code>	string	OPTIONAL	<code>night</code> , for moonlight through a window of the night, or <code>moon</code> , for moonlight while the moon is up. Default <code>night</code> .
<code>start, end</code>	time	OPTIONAL, <code>night</code> only	The window of the night. Default <code>{"event": "sunset"}</code> and <code>{"event": "sunrise"}</code> .
<code>fade</code>	integer (s)	OPTIONAL, <code>night</code> only	The fade at each end of the window, from 0 to 7200 s. Default 1800 s.
<code>cycle</code>	string or object	OPTIONAL	<code>astronomical</code> , the moon in the sky, or a cycle of its own, with <code>period</code> , in days, and <code>fullMoon</code> , a timestamp. Default <code>astronomical</code> .
<code>shiftDays</code>	integer	OPTIONAL, <code>astronomical</code> only	Moves the cycle by whole days, from -29 to 29. Default 0.

10.1 The Lunar Cycle

1. A product MUST compute the moon's illuminated fraction as this section says. With the `astronomical` cycle, the fraction at a moment u is k of Section 9.4 at the moment $u - s - 86400 \times \text{shiftDays}$, where s is the program's shift for the local date of u (Section 9.3).
2. With a positive `shiftDays` n , the moon is shown as it was n days before, so every full moon comes n days later than in the sky. Whole days keep each night's moon in step with its phase, so that, as in nature, the moon rises later each night after full moon, and the nights begin darker: the hour of darkness between sunset and moonrise is what sets corals spawning [Lin].
3. Owners use `shiftDays` for breeding: to bring the full moon of a coral's or fish's home waters to the aquarium, or to move spawning to a date when someone can watch. A program that follows a reef's sun with `location` already follows that reef's moon, and the sun's `shiftDays` moves the reef's seasons and its moon together (Section 9.1.5).
4. With a cycle of its own, the moon's age at u is $a = ((u - \text{fullMoon}) / (86400 \times \text{period}) + 0.5) \bmod 1$, from 0 at new moon, and its illuminated fraction is $(1 - \cos(360^\circ \times a)) / 2$. `period` is from 7 to 60 days, and

`fullMoon` is the moment of any one full moon. Such a cycle serves breeding programs that keep a cycle apart from the sky.

10.2 Timing and Level

1. A product **MUST** compute moonlight as this item says. Moonlight at a moment is $B = \text{minLevel} + (\text{level} - \text{minLevel}) \times k$, times the timing's share:
 - with `night`, in the window from `start` to `end` that starts on each local date (Section 8.2.4), the share rises from 0 to 1 over `fade` after the window starts, and falls back to 0 over `fade` before it ends, each fade being shortened to half the window if the window is shorter than two fades, and is 0 outside the window; and
 - with `moon`, the share is the moon's altitude α (Section 9.4) at the moment of Section 10.1.1, at the program's location, divided by 10° , kept within 0 and 1, so that moonlight shows while the moon is up, fading in as it rises through its first 10° and out as it sets (provisional).
2. `moon` timing needs a location: the sun's `location`, a table's `location`, or the configuration's, and the `astronomical` cycle.
3. Moonlight follows the program's time, after the light's offset (Section 15.2), and is not changed by clouds, storms, scale, or acclimation. Caps and dimming on a condition apply to it (Section 14).
4. A full moon gives about 0.05 to 0.3 lux [Kyba], and coral research systems set it at 0.3 to 1 lux at the water [Lin] [Koukoumftsis], while 1 % of a reef light's blue can be thousands of times brighter and has stopped corals spawning [Kaniewska]. Moonlight **SHOULD** therefore come from a channel made for it, at a few percent of that channel at most; a validator warns above 5 % (provisional), and when moonlight would go to a blue channel (Section 10.3).
5. Corals kept dark all night did not spawn either [Kaniewska], so a lunar program needs moonlight on the right nights as well as darkness on the others.

10.3 Moon Channels

1. In a program of curves, moonlight goes to the channels whose kind is in `kinds`.
2. On a light that has none of those kinds, moonlight **MUST** go to its `royal-blue` channels, or, if it has none, to its `blue` channels.
3. A channel that carries moonlight **MUST** show the higher of its daylight and its moonlight, so moonlight never dims the day.
4. In a color temperature program, moonlight is an intensity at the moon's `colorTemperature`. While moonlight's intensity is higher than daylight's, the light shows moonlight's color temperature and intensity, and otherwise daylight's.

10.4 Safety Checks

Table 13 — Checks on moonlight

Check	Presence	Applies to	What it requires	Section
The lunar cycle and its shift	REQUIRED	Controllers, lights	Moonlight follows the illuminated fraction of the moon in the sky, moved by whole days with <code>shiftDays</code> , or a cycle of its own.	10.1
When moonlight shows	REQUIRED	Controllers, lights	With the timing <code>moon</code> , moonlight shows while the moon is up, fading over its first 10° ; with <code>night</code> , it shows in its window, with its fades.	10.2
Moonlight never dims the day	REQUIRED	Controllers, lights	A channel that carries moonlight shows the higher of its daylight and its moonlight. On a light without a moon channel, royal blue, or else blue, carries moonlight.	10.3

11 Effects

11.1 Overview

1. A product **MUST** make the clouds, storms, and lightning of each local date from the program's `seed`, the date, and the effect, by the generator and the steps of Sections 11.2 to 11.5, so that every product makes the same effects for the same program on the same date.
2. Effects act only on daylight: they dim it, and lightning briefly raises it.
3. While a scene is shown, its level takes the place of the program's effects. Clouds and storms fade out with the scene's fade-in, and back in with its fade-out, because the scene fades from and to what the light shows (Section 13.4). Lightning **MUST NOT** flash from the start of a scene's fade-in to the end of its fade-out, nor during a restore fade (Section 13.4.3), however the light fades: by itself, when a revert falls due, or after `runProgram` [OAP-DC Sections 7.4.12 and 7.13].
4. Effects follow the light's effect time, after its `cCloudOffset` (Section 15.2).
5. Each effect falls in a window from its `start` to its `end` (Section 8.2.4), by default from sunrise to sunset. The effects of a date are those whose window starts on it.

11.2 Random Numbers

The generator is SplitMix64 [SplitMix], with all arithmetic on unsigned 64-bit integers, modulo 2^{64} .

1. For a program's `seed` S , a local date whose day number n is its number of days after 1970-01-01, negative before it, and an effect's stream c , 1 for clouds, 2 for storms, and 3 for lightning, the generator starts with $x = ((S \bmod 2^{32}) \times 2^{32} + (n \bmod 2^{32})) \text{ XOR } ((c \times 0x9E3779B97F4A7C15) \bmod 2^{64})$.
2. Each draw sets $x = (x + 0x9E3779B97F4A7C15) \bmod 2^{64}$, then $z = x$, $z = ((z \text{ XOR } (z >> 30)) \times 0xBF58476D1CE4E5B9) \bmod 2^{64}$, $z = ((z \text{ XOR } (z >> 27)) \times 0x94D049BB133111EB) \bmod 2^{64}$, and returns $z \text{ XOR } (z >> 31)$.
3. A whole number from a to b is $a + (\text{draw} \bmod (b - a + 1))$. A chance of $p\%$ is true when $(\text{draw} \bmod 100) < p$.
4. Draws **MUST** be made in exactly the order that Sections 11.3 to 11.5 give, whether or not their results are used.

For example, with `seed` 4207, on 14 October 2026 (day 20740), the clouds' generator starts at `0x9E3769D67F4A2D11`, and its first draw is 7152819103702853355, which is 55 modulo 100. So with `days` 70, that date has clouds. `test-vectors.json` gives further draws, and the effects of the example programs on several dates.

11.3 Clouds

Table 14 — Members of `cClouds`

Member	Type	Presence	Description
<code>coverage</code>	integer (%)	REQUIRED	About how much of the window is under cloud on a cloudy day, from 1 to 90.
<code>days</code>	integer (%)	OPTIONAL	The chance that a date has clouds, from 1 to 100. Default 100.
<code>duration</code>	object	OPTIONAL	The shortest and longest cloud, <code>min</code> and <code>max</code> , whole seconds, at least 10. Default 120 to 600 s.
<code>depth</code>	object	REQUIRED	The least and most a cloud dims daylight, <code>min</code> and <code>max</code> , whole percent, at most 95.
<code>fade</code>	integer (s)	OPTIONAL	How long a cloud takes to come and go, from 0 to 600 s. Default 30 s.
<code>start, end</code>	time	OPTIONAL	The window. Default sunrise to sunset.
<code>kinds</code>	array	OPTIONAL, programs of curves only	The kinds that clouds dim. Default every kind but <code>moon</code> .

On each local date:

1. The window runs from W_0 to W_1 , and its length is $L = \text{floor}(W_1 - W_0)$ seconds.
2. With the clouds' generator, if a chance of `days %` is false, the date has no clouds.
3. The mean cloud is $D = \text{floor}((\text{duration.min} + \text{duration.max}) / 2)$ seconds, and the mean gap between clouds $G = \text{floor}(D \times (100 - \text{coverage}) / \text{coverage})$.
4. Starting from $t = 0$, repeat: draw a gap, a whole number from $\text{floor}(G / 2)$ to $G + \text{floor}(G / 2)$; draw a duration, from `duration.min` to `duration.max`; draw a depth, from `depth.min` to `depth.max`; add the gap to t ; if t plus the duration is more than L , stop; otherwise there is a cloud from $W_0 + t$, for the duration, with the depth; and add the duration to t .
5. At x seconds after a cloud starts, its share is the smallest of 1, x / f , and $(\text{duration} - x) / f$, where f is the smaller of `fade` and half the duration; with $f = 0$, the share is 1. Daylight of the kinds it dims is multiplied by $1 - \text{depth} / 100 \times \text{share}$.
6. A cloud with a short `fade` changes daylight suddenly, which startles fish (Section 13.4). Clouds SHOULD have a `fade` of at least 10 s (provisional), and a validator warns about a shorter `fade` when `depth.max` is 10 % or more, which can change a channel by 10 percentage points or more at once.

11.4 Storms

Table 15 — Members of `storms` and `lightning`

Member	Type	Presence	Description
<code>chance</code>	integer (%)	REQUIRED	The chance that a date has a storm, from 1 to 100.
<code>duration</code>	object	OPTIONAL	The shortest and longest storm, <code>min</code> and <code>max</code> , whole seconds, from 10 to 3600. Default 900 to 2700 s.
<code>depth</code>	object	REQUIRED	The least and most a storm dims daylight, <code>min</code> and <code>max</code> , whole percent.
<code>fade</code>	integer (s)	OPTIONAL	How long a storm takes to come and go, from 0 to 3600 s. Default 600 s.
<code>start, end</code>	time	OPTIONAL	The window in which a storm falls whole. Default sunrise to sunset.
<code>kinds</code>	array	OPTIONAL, programs of curves only	The kinds that storms dim. Default every kind but <code>moon</code> .
<code>lightning</code>	object	OPTIONAL	Lightning during the storm.
<code>lightning.rate</code>	number	REQUIRED	Bursts a minute, on average, from 0.1 to 10.
<code>lightning.level</code>	number (%)	REQUIRED	The level of a flash.
<code>lightning.flashes</code>	integer	OPTIONAL	The most flashes in a burst, from 1 to 3. Default 2.
<code>lightning.kinds</code>	array	OPTIONAL, programs of curves only	The kinds that flash, never <code>red</code> , <code>deep-red</code> , or <code>far-red</code> . Default <code>white</code> , <code>cool-white</code> , <code>warm-white</code> , and <code>full-spectrum</code> .

On each local date:

1. The window and its length L are as for clouds.
2. With the storms' generator, if a chance of `chance %` is false, or L is 0, the date has no storm.
3. Otherwise draw a duration, from `duration.min` to `duration.max`, and take the smaller of it and L ; draw a start, from 0 to L less the duration; and draw a depth, from `depth.min` to `depth.max`.
4. The storm runs from W_0 plus its start, for its duration, with fades f , the smaller of `fade` and $\text{floor}(\text{duration} / 2)$. Its share, and how it dims daylight, are as for a cloud (Section 11.3.5).
5. Clouds continue during a storm, and both dim daylight.
6. Storms SHOULD have a `fade` of at least 10 s (provisional), as clouds should (Section 11.3.6), and a validator warns about a shorter `fade` when `depth.max` is 10 % or more.

11.5 Lightning

During a storm with `lightning`, with times in milliseconds after W_0 :

1. With the lightning generator, the mean interval is $I = \text{floor}(60000 / \text{rate})$, and bursts fall from the end of the storm's first fade, $(\text{start} + f) \times 1000$, to the start of its last, $(\text{start} + \text{duration} - f) \times 1000$.

2. Starting from *t* at the first of these, repeat: draw a wait, a whole number from $\text{floor}(l / 2)$ to $l + \text{floor}(l / 2)$, and take the larger of it and 5000; draw a count of flashes, from 1 to *flashes*; add the wait to *t*; for each flash, draw its length, from 60 to 120 ms; it is on from *t* for its length; add its length to *t*; and, before every flash but the first, first draw a gap, from 300 to 450 ms, and add it to *t*; if *t* is then after the last moment of item 1, discard this burst and stop.
3. While a flash is on, each channel of a lightning kind shows the higher of its daylight and *lightning.level*, before scale, acclimation, and caps. In a color temperature program, the intensity does.
4. A flash is a rise in a channel's level of 10 percentage points or more and its fall, as WCAG counts a flash [WCAG]. Flashes of a burst start at least 360 ms apart, a burst has at most three, and bursts are at least 5 s apart, so lightning never flashes more than three times in any second, and never for long. Three flashes in a second is the limit that web and broadcast guidance sets to protect people with photosensitive epilepsy, and flashes 360 ms apart are acceptable to broadcasters at any brightness [WCAG] [ITU-R BT.1702].
5. A light MUST NOT flash more than three times in any second, whatever it is asked to show. A light that cannot switch as fast as a program asks MAY lengthen flashes or leave them out, but MUST NOT add any.
6. Lightning never flashes red: broadcast guidance treats flashes of saturated red as more dangerous [ITU-R BT.1702].
7. Lightning is off unless the owner turns it on for a light, with its light member's *lightning* (Table 17). A controller MUST warn the owner, before turning it on, that flashing light can trigger seizures in people who are sensitive to it.
8. Lightning startles fish when it flashes in the dark, so storm windows SHOULD lie between sunrise and sunset, and *rate* SHOULD be no more than 4 bursts a minute (provisional). A validator warns otherwise.
9. Item 5 limits each light, not several lights together. The lights of a group keep their clocks within 0.5 s of their controller's (Section 15.3), so their flashes can fall up to a second apart, and two lights that each flash three times in a second could together flash six times in it, twice the limit of item 4; lights of different lighting functions, which may have different storms, are not in step at all. A configuration SHOULD turn *lightning* on for no more than one light of each lighting function, and a validator warns about a lighting function with more than one. A controller SHOULD warn the owner before turning lightning on for a second light that it controls, whether of the same lighting function or of another, and a validator warns about a configuration with lightning on for more than one light.

11.6 Where Effects Run

1. A light that runs its own program makes its own clouds, storms, and lightning (Section 16.3).
2. A controller that drives a light shows its clouds and storms through the levels it sends (Section 14.4).
3. A controller MUST NOT send lightning: neither a bus nor a network can time flashes of 60 to 120 ms reliably (provisional). A light that the controller drives shows its storm without lightning.
4. Because effects come from the seed and the date, the lights of a group show the same clouds without any message between them.

11.7 Safety Checks

Table 16 — Checks on effects

Check	Presence	Applies to	What it requires	Section
Effects repeat exactly	REQUIRED	Controllers, lights	Clouds, storms, and lightning come from the seed and the local date by the generator and the steps of Sections 11.2 to 11.5, the same on every product.	11.2
Flash limit	REQUIRED	Lights	A light never flashes more than three times in any second, from any source, and its program's lightning flashes only during a storm, at least 360 ms apart, in bursts at least 5 s apart.	11.5
Lightning only when turned on	REQUIRED	Controllers, lights	Lightning flashes only on a light whose <i>lightning</i> the owner has turned on, after a warning about photosensitive epilepsy, and never in red.	11.5
No lightning from a controller	REQUIRED	Controllers	A controller that drives a light shows its clouds and storms, but no lightning.	11.6

Check	Presence	Applies to	What it requires	Section
Scenes replace effects	REQUIRED	Controllers, lights	Lightning stops from the start of a scene's fade-in to the end of its fade-out, and during a restore fade, and clouds and storms fade out and back in with the scene, which fades from and to what the light shows.	11.1
One light flashes	RECOMMENDED	Controllers, configurations	Lightning is on for no more than one light of each lighting function; a controller warns before turning it on for a second light of any of its lighting functions, and a validator warns about a function, or a configuration, with more than one.	11.5
Clouds and storms fade gently	RECOMMENDED	Configurations	Clouds and storms fade over at least 10 s, so that daylight does not change suddenly; a validator warns about a shorter fade when the depth can reach 10 %.	11.3

12 Light Settings

12.1 Settings of Each Light

Table 17 — Members of a light

Member	Type	Presence	Description
<code>output</code>	identifier	REQUIRED	The output the light member drives.
<code>runOn</code>	string	OPTIONAL	<code>controller</code> , which sends the light its levels, or <code>light</code> , which runs a fixture program by itself (Section 16). Default <code>controller</code> .
<code>kind</code>	string	REQUIRED for <code>level-output</code> and <code>switch</code>	The kind of light the output carries (Section 7.3).
<code>threshold</code>	number (%)	OPTIONAL, <code>switch</code> only	A switch is on while its channel's level is above <code>threshold</code> . Default 0.
<code>offset</code>	integer (s)	OPTIONAL	How far the light runs behind the program, from -7200 to 7200 s. Default 0 (Section 15.2).
<code>cloudOffset</code>	integer (s)	OPTIONAL	How far its effects run behind the program's, from -300 to 300 s. Default 0 (Section 15.2).
<code>scale</code>	number (%)	OPTIONAL	The share of the program's daylight the light shows. Default 100.
<code>maxLevel</code>	number (%)	OPTIONAL	The cap on every channel, and on intensity. Default 100.
<code>maxLevels</code>	object	OPTIONAL	Caps for some kinds, and for <code>intensity</code> , keyed by kind.
<code>channels</code>	object	OPTIONAL, <code>light</code> only	Other kinds for some channels, keyed by channel <code>id</code> , or <code>none</code> (Section 8.6).
<code>lightning</code>	boolean	OPTIONAL	Whether the light flashes for lightning, which one light of a function should at most (Section 11.5.9). Default <code>false</code> (Section 11.5).
<code>mixes</code>	array	OPTIONAL, color temperature programs only	Mixes for this light, in place of the program's (Section 8.5.5).

1. `scale` multiplies daylight, lightning, and scenes, but not moonlight. It suits one program to lights of different strengths, or to a tank whose corals need less light than the program's author gave.
2. A channel's level MUST NOT be more than `maxLevel`, nor than the `maxLevels` entry for its kind. The intensity of a light that shows a color temperature MUST NOT be more than `maxLevel`, nor than `maxLevels.intensity`.
3. Caps apply to everything a light shows from its program: daylight, moonlight, lightning, and scenes.
4. Caps are the owner's statement of the most light that is safe for their livestock, such as a cap on ultraviolet, and protect against a program written for a weaker light.

12.2 Acclimation

A lighting function's `acclimation` has a `start` date, a number of `days`, from 1 to 180, and `from`, the share of daylight on the first day, from 1 to 99 %.

1. On a local date n days after `start`, with n from 0 to `days` - 1, the light MUST show $(\text{from} + (100 - \text{from}) \times n / \text{days})$ % of its daylight. Before `start`, and from $n = \text{days}$, it shows all of it.
2. The share multiplies daylight, lightning, and scenes, as `scale` does. It does not change moonlight.
3. The share changes once a day, at local midnight, so each day keeps the program's shape.
4. Owners use acclimation for a new light, new or moved corals, a program brighter than the last (Section 18.2), and the return of light after days without it.
5. A controller SHOULD offer an acclimation when a light is added or replaced.

For example, an acclimation from 2026-10-01 over 30 days from 50 % shows 50 % of daylight on 1 October, 70 % on 13 October, and all of it from 31 October.

12.3 Dimming on a Condition

A lighting function's `dim` lists caps that hold while a condition holds, each a `when` condition and a `maxLevel`.

1. While a `when` condition is true or unknown, every channel, and the intensity, of every light of the function MUST be at most its `maxLevel`. With several, the lowest applies.
2. An unknown condition counts as true, and raises `input-unavailable`: less light is the safe side when the reason to dim cannot be checked.
3. A controller MUST apply the start and end of a cap with a fade of 60 s (provisional), so that the lights do not jump.
4. For a light that runs its own program, the controller makes the cap from `dim`, or 100 when none holds, the cap of the light's program, as Section 16.4.5 says, so that it holds whatever the light shows, and stays after a scene's revert. The cap's end is not known, so it has no `revertAfter` [OAP-AU Section 17.5], and if the controller then fails, the cap stays until it returns.
5. While such a light shows a scene with levels or a color temperature (Section 13.2.5), the cap holds the scene too (Section 12.1.3). When the cap changes what the scene shows, the controller MUST send the scene again, with its levels, or its intensity, under the new cap, a `fadeTime` of 60 s, and the `revertAfter` that the scene's end needs, which keeps the pending revert's target [OAP-DC Section 7.4.6]. The controller MUST NOT send a `runProgram` without `revertAfter` to change the cap during a scene, because it cancels a pending revert, and so would end the scene [OAP-DC Section 7.4.7].
6. Owners use dimming when the water is too warm, when a light without its own thermal protection is too hot (Section 17.1.6), or while a hood is open.

12.4 Safety Checks

Table 18 — Checks on light settings

Check	Presence	Applies to	What it requires	Section
Caps over everything	REQUIRED	Controllers, lights	No channel shows more than <code>maxLevel</code> , or its kind's <code>maxLevels</code> entry, from daylight, moonlight, lightning, or a scene.	12.1
Acclimation	REQUIRED	Controllers, lights	Daylight, lightning, and scenes rise from the acclimation's <code>from</code> to all of the program over its <code>days</code> , changing at local midnight; moonlight does not change.	12.2
Dimming on a condition	REQUIRED	Controllers	While a <code>dim</code> condition is true or unknown, every light of the function is capped at its <code>maxLevel</code> , and the cap comes and goes over 60 s. A light that runs its own program takes it as the cap of its program, which stays after a scene's revert; a scene on such a light is sent again under the new cap, and does not end.	12.3

13 Scenes

13.1 Scenes

Table 19 — Members of a scene

Member	Type	Presence	Description
<code>id</code>	identifier	REQUIRED	Identifies the scene within the program.

Member	Type	Presence	Description
name	string	OPTIONAL	Name shown to the owner.
purpose	string	OPTIONAL	photo, viewing, maintenance, night, other, or an extension name.
levels	object	OPTIONAL, programs of curves only	The level of some kinds (Section 8.6.2).
colorTemperature, intensity	number	OPTIONAL, color temperature programs only	The scene's color temperature and intensity, both or neither.
fade	integer (s)	OPTIONAL	How long the scene takes to come and go, from 1 to 600 s. Default 10 s.
duration	integer (s)	OPTIONAL	How long the scene lasts, from 60 to 14400 s. Default 3600 s.

1. A scene of a program of curves gives `levels`, and a scene of a color temperature program gives `colorTemperature` and `intensity`. A channel whose kind, and its substitutes, are not in `levels` is off during the scene.
2. A scene with neither shows the program's own levels without effects, for owners who want the light to hold still.
3. Scene levels are scaled and acclimated like daylight, and capped (Section 12).
4. A controller's photo button starts the first scene whose `purpose` is `photo`, and likewise for other purposes.

13.2 Starting and Ending a Scene

1. The owner starts a scene with the controller, which needs the `start-mode` operation [OAP-SA Table 6]. It lasts its `duration`, or a shorter time the owner gives, or until the owner ends it, and is kept in the saved state (Section 7.7).
2. A controller MUST start a scene trigger's scene when its `when` condition becomes true, and MUST end it when the condition becomes false or unknown, or when the scene's `duration` has passed, whichever is first.
3. A lighting function shows one scene at a time: the one started last.
4. A controller MUST end every scene by the end of its `duration`, and no `duration` is longer than 4 hours.
5. For a light that runs its own program, the controller MUST send the scene so that it ends on the light even if the controller fails [OAP-AU Section 17.5]:
 - a scene with `levels`, with `set`, whose `levels` are the scene's levels after the light's settings (Section 14.1, step 7), whose `fadeTime` is the scene's fade, and whose `revertAfter` is the time until the scene's fade-out begins;
 - a scene of item 13.1.2, with `runProgram`, with `effects false`, and the same `fadeTime` and `revertAfter` (Section 16.4); and
 - a scene of a color temperature program, with `setColorTemperature` and the same `fadeTime` and `revertAfter` to a light with the `color-temperature` feature, and otherwise with `set` and the levels of the light's mixes (Section 8.5).

When the revert falls due, a light of Device Classes revision 0.3 or later fades back to its program over the fade of the latest command with `revertAfter` [OAP-DC Section 7.4.12]: the scene's fade, and so reaches it at the scene's end, as Section 13.4.2 requires, or 60 s if the controller sent the scene again for a cap (Section 12.3.5).

6. When a scene's fade-out begins, or the scene ends before its time, the controller MUST send `runProgram` with a `fadeTime` of the scene's fade (Section 16.4), whether or not the revert falls due at the same moment. The light then fades back to its program over the scene's fade, as Section 13.4.2 requires, even after the controller sent the scene again for a cap; it shows its program's effects again, which a light of Device Classes revision 0.4 or earlier does not restore when a revert falls due [OAP-DC Table 4]; and a light of revision 0.3 or earlier takes back its cap (Section 16.4.5). When a scene ends before its time, its fade-out begins then (Section 13.4.2).

7. A light of Device Classes revision 0.1 or 0.2 returns to its program at once when a revert falls due [OAP-DC Table 4]. For such a light, the controller sends the scene with `revertAfter` the time until the scene's end, so that the light jumps back to its program only if its controller has failed. Such a light also cancels a pending revert with every `runProgram` [OAP-DC Table 4], so, before it sends `runProgram` with `revertAfter` while a revert is pending, the controller sends `runProgram` without `revertAfter`, and the new revert's target is the program. A light of revision 0.1 has no `revertAfter` for `setColorTemperature`, and no `runProgram` (Section 16.4.4). The controller sends such a light a scene of a color temperature program as `set` with the levels of the light's mixes, or, for a light without mixes, with `setColorTemperature`, which then ends on the light only when the controller ends it.

13.3 Photo Mode

Photo mode is a scene whose `purpose` is `photo`.

1. Controllers SHOULD let the owner start photo mode with one action, from every app they offer.
2. Clouds, storms, and lightning do not show during photo mode once its fade-in has ended, as during every scene, so that photographs are evenly lit.
3. A photo scene SHOULD balance whites and blues so that colors photograph as the eye sees them, and SHOULD last 30 minutes (provisional).
4. A scene whose `purpose` is `night`, such as a dim light for watching fish at night, SHOULD fade in over at least 60 s, because it starts from darkness.

13.4 Fades

A scene's fade, f , is its `fade`, or half its `duration` if that is shorter.

1. A scene fades in evenly over f , from what the light shows when the scene starts to the scene's level. During the fade-in, a channel's level MUST be $L_0 + (S - L_0) \times x / f$, where L_0 is what the light showed when the scene started, S the scene's level now (Section 14.1, step 7), and x the seconds since the scene started. Unless another scene was shown, L_0 is the program's level, its clouds and storms included (Section 14.1, step 6).
2. A scene fades out evenly over the last f of its `duration`, or, when it ends before its time, over the f after it ends (Section 13.2.6), from what the light shows back to what the program then shows. During the fade-out, a channel's level MUST be $S_1 + (P - S_1) \times y / f$, where S_1 is what the light showed when the fade-out began, P the program's level now, its clouds and storms included, and y the seconds since the fade-out began. So a scene that starts or ends during a storm starts from, and returns to, the storm's light, without a jump. A light that runs its own program fades in the same way when it is sent the scene with a `fadeTime`, and when its revert falls due [OAP-DC Sections 7.4.10 and 7.4.12].
3. A light MUST return to its program with the restore fade, from what it shows to the program's level, evenly over `restoreFade` seconds:
 - after power-up, from 0 % (Section 17.4.1);
 - when a higher layer's stop or run ends (Section 7.6.3);
 - when its program is replaced; and
 - when its clock becomes valid again (Section 17.4.2).

During the fade, a channel's level is $L_0 + (P - L_0) \times x / \text{restoreFade}$, where L_0 is what it showed when the fade began, P the program's level now, without lightning (Section 11.1.3), and x the seconds since the fade began. A light the controller drives fades up by itself after power-up, until its controller resumes control and continues the restore fade from what the light shows (Section 17.4.1). A `switch` output cannot fade: it switches on when the level of its fade passes its `threshold` (Section 14.3).

4. A controller that drives a light SHOULD give every command that changes a channel by more than 10 percentage points a `fadeTime` of at least 2 s, except a stop, which it MUST NOT delay [OAP-AU Section 17.4.4].
5. A curve SHOULD NOT rise from darkness to bright light in less than 20 minutes (provisional): a validator warns about a rise from below 5 % to above 50 % in less time.

6. A lighting function's `restoreFade` SHOULD be at least 60 s (provisional), and a validator warns about a shorter one: a shorter fade brings a light back from darkness to its program almost as suddenly as a switch.

Fish are easily startled when light comes on at night, and some jump; laboratory guidance asks for 20 to 30 minutes of brightening and dimming [CCAC].

13.5 Safety Checks

Table 20 — Checks on scenes

Check	Presence	Applies to	What it requires	Section
Scenes end by themselves	REQUIRED	Controllers, lights	A scene lasts at most its <code>duration</code> , and reaches a light that runs its own program with <code>revertAfter</code> , so that it ends even if the controller fails, with a fade on a light of Device Classes revision 0.3 or later; a controller that keeps running sends <code>runProgram</code> when the fade-out begins.	13.2
Scenes from conditions	REQUIRED	Controllers	A scene trigger starts its scene when its condition becomes true, and ends it when the condition ends or the scene's <code>duration</code> has passed.	13.2
Fades and the restore fade	REQUIRED	Controllers, lights	Scenes fade in from what the light shows, and fade out back to what the program shows, clouds and storms included; lights that dim return to their program with the restore fade after power-up, after a higher layer's stop, and after a new program, and, from Device Classes revision 0.4, fade up to a fail-safe state that lights them.	13.4
Gentle restore fades	RECOMMENDED	Configurations	A lighting function's <code>restoreFade</code> lasts at least 60 s, so that a light does not come back from darkness as suddenly as a switch; a validator warns about a shorter one.	13.4

14 Computing Levels

14.1 Programs of Curves

For a light at a moment t , with its program time $u = t - \text{offset}$ and its effect time $v = t - \text{cloudOffset}$ (Section 15.2), a product MUST find each channel's level in these steps:

- Daylight:** the level at u of the curve the channel follows (Sections 8.3, 8.6, and 8.7), or 0 if it follows none, times the seasonal share of a table on the local date of u (Section 9.5.5).
- Clouds and storms:** multiplied by the share each cloud and storm at v leaves, for the kinds they dim (Sections 11.3 and 11.4).
- Lightning:** during a flash at v , for a channel of a lightning kind, the higher of this and `lightning.level`, if the light's `lightning` is `true`, no scene is shown, from the start of its fade-in to the end of its fade-out, and no restore fade is in progress (Sections 11.1.3 and 11.5).
- Scale and acclimation:** multiplied by `scale` / 100 and by the acclimation's share on the local date of u (Sections 12.1 and 12.2).
- Moonlight:** for a channel that carries moonlight, the higher of this and the moonlight at u (Section 10).
- Caps:** the lowest of this, `maxLevel`, the `maxLevels` entry for the channel's kind, and any cap from `dim` (Sections 12.1 and 12.3). This is the program's level.
- Scene:** while a scene is shown, the level of Section 13.4.1 during its fade-in, of Section 13.4.2 during its fade-out, and otherwise S , the scene's level. S is the scene's level for the channel (Section 13.1), times `scale` / 100 and the acclimation's share, at most the caps of step 6; for a scene of Section 13.1.2, it is the result of steps 1, 4, 5, and 6.
- Restore fade:** during a restore fade, the level of Section 13.4.3, with P the result of step 7.

The light then applies its own thermal limit (Section 17.1).

Caps come before the scene so that a scene fades between levels a light can show, as a light that runs its own program does when it is sent a scene with `set` and a `fadeTime` (Section 13.2.5).

14.2 Color Temperature Programs

For a color temperature program, a product MUST find the color temperature and intensity in these steps:

1. **Daylight:** the color temperature K and intensity I at u (Section 8.4), with I times the seasonal share of a table (Section 9.5.5).
2. **Clouds and storms:** I is multiplied by the share each cloud and storm at v leaves.
3. **Lightning:** during a flash at v , I is the higher of I and `lightning.level`, if the light's `lightning` is `true`, no scene is shown, from the start of its fade-in to the end of its fade-out, and no restore fade is in progress.
4. **Scale and acclimation:** I is multiplied by `scale` / 100 and the acclimation's share.
5. **Moonlight:** if the moonlight's intensity at u is higher than I , K and I are the moonlight's (Section 10.3.4).
6. **Caps:** I is at most `maxLevel`, `maxLevels.intensity`, and any cap from `dim`. This is the program's color temperature and intensity.
7. **Scene:** while a scene is shown, K and I fade as Sections 13.4.1 and 13.4.2 say, K evenly in mireds and I evenly, to and from the scene's `colorTemperature` and its `intensity` times `scale` / 100 and the acclimation's share, at most the caps of step 6, or, for a scene of Section 13.1.2, the result of steps 1 and 4 to 6.
8. **To the light:** a light with the `color-temperature` feature is sent K , kept within its `minColorTemperature` and `maxColorTemperature`, and I . Any other light shows each channel's level from its mixes at K , its light member's or else the program's, times I / 100 (Section 8.5), at most `maxLevel` and the `maxLevels` entry for its kind.
9. **Restore fade:** as in Section 14.1, step 8.

14.3 Level Outputs and Switches

1. A `level-output` output is one channel of its light member's `kind`, and its level is found as in Section 14.1, or for a color temperature program from the mixes as in Section 14.2.
2. A `switch` output is one channel of its light member's `kind`, whose level is found in the same way without clouds, storms, and lightning. It is on while that level is above its `threshold`, and off otherwise. Effects are left out so that a cloud never switches a light off and on.

14.4 Driving Lights from the Controller

1. For a light it drives, a controller MUST send commands that keep each channel within 1 percentage point of the level of Section 14.1 or 14.2, or of a level the program gives within the last 2 s.
2. It SHOULD send each light the level of a coming moment, with a `fadeTime` until then, for example the next point of a curve, or 60 s ahead, so that the light fades smoothly with few commands. A fade changes each level evenly over its `fadeTime`, and, on a light of Device Classes revision 0.4 or later, the color temperature of `setColorTemperature` evenly in mireds, as Section 14.2 does [OAP-DC Section 7.4.10].
3. It uses `set` with `levels` for a `light`, `setColorTemperature` for a light with the `color-temperature` feature that shows a color temperature program, `set` with `level` for a `level-output`, and `set` with `on` for a `switch`.
4. It SHOULD NOT send a light more than one command a second (provisional), except a stop, which it never delays, as Automation Section 17.4.4 recommends for repeated commands. It sends no command only to repeat a state [OAP-AU Section 17.4.3].
5. It sends nothing to a light under local control (`controlSource local`), except the stops that protect the system, which Automation Section 17.4.2 lists: an interlock's, `stop-all`'s, and those of the emergency and power-loss modes.

15 Groups

15.1 Lights That Run Together

1. The lights of a lighting function are a group: they run one program, each with its own settings (Table 17).
2. A group's lights need not come from one maker, share a binding, or have the same channels. Some may run the program themselves while the controller drives others.

3. Lights of a group never need to exchange messages: each computes the program from the same members, the same seed, and the same clock.
4. A tank with lights that should run different programs, such as a display and its refugium, has a lighting function for each.

15.2 Staggering

1. A light's `offset` delays its curves and moonlight: it MUST show at t what the program gives at $t - \text{offset}$. Its acclimation follows the local date of $t - \text{offset}$.
2. A light's `cloudOffset` delays its clouds, storms, and lightning in the same way, or, when negative, brings them forward.
3. Scenes, caps, and restore fades follow the moment itself, so a group enters and leaves a scene together.
4. Owners stagger lights to make the sun rise across a tank, with an `offset` of a few minutes for each light from east to west, or to brighten a fish room rack by rack.
5. Increasing `cloudOffset` by 10 to 30 s from one light to the next makes each cloud pass from light to light, as a cloud's shadow crosses a reef.

15.3 Clocks

1. A light that runs its own program takes the time from its controller through its binding: the module bus's time synchronization [OAP-MB Section 9.4], or the time that the Device Protocol gives in its supervision messages and its time exchange [OAP-DP Section 12.3].
2. It SHOULD keep its clock within 0.5 s of its controller's (provisional), so that the lights of a group stay together.

15.4 Safety Checks

Table 21 — Checks on groups

Check	Presence	Applies to	What it requires	Section
Lights in step	REQUIRED	Controllers, lights	The lights of a group show the same program at their offsets, without messages between them.	15.2
Clocks together	RECOMMENDED	Lights	A light that runs its own program keeps its clock within 0.5 s of its controller's.	15.3

16 Lights That Run Their Own Programs

16.1 Fixture Programs

1. A light with the `local-program` feature runs the fixture program in its `program` attribute [OAP-DC Section 9.4], written as JSON text [RFC 8259] without whitespace between tokens, or a reference to it (Section 16.2). An empty `program` means that the light follows `set` commands.
2. A fixture program has the members of Table 6 that say what the program does (`method`, `seed`, `sun`, `curves` or `colorTemperature`, `moon`, `clouds`, `storms`, `scenes`, `fixture`, and `extensions`), and MAY have its `id` and `name`; the members of Table 17 that apply to the light (`offset`, `cloudOffset`, `scale`, `maxLevel`, `maxLevels`, `channels`, and `lightning`); and the members of Table 22. A light member's `mixes` take the place of the program's in `colorTemperature` (Section 8.5.5). It MUST be valid against `$defs/fixtureProgram` in `schema/lighting-programs.schema.json`.
3. A fixture program never has the sun source `configuration`: the controller writes the configuration's astronomy into it, as a `location` or a `table`, with its `shift`, `noonAt`, and `shiftDays`.
4. A light MUST reject a write of a program it cannot run in full, such as one whose `method` it does not implement (Section 9.1.4), and keep its previous program [OAP-DC Section 7.13.2].
5. There is one conformance level: a conforming light that runs its own programs runs every fixture program that the schema allows, up to 65536 bytes (Section 21.2.1; Appendix D, issue 14). Parsed into numbers, the largest takes less than 16 kB, which a light can keep in its flash memory rather than its working memory. A

light that cannot run such programs does not conform as a light that runs its own programs, and its controller drives it, as it does any light that rejects the program written to it (Section 16.2.6).

Table 22 — Further members of a fixture program

Member	Type	Presence	Description
<code>posixTimeZone</code>	string	REQUIRED	The local time zone, as a POSIX TZ string [POSIX], such as CET-1CEST,M3.5.0,M10.5.0/3, as in Automation.
<code>acclimation</code>	object	OPTIONAL	The lighting function's acclimation (Section 12.2).
<code>restoreFade</code>	integer (s)	OPTIONAL	The lighting function's restore fade (Section 13.4). Default 900 s.

16.2 Writing a Fixture Program

- For a light whose `runOn` is `light`, the controller MUST write its fixture program within 60 s (provisional) of its differing from what the light reports. For a light whose `runOn` is `controller` and that has the `local-program` feature, it writes an empty `program`.
- When the text fits in one value of the light's binding, the controller writes the text. On the module bus, a value holds 256 bytes [OAP-MB Section 11.6.1]; over the Device Protocol, the text fits when the message that writes it is no longer than the 8192 bytes every device accepts [OAP-DP Section 9.3].
- Otherwise, it writes a reference: the JSON text of an object with `sha256`, the SHA-256 hash [FIPS 180-4] of the text, in lowercase hexadecimal, and `size`, its length in bytes, at most 65536 (provisional). It serves the text as a file at the path `oap/lighting/` followed by the hash and `.json`, for as long as the light holds the reference: on the module bus with `uavcan.file.Read.1.1` [OAP-MB Section 11.8], and over the Device Protocol with `file.read`, the write of the reference offering the file to the light [OAP-DP Section 13.2.2].
- A light that is written a reference MUST read the file from its controller, and check its size and hash, and the program it holds, before it runs it. Until it has, it keeps its previous program, and reports its previous `program`; if a check fails, it keeps them.
- A light keeps its fixture program across power loss [OAP-DC Section 7.3.3], so that after power-up it runs it without its controller.
- The condition `program-not-loaded` holds while a light whose `runOn` is `light` does not report the fixture program its controller wrote to it. Its alarm is raised once it has lasted its delay, 60 s (provisional), which `conditions.json` holds (Section 7.8.3). Once the condition has lasted its delay, the controller MUST drive the light itself, as for `runOn controller`, until the light reports the program. Meanwhile the light runs its previous program, so a program that takes a few seconds to load does not take the light off its program.

A fixture program of curves with sun, moon, clouds, a storm, and three scenes takes about 2000 bytes, so on the module bus it is written as a reference, and over the Device Protocol as text. `examples/light-network.json` and `examples/light-module.json` show both.

16.3 Running a Fixture Program

- A light with a fixture program follows Sections 8 to 15 by itself, with its own clock, and MUST meet Section 7.10 against the reference implementation.
- With `failSafeAction continue`, a light that was following its program when it lost contact with its controller MUST keep running it, with the cap of its program (Section 16.4.5), and with any pending revert, which still falls due [OAP-FS Section 7.3]. With `failSafeAction state`, it enters `failSafeState` when it loses contact, as Fail-Safe Behavior requires, and, from Device Classes revision 0.4, fades to a `failSafeState` that lights it from what it shows [OAP-DC Section 8.3.6]. A light that a latched stop holds keeps it in place of either [OAP-FS Section 8.4].
- It reports `programRunning` [OAP-DC Section 9.4]: `true` while it shows its program, and `false` while it shows levels from `set` or `setColorTemperature`, a stop, or its fail-safe state.
- Without a valid clock, it runs no program (Section 17.4).
- A light cannot read Automation inputs, so scenes, caps from `dim`, and holds reach it through its controller (Section 16.4).

16.4 Under a Controller

1. A controller returns a light to its program with `runProgram` [OAP-DC Section 9.4], whose parameters are all OPTIONAL:

- `fadeTime` (number, s): the fade from what the light shows to its program; default the fixture program's `restoreFade`;
- `maxLevel` (number, %): the cap of the light's program, on every channel and the intensity (item 5);
- `effects` (boolean): `false` stops the program's effects until the light leaves its program, or receives another `runProgram`; default `true`; and
- `revertAfter` (number, s): as for `set` [OAP-DC Section 7.4].

The light cancels any levels from `set` or `setColorTemperature`, and shows its program. Without `revertAfter`, it also cancels any pending revert; with `revertAfter`, a pending revert keeps its target, and only its time changes [OAP-DC Section 7.4.6]. The revert target of a light that was following its program is its program. From Device Classes revision 0.5, a light that returns to its program when a revert falls due shows its program's effects again [OAP-DC Section 7.13.8]; a light of an earlier revision keeps them as they were, so the controller restores them with `runProgram` (Section 13.2.6).

2. The controller sends it:
 - to resume control of the light, for example after contact returns [OAP-FS Section 7.4], with the parameters that keep what it shows, such as its current cap [OAP-DC Section 7.13.5];
 - when a higher layer's stop or run of the light ends, with `fadeTime` the restore fade (Section 7.6.3);
 - when a scene's fade-out begins, or it ends before its time, with `fadeTime` the scene's fade (Section 13.2.6);
 - for a scene that shows the program without effects, with `effects false` and `revertAfter` (Section 13.2.5); and
 - whenever the cap from `dim` changes while the light shows its program, as item 5 says.
3. A controller MUST NOT let a light run its own program unless the light implements `runProgram`, as every light of Device Classes revision 0.2 or later with the `local-program` feature does; it drives any other light itself. Without the command, a controller could not return the light to its program after a stop with no known end, such as a power-loss mode [OAP-AU Section 17.3.4].
4. A light of Device Classes revision 0.1 has neither `runProgram` nor `programRunning`. Revision 0.1 of this Specification carried them as the extension command `org.openaquariumproject.run-program`, with the parameters of item 1, which cancels any pending revert, and as `org.openaquariumproject.program-running` in the endpoint's `extensions` [OAP-DC Table 5]. A controller MAY let such a light run its own program if the light reports `org.openaquariumproject.program-running`, and then uses these names in place of `runProgram` and `programRunning`. A light of revision 0.2 or later SHOULD NOT use them.
5. The cap of a light's program is the cap from `dim`, or 100 when none holds (Section 12.3.4). A controller MUST set it as this item says, so that it holds whatever the light shows:
 - while the light shows its program, with `runProgram`, with the cap as `maxLevel` and a `fadeTime` of 60 s, so that the light fades to it, and, during a scene that shows the program without effects, with that scene's `effects false` and `revertAfter`, which keeps the pending revert's target;
 - otherwise, to a light of Device Classes revision 0.4 or later, by writing `programMaxLevel`, which changes nothing the light shows until it returns to its program, applies then, and stays when a revert falls due [OAP-DC Section 7.13.7]; and
 - otherwise, to a light of an earlier revision, whose cap only `runProgram` sets, with the first `runProgram` that returns the light to its program.

A light of Device Classes revision 0.3 or earlier takes a `runProgram` without `maxLevel` as a cap of 100 %, and a revert to its program restores the cap it had before, so every `runProgram` that a controller sends such a light MUST carry the cap in force as `maxLevel`. To a light of revision 0.4 or later, a `runProgram` without `maxLevel` leaves the cap as it is, and the controller leaves out `maxLevel` when it shows the program only for a while, as for a scene that shows the program without effects [OAP-DC Section 7.13.7]. If the controller

fails during a scene on a light of revision 0.3 or earlier, the light returns to its program with the cap it had before the scene.

- The controller watches `programRunning` and the light's `levels`, and shows the owner what the light shows (Section 7.9).

16.5 Safety Checks

Table 23 — Checks on lights that run their own programs

Check	Presence	Applies to	What it requires	Section
The same program on a light	REQUIRED	Lights	A light that runs a fixture program shows what the reference implementation computes, within Section 7.10.	16.3
Writing fixture programs	REQUIRED	Controllers	A controller writes each fixture program within 60 s of its differing from what the light reports: as text when it fits the binding, and otherwise as a reference to a file it serves.	16.2
Checking a program file	REQUIRED	Controllers, lights	A light checks a program file's size and hash before it runs it, and keeps its previous program if a check fails; the controller then raises <code>program-not-loaded</code> and drives the light.	16.2
Returning to the program	REQUIRED	Controllers, lights	A controller returns a light that runs its own program to it with <code>runProgram</code> , which cancels levels from <code>set</code> , and a pending revert unless it carries <code>revertAfter</code> ; a light without <code>runProgram</code> does not run its own program.	16.4
Running without a controller	REQUIRED	Lights	A light with <code>failSafeAction continue</code> keeps running its program, with the cap of its program, when it loses its controller.	16.3

17 Heat and Failures

17.1 Fixture Temperature

- A light with the `fixture-temperature` feature MUST apply its thermal limits to every level it shows, from any source: its own program, a scene, lightning, `set`, `setColorTemperature`, `runProgram`, its fail-safe state, and its power-up state.
- While it dims for heat, it MUST report `thermalDimming true`, and `overtemperature` in its endpoint's `faults` [OAP-DC Section 9.4], so that an alarm is raised (Section 7.8.2). It MAY report how far it has dimmed as `thermalScale`.
- When it has cooled, it MUST return to the levels it should show over at least 60 s (provisional), not at once.
- A controller MUST NOT raise a light's levels to make up for thermal dimming, MUST NOT treat the difference as a refused command or a fault, and MUST NOT send commands only because the levels the light reports differ from those it was sent while `thermalDimming` is `true`.
- A controller MUST show the owner that a light is dimmed for heat (Section 7.9).
- A light without thermal protection of its own, such as LEDs dimmed by `level-output` endpoints, can be protected by a temperature sensor on its heat sink and a `dim` rule (Section 12.3). That is a second layer, not a substitute for thermal protection in the light's design.

17.2 Fail-Safe Configuration

Table 24 gives the RECOMMENDED fail-safe configuration [OAP-FS Section 13] of lights. It follows the Fail-Safe Behavior recommendation for the category `light` [OAP-FS Table 3].

Table 24 — Recommended fail-safe configuration of lights

Light	On loss of contact	After power-up	On battery	Notes
Running its own program	<code>continue</code>	<code>last</code>	<code>stop</code>	It keeps its program, with its moon, effects, acclimation, and cap. A scene ends when its revert falls due, with a fade on a light of Device Classes revision 0.3 or later. After power-up, it runs its program, beginning with the restore fade from 0 %, if it was following it.

Light	On loss of contact	After power-up	On battery	Notes
Driven by the controller	continue	last	stop	It keeps its last levels until its controller returns: midday light through the night, or darkness through the day. Neither is what the program wants, and the controller's heartbeat tells the owner, so that someone can step in. After power-up, it fades up to its last levels over 900 s (provisional), and its controller then continues the restore fade (Section 17.4.1). An owner who prefers a fixed light, or darkness, uses <code>state: from Device Classes revision 0.4</code> , the light fades to a <code>failSafeState</code> that lights it over 900 s, and reaches darkness at once.

1. A controller MUST send scenes, and every other temporary state of a light, with `revertAfter` [OAP-AU Section 17.5], so that a light it leaves behind does not stay in a scene, except `setColorTemperature` to a light of Device Classes revision 0.1, which has no `revertAfter` (Section 13.2.7).
2. "On battery" is the Automation power-loss mode, which stops lights [OAP-AU Table 29].
3. From Device Classes revision 0.4, a light whose `failSafeState` lights it fades to it, from 0 % after power-up, beginning when its `powerOnDelay` ends, and from what it shows when it enters it at any other time, such as on loss of contact, over 900 s (provisional) [OAP-DC Sections 8.3.5 and 8.3.6]. A light of an earlier revision enters it at once.

17.3 Lost Inputs

Lighting defines what it does when an input is unavailable, as Fail-Safe Behavior Section 10.3 requires.

Table 25 — What lighting does when an input is unavailable

Input	Behavior	Section
The controller's clock	Lights the controller drives take their fail-safe behavior; lights that run their own programs keep to their own clocks. <code>input-unavailable</code> is raised.	17.4
A light's clock	The light runs no program; its channels are off, and it reports <code>clock-invalid</code> .	17.4
A <code>dim</code> condition	The cap applies.	12.3
A scene trigger's condition	The scene ends.	13.2
A light's device	Its endpoints take their fail-safe behavior; the controller shows the light as lost.	17.5
A fixture program's file	The light keeps its previous program; <code>program-not-loaded</code> is raised, and the controller drives the light.	16.2

17.4 Power, Restarts, and the Clock

1. After power-up, a light follows its `powerOnBehavior` [OAP-FS Section 9.1]. A light that restores its program or its levels MUST fade up to them from 0 %, beginning when its `powerOnDelay` ends, as Device Classes Section 8.3.3 requires, and, from Device Classes revision 0.4, so MUST a light that enters a `failSafeState` that lights it [OAP-DC Section 8.3.5]:
 - a light that runs its own program, and either was following it when it lost power, with `powerOnBehavior last`, or has `powerOnBehavior fail-safe` and `failSafeAction continue`, runs it once its clock is valid, beginning with the restore fade from 0 % (Section 13.4.3);
 - a light whose `powerOnBehavior` is `last` and that restores levels, such as one its controller drives, fades up to them over 900 s (provisional), the default restore fade, because it does not know its function's `restoreFade` [OAP-DC Section 8.3.3]; and
 - a light whose `powerOnBehavior` is `fail-safe` and whose `failSafeState` lights it fades up to that state over 900 s (provisional) [OAP-DC Section 8.3.5].

A light that a latched stop held when it lost power stays off [OAP-FS Section 8.4]. Once the controller resumes control of a light it drives, it continues with the restore fade from what the light shows (Section 13.4.3). So no light that dims jumps from darkness to midday levels after a power cut, unless its `restoreFade` is very short (Section 13.4.6); a `switch` output can only switch on.

2. A light whose clock is not valid MUST run no program, and MUST keep its channels off, until it has the time from a battery-backed clock or from its controller, which its binding gives it within seconds of contact

(Section 15.3). It reports `clock-invalid` meanwhile [OAP-FS Section 10.5.6]. A program that cannot know the time of day cannot know whether it is day.

3. After a restart, a controller restores its saved state, ends scenes whose end has passed, and continues the rest [OAP-AU Section 17.7]; lights it drives begin with the restore fade from what they report.
4. While the controller's clock is not valid, its lighting functions' demand on the lights it drives MUST be their fail-safe behavior [OAP-FS Section 10.3.2], and it raises `input-unavailable`. Lights that run their own programs keep to their own clocks, which stay valid (Section 7.5.6).
5. When mains power returns after a power-loss mode, the lights return to their programs with the restore fade, as after power-up [OAP-FS Section 9.5.4].
6. A light SHOULD keep its levels while it updates its firmware [OAP-FS Section 11.2], and returns to its program with the restore fade afterwards.

17.5 A Failed Controller

When the controller fails:

1. lights that run their own programs, with `failSafeAction continue`, keep their programs, with their moon, effects, and acclimation, and keep the cap of their program, the last cap from `dim` (Section 16.4.5);
2. a scene ends on each light when its revert falls due, fading back to the program on a light of Device Classes revision 0.3 or later (Section 13.2.5);
3. lights the controller drove keep their last levels, or, with `failSafeAction state`, take their fail-safe state, fading to it from Device Classes revision 0.4 (Section 17.2);
4. a light that a latched stop holds stays off [OAP-FS Section 8.4]; and
5. the controller cannot raise an alarm, so the alarm outputs and heartbeats of the Alarms and Notifications specification tell the owner [OAP-AN Section 18].

17.6 Safety Checks

Table 26 — Checks on heat and failures

Check	Presence	Applies to	What it requires	Section
Thermal limits over everything	REQUIRED	Controllers, lights	A light's thermal limits apply to every level it shows, from any source, and a controller never makes up for them.	17.1
No clock, no program	REQUIRED	Lights	A light whose clock is not valid runs no program, and returns to its program with the restore fade once it has the time.	17.4
A controller without a clock	REQUIRED	Controllers	While a controller's clock is not valid, lights it drives take their fail-safe behavior, and lights that run their own programs keep them.	17.4
Temporary states end on the light	REQUIRED	Controllers	Every scene, and every other temporary state of a light, reaches the light with <code>revertAfter</code> .	17.2

18 Sharing

18.1 Program Documents

1. A program document MUST be a JSON text [RFC 8259] encoded in UTF-8, whose top-level value is an object with the members of Table 27, valid against `schema/lighting-programs.schema.json`, and meeting every requirement of Sections 8 to 13 for each of its programs.
2. Each program's `id` is unique within the document.
3. A program document describes light, not a tank: it carries no bindings, saved state, or settings of particular lights. Those belong to the configuration that runs it (Section 7.2).
4. Program documents are exchanged as files or messages of the media type `application/json`.

Table 27 — Members of a program document

Member	Type	Presence	Description
<code>oapLightingPrograms</code>	string	REQUIRED	Revision of this Specification. For this revision, "0.3"; a product also reads documents of revisions 0.1 and 0.2, whose members are the same.
<code>generator</code>	object	OPTIONAL	The product that wrote the document, as a Data Model Generator [OAP-DM Table 2].
<code>created</code>	timestamp	OPTIONAL	When the document was written.
<code>programs</code>	array	REQUIRED	The programs, at least one (Section 8).
<code>extensions</code>	object	OPTIONAL	Extensions.

18.2 Importing and Exporting

1. A controller **MUST** be able to export any program it runs as a program document, and **MUST** be able to import one, without a network service. A light that offers programs of its own, for example through its maker's app, **SHOULD** be able to import and export them as program documents.
2. A product **MUST** validate a program document before it uses it, and **MUST** reject it, without changing what runs, if it is not valid [OAP-AU Section 19.2].
3. Before it runs an imported program on a light, a controller **MUST** show the owner which curve each channel follows, and which channels stay off (Section 8.6), and where the program's sun comes from. A program whose sun source is `configuration` needs the configuration's `astronomy`; the controller **MUST** ask the owner for one if there is none.
4. When a program replaces another on a light, the controller **SHOULD** compare their daily light: for a date near today, the sum over the light's channels of each channel's level, after the light's settings, over the day, in percent-hours; or, where every channel of the light gives its `spectrum` and `photonFlux`, the light's photosynthetic photons over the day, in mol (Section 19.3). If the new program gives more than 20 % (provisional) more light than the old, the controller **SHOULD** propose an acclimation from the old program's share of the new, over 30 days (provisional) (Section 12.2).
5. The owner **MUST** confirm an imported program before it runs [OAP-AU Section 19.2.5].

Corals take from a few days to more than three weeks to adjust to more light [DiPerna], and a program written for another light can give much more light than it seems. Item 4 makes that visible, and the acclimation brings the new light in gradually.

18.3 Licenses, Credit, and Privacy

1. A program shared with others **SHOULD** give its `license`, an SPDX license expression [SPDX], such as `CC0-1.0`, which needs no credit, or `CC-BY-4.0`, which needs credit to the author.
2. A product that imports a program **MUST** keep its `author` and `license`, show them to the owner, and keep them when it shares the program again.
3. A `location` can be the owner's own home. Before it shares a program, a controller **SHOULD** offer to replace a `location` that is the configuration's own with the sun source `configuration`, and to remove the author's name and the description [OAP-AU Section 21].
4. A program document carries nothing about the owner except what the owner puts in it.

19 History

1. Light measured by a PAR sensor or a PAR meter is recorded as Data Model observations of `ppfd` [OAP-DM Section 9.2], with the measurement position, such as "sand bed, centre", in `notes`.
2. A controller with a PAR sensor **MAY** record each local date's daily light integral as an observation of the Data Model parameter `daily-light-integral`, in `mol/m2/d`, moles of photons per square metre per day, whose `time` is the start of the date [OAP-DM Section 9.2]. It is the sum, over the date, of each reading's PPFD times the seconds until the next reading, divided by 1 000 000, from readings no more than 60 s apart; a date with a longer gap has no value. Revision 0.1 recorded it as the extension parameter

`org.openaquariumproject.daily-light-integral`, in the extension unit `org.openaquariumproject.mol-per-m2-day`, which a consumer reads as `daily-light-integral` in `mol/m2/d`.

- Where every channel of a light gives its `spectrum` and `photonFlux` [OAP-DC Section 7.15], a controller SHOULD show the light's photosynthetic photon flux at the levels it shows, as Device Classes Section 7.15.4 estimates it, and its photosynthetic photons over each local date, in `mol`: the sum, over the date, of that flux times the seconds it lasts, divided by 1 000 000. Both estimate the light that leaves the light, not what reaches the livestock, and the controller shows them as estimates.
- A controller MAY estimate a vessel's daily light integral from the photosynthetic photons of the lights whose equipment is in it, over a local date, divided by the area of its water's surface, its `length` times its `width` [OAP-DM Section 8.2], in m^2 . That is more than the livestock receive, because some light misses the water and the water absorbs some, so the controller shows it as an estimate, and records no observation of it, which item 2 keeps for measured light.
- The program a lighting function runs is part of the configuration, which the controller exports [OAP-AU Section 19.1], and changes to it are audited [OAP-SA Section 12].
- Thermal dimming is recorded through the history of the `overtemperature` alarm [OAP-AN].

This answers Data Model issue 5 [OAP-DM Appendix C]: the Data Model records light as measured, programs are configuration, and spectra belong to the light's maker, who publishes them, from which a controller estimates light in photons (Appendix D, issue 1).

20 Recommended Values

Table 28 gives the RECOMMENDED values, and the defaults of this Specification, that do not depend on the system. They are provisional, and are published in machine-readable form in `checks.json`. Values that depend on the system, such as levels, scales, caps, and the photoperiod, are set by the owner and the program's author.

Table 28 — Recommended values

Area	Value	Recommended	Why
Programs	Level agreement	0.5 %	Half a percentage point is less than the eye or a coral can tell apart, and allows for differences in arithmetic between products (Section 7.10).
Programs	Brighter program	20 %	Corals take from a few days to more than three weeks to adjust to more light, so a large increase is brought in gradually [DiPerna]. The threshold itself is a starting point.
Programs	Acclimation for a brighter program	30 days	Longer than the slowest coral adjustment measured, over 20 days [DiPerna], and the period one light maker recommends for a new light.
Sun and moon	Sun tolerance	2 s	The method gives the same result to within milliseconds on any product that computes in <code>binary64</code> ; 2 s allows for differences in arithmetic (Section 9.6).
Moonlight	Moonlight level	5 %	On a channel made for moonlight. Even this is far brighter than the moon on most lights: a full moon gives 0.05 to 0.3 lux [Kyba], and coral research systems use 0.3 to 1 lux [Lin] [Koukoumaftsis].
Moonlight	<code>fade</code>	1800 s	Half an hour, as dusk deepens into night.
Moonlight	Moon fade altitude	10°	The moon climbs 10° in about 40 minutes near the equator, a fade like its brightening as it clears the horizon.
Effects	Shortest cloud	120 s	One light maker's clouds last 3 to 6 minutes, and come every 10 minutes; 2 to 10 minutes covers that, with variety.
Effects	Longest cloud	600 s	As for the shortest cloud.
Effects	Cloud fade	30 s	A cloud's shadow comes and goes in seconds to a minute; half a minute does not startle fish.
Effects	Shortest storm	900 s	One commercial controller allows storms of 1 to 60 minutes; 15 to 45 minutes is a passing afternoon storm.
Effects	Longest storm	2700 s	As for the shortest storm.
Effects	Storm fade	600 s	A storm darkens the sky over minutes, not seconds.

Area	Value	Recommended	Why
Effects	Shortest effect fade	10 s	A cloud or storm that comes and goes in less than 10 s changes daylight about as quickly as a light switched on, which startles fish; real cloud shadows come and go over seconds to a minute (Sections 11.3 and 11.4).
Effects	Lightning rate	4 a minute	No maker publishes a rate, and no study of lightning effects on fish was found. Rare bursts look more natural and startle less. Provisional.
Effects	Flashes in a second	3	The limit of WCAG 2.2 and ITU-R BT.1702-3, which protect people with photosensitive epilepsy [WCAG] [ITU-R BT.1702].
Effects	Flash spacing	360 ms	ITU-R BT.1702-3 accepts flashes whose leading edges are at least 360 ms apart, at any brightness [ITU-R BT.1702].
Effects	Gap between bursts	5 s	ITU-R BT.1702-3 warns about flashing that lasts more than 5 s [ITU-R BT.1702]; bursts of under 1.3 s, at least 5 s apart, never come close.
Light settings	Acclimation from	50 %	One light maker recommends 30 days at 50 %, and another starts at 50 % and reaches the full program over 50 days.
Light settings	Acclimation days	30 days	As for the acclimation's start; longer than the slowest coral adjustment measured, over 20 days [DiPerna].
Light settings	Cap fade	60 s	A minute is slow enough not to startle fish, and quick enough to cut heat soon after the water warms.
Scenes	fade	10 s	Quick enough for a photograph, and slow enough not to startle fish in daylight.
Scenes	Night scene fade	60 s	A scene that starts from darkness, such as night viewing, starts gently.
Scenes	duration	3600 s	Makers' manual and maintenance modes run for 1 to 2 hours, and a scene that ends by itself cannot be forgotten.
Scenes	Photo scene	1800 s	Long enough to photograph a tank, short enough that corals miss little of their day.
Scenes	Longest scene	14400 s	One commercial controller's maintenance programs run up to 240 minutes.
Scenes	Ramp from darkness	1200 s	Guidance for laboratory fish asks for 20 to 30 minutes of brightening and dimming, because fish are easily startled when light comes on at night [CCAC].
Groups	Clock agreement	0.5 s	Half a second is below what an owner sees in a fade or a cloud's shadow; both bindings keep time far better.
Lights that run their own programs	Writing a program	60 s	Long enough for a module to read a 2000-byte file over a busy bus, and short enough that an owner sees a change at once.
Lights that run their own programs	Largest fixture program	65536 bytes	Room for 15 curves of 48 points, mixes, and scenes, many times the size of the examples, within the memory of a small light's microcontroller.
Heat and failures	restoreFade	900 s	Between the 20 to 30 minutes that fish guidance asks for at dawn [CCAC] and the quick return that owners expect after a power cut.
Heat and failures	Shortest restore fade	60 s	A fade shorter than a minute brings a light back from darkness almost as suddenly as a switch; a minute matches the night scene's fade and the cap fade (Section 13.4.6).
Heat and failures	Return after cooling	60 s	A light that cools and warms again does not pulse visibly.

21 Conformance

21.1 Controllers

A conforming controller:

1. supports lighting functions, and every member of programs, lighting functions, and program documents that this Specification defines, with the method `oap-1`;

2. computes levels as Section 14 says, within Section 7.10, and drives lights as Section 14.4 says;
3. meets every requirement for controllers in Sections 7 to 19, including those for lights that run their own programs (Section 16);
4. treats a lighting function it cannot run in full, such as one whose `method` it does not implement, as unsupported [OAP-AU Section 19.2.3]: it tells the owner before it accepts the document, raises `unsupported`, and leaves the function's lights to their fail-safe behavior; and
5. passes the test procedures of Section 21.5 that apply to controllers.

21.2 Lights

1. A conforming light with the `local-program` feature runs fixture programs as Section 16 says, within Section 7.10; implements POSIX TZ strings (Section 7.5), the method `oap-1` (Section 9), the generator and effects (Section 11), and `runProgram` and `programRunning` (Section 16.4); reads references (Section 16.2); and passes the test procedures of Section 21.5 that apply to lights.
2. Every conforming light, whether or not it runs programs, meets Sections 7.4, 11.5.5, and 17.1.

21.3 Documents

A conforming program document meets Section 18.1, and a conforming configuration document with lighting functions meets Section 7.2 and every requirement of Sections 8 to 13 that applies to documents, including those a schema cannot express, such as the order of points and the members that only some programs allow.

21.4 Schema and Tools

- `schema/lighting-programs.schema.json`, whose `$id` is <https://openaquariumproject.org/schemas/lighting-programs/0.3>, defines the program document at its root, and the program (`$defs/program`), lighting function (`$defs/lighting`), fixture program (`$defs/fixtureProgram`), program reference (`$defs/programReference`), and saved state of a lighting function (`$defs/savedState`), with the saved state of revision 0.2 (`$defs/savedStateRevision0.2`).
- `channel-kinds.json` holds Table 8, and `checks.json` the safety checks, their test procedures, and the recommended values.
- `tools/validate_lighting.py` validates program documents, Automation configuration documents with lighting functions, device descriptions of lights with fixture programs, and the test vectors, against the schema and the rules of this Specification that a schema cannot express. It warns about points that fall out of order on some date, sudden steps from darkness, short restore fades, bright moonlight, clouds and storms with short fades, lightning outside the day, lightning on more than one light of a function or of a configuration, lights that cannot show a program's lightning, mixes that a light does not use, methods that this revision does not define, and the function type and saved state of revisions 0.1 and 0.2.
- `tools/lighting_reference.py` is a reference implementation of local time from POSIX TZ strings, the method `oap-1`, the generator, effects, levels, daily light, and photosynthetic photons from published spectra.
- `test-vectors.json` holds its results for sun and moon, local time, the generator, effects, levels, and scenes that fade during a storm, written by `tools/make_lighting_vectors.py`.

```
python tools/validate_lighting.py specs/lighting-programs/examples/*.json specs/lighting-programs/test-vectors.json
```

21.5 Test Procedures

Tests use a controller with simulated lights whose levels, commands, and refusals the tester can watch, whose power the tester can remove, and whose clock the tester can set or stop, and a light with the `local-program` feature that can run with no controller. "Within 1 s" allows for the controller's evaluation interval of 1 s [OAP-AU Section 17.1]. Every safety check of Tables 9, 11, 13, 16, 18, 20, 21, 23, and 26 has a test in Table 29, which follows the checks with tests of the other requirements. Each test names the requirements it checks, as sections or as numbered items of sections, and every MUST and MUST NOT of Sections 7 to 22 is checked by at least one test, which `tools/render_spec_tables.py` checks. A pass criterion that checks what a SHOULD asks applies to a product that does it.

Table 29 — Test procedures

Test	Applies to	Requirements	Method	Pass criteria
Channels without a curve stay off (8.6)	Controllers, lights	8.6.1, 8.6.2, 8.6.6	Run the reef example on a light with <code>cool-white</code> , <code>green</code> , <code>far-red</code> , <code>uv</code> , and <code>moon</code> channels, on a controller and as a fixture program.	<code>cool-white</code> follows the <code>white</code> curve and <code>uv</code> the <code>violet</code> curve; <code>green</code> and <code>far-red</code> stay at 0 % all day; <code>moon</code> shows only moonlight. The controller showed this before the program ran, and the controller and the light agree within 0.5 percentage points.
Points in order (8.3)	Controllers, lights	8.3.3, 8.3.4	Run a curve with a point at <code>{"event": "sunset"}</code> at 100 % and then one at <code>"20:00"</code> at 0 %, at 52° N, on 21 June and 21 December.	In June, when sunset falls after 20:00, the <code>20:00</code> point is moved to sunset, so the level drops there at once; in December the level falls from sunset to 20:00. The controller and the light agree.
Color temperature without mixes (8.4)	Controllers	8.4.5	Assign the planted example, with its mixes removed, to a light without the <code>color-temperature</code> feature, to the same light with a light member that has <code>mixes</code> , and to a light with the feature.	The controller refuses the first, and says why. It runs the second with the light member's mixes, and the third with <code>setColorTemperature</code> , within the light's color temperature range.
Programs checked before use (18.2)	Controllers, lights	7.2.6, 8.1.1, 16.1.2, 16.1.4, 18.1.1, 18.2.2, 22.1	Import program documents with a point at 120 %, a lightning <code>rate</code> of 20, and a curve of 49 points, and write each as a fixture program to a light. Then write the light a fixture program with 17 scenes, and one of 70 000 bytes.	Each is rejected. The program that ran before still runs, and the light reports its previous <code>program</code> .
Brighter programs come in gradually (18.2)	Controllers	18.2.4	Replace the minimal example on a light with a copy whose level is 100 % instead of 70 %.	The controller shows that the new program gives about 43 % more light, and proposes an acclimation from 70 % over 30 days.
Sun and moon by the method (9.6)	Controllers, lights	9.6.1, 9.6.2	Compute every entry of <code>sun</code> and <code>moon</code> in <code>test-vectors.json</code> , including those with a POSIX TZ string and those across a change of daylight saving time.	Every event is within 2 s of the vector, every illuminated fraction within 0.001, and every altitude within 0.01°.
Local time on a light (7.5)	Lights	7.5.3	Compute every entry of <code>timeZones</code> in <code>test-vectors.json</code> , and run the minimal example with <code>posixTimeZone CET-1CEST,M3.5.0,M10.5.0/3</code> from 27 March to 31 March and from 23 October to 27 October 2026.	Every offset matches. The light rises from 10:00 local time on every day, including 29 March and 25 October.
The lunar cycle and its shift (10.1)	Controllers, lights	10.1.1	Run the reef example through November 2026 with <code>shiftDays 0</code> , with <code>shiftDays 5</code> , and with a cycle of 28 days whose <code>fullMoon</code> is 2026-10-26T04:12:00Z.	Moonlight is brightest on the nights of the full moons of 26 October and 24 November with <code>shiftDays 0</code> , five days later with <code>shiftDays 5</code> , and on 26 October and 23 November with the cycle of its own. Levels agree with the reference

Test	Applies to	Requirements	Method	Pass criteria
				implementation within 0.5 percentage points.
When moonlight shows (10.2)	Controllers, lights	10.2.1	Run the reef example in Munich for the 24 hours after the full moon of 24 November 2026, and the planted example through the same night.	The reef light's moon channel rises from 0 % as the moon rises and falls to 0 % as it sets, agreeing with the reference implementation. The planted light shows moonlight from 15 minutes after sunset until 00:30, with 15-minute fades.
Moonlight never dims the day (10.3)	Controllers, lights	10.3.2, 10.3.3	Run the reef example on a light that has no moon channel.	Its <code>royal-blue</code> channels show moonlight by night and their curve by day, and never less than either.
Effects repeat exactly (11.2)	Controllers, lights	11.1.1, 11.2.4	Compute every entry of <code>random</code> and <code>effects</code> in <code>test-vectors.json</code> , and run the reef example through the stormy date of the vectors on two lights from different makers, each with lightning on, in a lighting function of its own.	Every draw matches, and every cloud, storm, and flash starts within 1 ms of the vector. The two lights' clouds start within 1 s of each other.
Flash limit (11.5)	Lights	11.5.5	Measure a light's output with a photodiode through a storm with a <code>rate</code> of 10 and <code>flashes</code> 3, with the light's <code>lightning</code> <code>true</code> , and then while it is sent <code>set</code> commands that change between 0 % and 100 % ten times a second.	No second holds more than three flashes; flashes of the storm start at least 360 ms apart, and its bursts are at least 5 s apart.
Lightning only when turned on (11.5)	Controllers, lights	11.5.7	Import the reef example and assign it to a light; then turn lightning on for the light; then try to import a copy whose lightning <code>kinds</code> include <code>red</code> .	The light shows the storm without lightning until the owner turns it on; the controller warns about flashing light before it does; the copy with <code>red</code> is rejected.
No lightning from a controller (11.6)	Controllers	11.6.3	Drive a light without the <code>local-program</code> feature with the reef example through the stormy date of the vectors.	The storm dims the light as the vectors give, the light never flashes, and the validator warns about the light.
Scenes replace effects (11.1)	Controllers, lights	11.1.3, 14.1	Start the reef example's photo scene during the storm of the stormy date, with lightning on, and end it after 5 minutes. Later in the storm, stop the light with an override for 2 minutes.	No flash shows from the start of the scene's fade-in to the end of its fade-out, nor during the restore fade after the override. The light fades evenly from the storm's level to the scene's, and back to the storm's, without a jump, and lightning resumes after each fade.
One light flashes (11.5)	Controllers, configurations	11.5.9	Turn lightning on for the left-hand light of the reef-system example, whose right-hand light already has it. Then turn it on for a light of another lighting function of the same controller. Validate a copy of the example with lightning on both display lights, and one with lightning on a display light and a light of another function.	The controller warns before it turns lightning on each time, saying that each light limits its own flashes, not those of several lights together. The validator warns about each copy.

Test	Applies to	Requirements	Method	Pass criteria
Clouds and storms fade gently (11.3)	Configurations	11.3.6, 11.4.6	Validate a copy of the reef example whose clouds have a <code>fade</code> of 0 s, and one whose storm has a <code>fade</code> of 5 s.	The validator warns about each.
Caps over everything (12.1)	Controllers, lights	12.1.2	Run the reef example with <code>maxLevel</code> 40 and <code>maxLevels uv</code> 10 through the stormy date, with the photo scene at noon.	No channel exceeds 40 %, and <code>uv</code> never exceeds 10 %, including during lightning and the scene.
Acclimation (12.2)	Controllers, lights	12.2.1	Run the reef example with an acclimation from 50 % over 30 days, and compare each day's levels with those without it.	On day <i>n</i> , daylight, lightning, and scenes are $(50 + 50 \times n / 30)$ % of those without acclimation, changing at local midnight; moonlight is the same; from day 30 they are equal.
Dimming on a condition (12.3)	Controllers	12.3.1, 12.3.3, 12.3.4, 12.3.5, 16.4.5	In the reef-system example, warm the water above 28.5 °C, then disconnect the temperature probe, then reconnect it at 27 °C. Repeat with the lights driven by the controller. Then, with the lights running their own programs, of Device Classes revision 0.5, start the photo scene while the water is cool, warm the water during the scene, and stop the controller 5 minutes later. Repeat with lights of revision 0.3, with the controller running.	Each light's channels fall to at most 50 % over 60 s while the water is warm and while the probe is disconnected, with <code>input-unavailable</code> raised, and return over 60 s at 27 °C. Lights that run their own programs are sent <code>runProgram</code> with <code>maxLevel</code> 50, a <code>fadeTime</code> of 60 s, and no <code>revertAfter</code> . During the scene, the lights of revision 0.5 are written <code>programMaxLevel</code> 50, and sent the scene again under it with the same <code>revertAfter</code> ; the scene continues, and after its revert they show their program at no more than 50 %, though the controller is stopped. The lights of revision 0.3 are sent the scene again in the same way, and every <code>runProgram</code> they are sent carries <code>maxLevel</code> 50, including the one when the scene's fade-out begins.
Scenes end by themselves (13.2)	Controllers, lights	13.2.4, 13.2.5, 13.2.6, 13.2.7, 17.5	Start the reef example's photo scene on lights of Device Classes revision 0.5 that run their own programs, and stop the controller 5 minutes later. Repeat with the still-water scene, with lights of revision 0.4 and a running controller, and with lights of revision 0.2 whose controller keeps running.	The lights fade to the scene in 10 s from what they showed, and 10 s before the scene's end fade back to their program, reaching it at the end, though the controller is stopped. The scene's commands carry <code>revertAfter</code> the time until the fade-out, and the still-water scene is sent as <code>runProgram</code> with <code>effects false</code> . Where

Test	Applies to	Requirements	Method	Pass criteria
				the controller keeps running, it sends <code>runProgram</code> with a <code>fadeTime</code> of 10 s when the fade-out begins, and the lights of revision 0.4 show their clouds again after the still-water scene. The lights of revision 0.2 are sent the scene with <code>revertAfter</code> the time until its end.
Scenes from conditions (13.2)	Controllers	13.2.2	In the reef-system example, start maintenance mode, which lasts 2 hours; later, start it again and end it after 10 minutes.	The work scene starts with the mode, and ends after 60 minutes the first time, though the mode continues, and after 10 minutes the second. Each time, the lights return to their program.
Fades and the restore fade (13.4)	Controllers, lights	13.4.1, 13.4.2, 13.4.3, 17.4.1	Compute every entry of <code>scenes</code> in <code>test-vectors.json</code> . Then, at noon, remove and restore a light's power; start and end a power-loss mode; and replace its program. Do this with the light running its own program and with the light driven by the controller. Then remove and restore the power of a light of Device Classes revision 0.5 whose <code>powerOnBehavior</code> is <code>fail-safe</code> and whose <code>failSafeState</code> lights each channel at 50 %, with no controller, and of a light that a latched stop holds.	Every level of the scenes is within 0.5 percentage points of the vector. Each time, the light rises from what it showed to the program's level evenly over its <code>restoreFade</code> , by default 900 s, and never jumps; after power-up, the light the controller drives fades up by itself over 900 s, until its controller resumes control and continues the restore fade from what it shows. The light with the fail-safe state fades up to 50 % over 900 s, and the latched light stays off.
Gentle restore fades (13.4)	Configurations	13.4.6	Validate a copy of the reef-system example whose display lighting function has a <code>restoreFade</code> of 30 s.	The validator warns about the restore fade.
Lights in step (15.2)	Controllers, lights	15.2.1	Run the display lights of the reef-system example, from two makers if possible, with lightning on the right-hand light, through the stormy date of the vectors, and record both lights' levels.	The right light's curves follow the left's 300 s later, and its clouds 20 s later, within 1 s and 0.5 percentage points, and both enter and leave a scene together.
Clocks together (15.3)	Lights	15.3.2	Run a light that runs its own program for 7 days with its controller, and compare their clocks every hour.	They never differ by more than 0.5 s.
The same program on a light (16.3)	Lights	7.10.3, 16.3.1	Write the fixture program of each entry of <code>levels</code> in <code>test-vectors.json</code> to a light, set its clock to each sample time, and read its levels.	Every channel is within 0.5 percentage points of the vector.
Writing fixture programs (16.2)	Controllers	16.2.1, 16.2.2, 16.2.3	Assign the reef example to a light over the Device Protocol and to one on the module bus, then change one light's <code>scale</code> .	Within 60 s, the network light holds its fixture program as text, and the module a reference whose file

Test	Applies to	Requirements	Method	Pass criteria
				the controller serves; both hold the new program within 60 s of the change.
Checking a program file (16.2)	Controllers, lights	16.2.4, 16.2.6	Write a light a reference whose file has another hash, then one whose file is cut short, then a correct one.	After each of the first two, the light keeps its previous program and reports its previous program; the controller raises <code>program-not-loaded</code> 60 s later, with the light's output as its subject, and drives the light meanwhile. After the third, the light runs the new program, and the alarm clears.
Returning to the program (16.4)	Controllers, lights	16.3.3, 16.4.1, 16.4.2, 16.4.3, 16.4.4, 16.4.5	Stop a light that runs its own program with a power-loss mode, and end the mode; start the photo scene and end it after 5 minutes; and cut contact for longer than the fail-safe timeout, then restore it, each with a cap of 60 % from <code>dim</code> in force. Do this with a light of Device Classes revision 0.5 and one of revision 0.3. Then assign the program, with <code>runOn Light</code> , to a light of Device Classes revision 0.1 that does not report <code>org.openaquariumproject.program-running</code> .	Each time, the controller sends <code>runProgram</code> , and the light returns to its program with the fade the command gives, at no more than 60 %. Every <code>runProgram</code> to the light of revision 0.3 carries <code>maxLevel</code> 60. <code>programRunning</code> is <code>false</code> while the light is stopped and during the scene, and <code>true</code> after. The controller drives the light of revision 0.1 itself.
Running without a controller (16.3)	Lights	16.3.2	Stop the controller of a light that runs the reef example, with <code>failSafeAction continue</code> and lightning on, for 48 hours across the stormy date of the vectors, with a cap of 70 % from <code>dim</code> in force; then repeat while the light is stopped by an override with 30 minutes left, and while an interlock's latched stop holds it.	The light keeps its curves, moon, clouds, storm, lightning, acclimation, and its cap of 70 %, matching the reference implementation, and reports <code>communication-lost</code> . The light stopped by the override stays off until its revert falls due, and then returns to its program; the latched light stays off.
Thermal limits over everything (17.1)	Controllers, lights	17.1.1, 17.1.2, 17.1.3, 17.1.4, 17.1.5	Warm a light with the <code>fixture-temperature</code> feature until it dims: while it runs its program, during a scene, during lightning, and after <code>set</code> with every channel at 100 %. Then let it cool.	Its output stays within its thermal limit from every source. It reports <code>thermalDimming true</code> and <code>overtemperature</code> , the controller raises the alarm with the light's endpoint as its subject, shows that the light is dimmed for heat, and sends no command to make up for it. After cooling, it returns to its levels over at least 60 s.
No clock, no program (17.4)	Lights	7.5.6, 17.4.2	Power up a light that runs its own program, with its clock not valid and no controller; after 10 minutes, connect its controller; then stop the controller for 2 hours.	Its channels stay off, and it reports <code>clock-invalid</code> , until it has the time; it then rises to the

Test	Applies to	Requirements	Method	Pass criteria
				program's level evenly over the restore fade. Without its controller, its clock stays valid, and it keeps its program.
A controller without a clock (17.4)	Controllers	7.5.5, 17.4.4	Restart the controller with its clock not valid, with one light it drives and one that runs its own program.	The driven light takes its fail-safe behavior, and <code>input-unavailable</code> is raised. The other light keeps its program.
Temporary states end on the light (17.2)	Controllers	17.2.1	Capture the commands a controller sends to lights it drives and to lights that run their own programs, for scenes, overrides with an end, and modes with a duration.	Every one has <code>revertAfter</code> , with the time the state has left, except <code>setColorTemperature</code> to a light of Device Classes revision 0.1, for a scene of a color temperature program without mixes (Section 13.2.7).
Invalid configurations	Controllers, configurations	7.2.3, 7.2.4, 7.2.6, 7.3.3, 9.1.3, 9.5.2, 12.1	Import configurations that each break one rule for documents: a lighting function whose <code>outputs</code> list an output that has no light member; a lighting function with <code>onUnavailable</code> ; a light member with <code>kind</code> for a light output; a light member with <code>mixes</code> in a function whose program is a program of curves; a function that fails its schema; a program whose sun source is <code>configuration</code> in a configuration without <code>astronomy</code> ; a seasonal table with a date twice; and a table entry whose sunrise is after its sunset.	The controller and the validator reject each one, saying why, and the running configuration is unchanged.
Lighting in the control process	Controllers	7.1.4	With a light the controller drives, whose <code>failSafeAction</code> is <code>state</code> , stop the controller's evaluations while its communication task keeps running.	The controller stops supervising the light, which reaches its fail-safe state within its <code>failSafeTimeout</code> plus 2 s.
Levels and output	Lights	7.4.2	Read the manufacturer's documentation of a light, and measure each channel's output with a light meter at 10 %, 50 %, and 100 %.	The documentation says how the light's output depends on the level, and the measurements agree with it.
Scenes kept across a restart	Controllers	7.7.4	Start the reef-system example's photo scene, and restart the controller 5 minutes later. Start it again, stop the controller, and start it again after the scene's end has passed.	After the first restart, the scene continues with the time it had left. After the second, the lights return to their program. The saved state holds the scene while it lasts, and not after.
What the owner sees	Controllers	7.9	With the reef-system example running, look at each lighting function in every app the controller offers, during the photo scene, with the right-hand light dimmed for heat, the water warm, and an acclimation in force.	Each app shows every item of Section 7.9: the program, which light runs it itself, today's sun and moon, each channel's kind and level, what holds it down, the scene's time left, the acclimation's day and share, today's effects

Test	Applies to	Requirements	Method	Pass criteria
				on request, and the light's overtemperature.
Levels by the steps	Controllers	7.10.1, 14.1, 14.2, 14.4.1	Compute every entry of <code>levels</code> and <code>scenes</code> in <code>test-vectors.json</code> on a controller, and drive a light, and a light with the <code>color-temperature</code> feature that runs the planted example, with them.	Every level is within 0.5 percentage points of the vector, and the commands keep each channel within 1 percentage point of it.
Mixes from spectra	Controllers	8.5.5, 8.5.6	Assign the planted example, with its mixes removed, to a light without the <code>color-temperature</code> feature whose channels each give their <code>spectrum</code> and <code>photonFlux</code> , and measure its light with a spectrometer at 2700 K and 6500 K.	The controller proposes mixes for the light, says which of the program's color temperatures the light cannot show, and runs the program with the mixes once the owner confirms them. The measured correlated color temperature is within 5 % (provisional) of the program's wherever the light can show it.
Light in photons	Controllers	18.2.4, 19.3, 19.4	Run the reef example on the light of <code>examples/light-network.json</code> , whose channels give their <code>spectrum</code> and <code>photonFlux</code> , for a day, and replace its program with a copy whose white curve peaks at 70 % instead of 45 %. Then give the light's vessel <code>dimensions</code> .	The controller shows the light's photosynthetic photon flux, and its photosynthetic photons over the day, as estimates, within 1 % of what the reference implementation computes from the same levels. It compares the two programs by their photons. Where it estimates the daily light integral, it shows it as an estimate, the photons over the area of the water's surface, and records no observation of it.
Methods a product does not implement	Controllers, lights	9.1.4, 16.1.4, 21.1.4	Import a configuration whose lighting function's program has the <code>method org.example.other</code> , and write the same program, as a fixture program, to a light.	The controller tells the owner before it accepts the document, raises <code>unsupported</code> , and runs none of the function's lights, which take their fail-safe behavior. The light rejects the fixture program and keeps its previous one. The validator warns about the method.
Stops are never delayed	Controllers	13.4.4	During a slow fade of a light the controller drives, start an interlock that stops it, and then a power-loss mode.	Each stop is sent at once, without a <code>fadeTime</code> , and the light is off within 1 s.
Export and import	Controllers	18.2.1	With the network disconnected, export each program the controller runs as a program document, and import it into a second controller.	Each document is valid, and the second controller runs each program the same way, within 0.5 percentage points.

Test	Applies to	Requirements	Method	Pass criteria
What an import shows	Controllers	18.2.3, 18.2.5	Import the reef example and assign it to a light with a green channel. Then import a program whose sun source is configuration into a configuration without astronomy .	Before either runs, the controller shows which curve each channel follows, that green stays off, and where the sun comes from, and waits for the owner to confirm. It asks for a location for the second.
Authors and licenses kept	Controllers	18.3.2	Import a program document whose program has an author and the license CC-BY-4.0 , and share it again.	The controller shows the author and the license, and the shared document keeps both.

22 Security Considerations

1. **Programs are data.** A program contains no code, and a product parses it with the limits the schema sets: 15 curves of 48 points, 16 mixes, 16 scenes, and fixture programs of at most 65536 bytes. A light **MUST** reject a program that exceeds them, rather than run part of it.
2. **Authorization.** Changing or importing a program needs the [change-configuration](#) operation, and starting or ending a scene the [start-mode](#) operation, which covers scenes as well as modes [OAP-SA Table 6]. Lights accept fixture programs, and [runProgram](#), only from their controller [OAP-FS Section 7.5].
3. **Shared programs.** A program from someone else can ask for more light than corals tolerate, or for lightning. It can never exceed the owner's caps, the flash limit of Section 11.5, or a light's thermal limits, and the controller shows it, and its light compared with the program it replaces, before it runs (Section 18.2).
4. **Program files.** A light reads a fixture program file only from its controller, and checks its size and hash before it runs it (Section 16.2.4). Over the Device Protocol, the file and the reference that names it come over the session authenticated with the controller's key [OAP-DP Section 10], so a program cannot be swapped on the way. The module bus does not authenticate messages [OAP-MB Section 15]: the hash shows a file that was damaged on the way, but someone who can send on the bus can write both a reference and the file it names, as they can send any command. The module bus is a wired bus inside the owner's system, and a later revision of the Module Bus specification may add message authentication [OAP-MB Section 15.3].
5. **Flashing light.** Lightning is limited for people as well as for fish (Section 11.5): each light enforces the limit whatever it is sent, and a controller warns before it turns lightning on for more than one light, because the limit holds for each light, not for several together.
6. **Location.** A program's location can reveal where an owner lives (Section 18.3).

Appendix A — Approval Checklist

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

Item	Status	Details
Section 2 describes compliance with all five tenets.	Done	
A public review period of at least 30 days has been announced and completed.	Not started	
At least two independent implementations exist, at least one from an organization not represented among the Authors.	Not started	
The implementations have passed the test procedures in Section 21.5, including a light that runs its own program and a controller.	Not started	
A program written by one implementation has run on a light of another, and the two showed the same levels and effects.	Not started	
Every normative reference is publicly available.	Done	

Item	Status	Details
The checks file, channel kinds file, schema, test vectors, and examples are published, and the examples and test vectors pass the validator.	Done	
Experienced aquarists have reviewed the recommended values, and a specialist in photosensitive epilepsy the limits on lightning (Appendix D, issues 8 and 17).	Not started	
The Open Issues appendix is empty, or each open issue is explicitly deferred.	Not started	
No unresolved Exclusion Notices for this Specification are recorded in <code>Notices.md</code> .	Done	
The Maintainer has recorded consensus to approve.	Not started	

Appendix B — Examples (Informative)

The `examples` directory contains:

- `minimal.json`: the smallest useful program document, a white light from 10:00 to 20:00 with half-hour fades.
- `reef.json`: a reef program that follows the sun and moon of Lizard Island, with solar noon moved to 13:00 wherever it runs: blues that lead the whites at dawn and outlast them at dusk, violet and ultraviolet at midday, moonlight while the moon is up, light clouds on most days, an afternoon storm with lightning on about one day in twelve, and photo, still-water, and working scenes.
- `planted.json`: a planted-tank program set by color temperature, from 2700 K at dawn to 6500 K at midday, on a seasonal table that lengthens the day a little in summer, with mixes for lights with warm-white, cool-white, red, and blue channels, faint moonlight in the early night, a few thin clouds, and viewing and photo scenes.
- `reef-system.json`: an Automation configuration in which two lights run the reef program by themselves, staggered by five minutes, with clouds that pass from left to right and lightning on the right-hand light only, an acclimation for a new light, a working scene during a two-hour maintenance mode, and dimming while the water is warm; and a refugium light on a switch that runs a reverse daylight cycle from the owner's own location; with an active photo scene in its saved state, as `state.functions`.
- `light-network.json`: a network light's device description of Device Classes revision 0.5, with the fixture program of the right-hand light written as text, its `programMaxLevel`, and the `spectrum` and `photonFlux` of each channel.
- `light-module.json`: a module bus LED driver's device description of Device Classes revision 0.5, with a reference to the fixture program of the left-hand light.

Appendix C — Commercial Features (Informative)

How this Specification covers the lighting features of commercial controllers and lights. The products compared are three widely used aquarium controllers, and reef and planted-tank lights from several makers.

Table 30 — Commercial lighting features

Feature	In commercial products	This Specification
Curves of points for each channel	Every product; from 4 editable points to 24, and 31 in one preset; none says how it fills the gaps	Curves for kinds, of up to 48 points, with defined interpolation (Section 8.3)
Times from sunrise and sunset	Offsets from a controller's sun times; a sunrise that moves a light's whole program; templates set by sunrise and sunset	Sun events and shares of the day (Section 8.2)
A location's or a season's day	A controller's table of monthly sun times; another controller's day length of a chosen place; the monthly tables of coral spawning research	A location with the method <code>oap-1</code> , or a seasonal table with its daylight (Section 9)
Color temperature	A channel set in kelvin; a kelvin slider; a single color setting that crossfades blue and white	Color temperature programs, changing in mireds, with mixes for other lights (Sections 8.4 and 8.5)

Feature	In commercial products	This Specification
Lunar cycle	Two controllers and two light makers by date; one light maker on 28 days and another on 30	The moon in the sky, or a cycle of its own (Section 10.1)
Moonrise and moonset	One controller	Moonlight while the moon is up (Section 10.2)
Adjustment for breeding	One controller's reference full moon; research systems that move a reef's dates and clock	<code>shiftDays</code> of the moon, and of the sun with the moon, <code>shift</code> , and <code>noonAt</code> (Sections 9.1 and 10.1)
Clouds	One controller and three light makers; another controller on some lights	Clouds with coverage, depth, duration, and fade, the same on every product (Section 11.3)
Storms and lightning	One controller and two light makers; another controller on some lights; no maker publishes a flash rate	Storms, and lightning within flash limits that protect people with photosensitive epilepsy (Sections 11.4 and 11.5)
Acclimation	One controller and four light makers; another controller for some lights	Acclimation by date (Section 12.2), and when a brighter program replaces another (Section 18.2)
Dimming or switching off when hot	One controller dims in proportion to water temperature; two others switch off; lights protect themselves, with no published thresholds	Dimming on a condition (Section 12.3), and a light's own limits over everything (Section 17.1)
Staggered sunrise and gliding clouds	One maker's controller and lights offset by up to an hour, and clouds by seconds; another maker's staggered sunrise across grouped lights	<code>offset</code> and <code>cloudOffset</code> (Section 15.2)
Scenes and photo mode	One controller's maintenance programs and photo mode, another's modes, one light maker's manual override, and two light makers' feed modes	Scenes that end by themselves (Section 13)
Running without the controller	Most lights; one maker's light bar runs at 100 % without its data cable	Fixture programs with <code>continue</code> , and no program without a clock (Sections 16 and 17)
Program files	Binary, XML, JSON, or cloud only; none read another maker's	Program documents (Section 18)

Appendix D — Open Issues (Informative)

Issues 1 to 17 were open in revision 0.1, and issues 19 and 20 arose in revision 0.2. The Project Lead decided issues 7 and 18 on 15 October 2026, and issues 1, 14, 19, and 20 on 16 October 2026, accepting the recommendation in each. On 17 October 2026 the Project Lead froze the scope of revision 1.0, and extended the decision of issue 18 to lights of different lighting functions; revision 0.3 applies that decision, uses what Device Classes revision 0.4 added for issues 1, 19, and 20, and resolves issues 2, 4, and 6, which other specifications settled. Issue 21 records the choices that revision 0.3 made where those decisions left them open, which the Project Lead accepted on 18 October 2026. Issues 9 to 13, 15, and 16 are deferred beyond revision 1.0, and are tracked outside this Specification. Issues 8 and 17 stay open until a specialist, the bench tests, and owners' experience confirm them. Each issue says how a revision resolves it, or why it stays open. Issue numbers do not change, because other specifications refer to them.

1. **Standard spectra for channel kinds.** *Decided by the Project Lead on 16 October 2026; resolved in revision 0.2: no standard spectra; lights publish their own from Device Classes revision 0.4 (its Section 7.15; Appendix D, issue 24), which revision 0.3 uses.* This Specification leaves spectra to manufacturers, and the question was whether it should define them. For standard spectra, or required spectral data: makers' color names are not comparable, and even datasheets bin royal blue by peak wavelength at one LED maker and by dominant wavelength at another; channels are often blends that one peak cannot describe; reef "kelvin" is a label rather than a measured color temperature; estimating PAR, or a coral's useful light, needs spectra; and horticultural lighting requirements already ask for the spectrum of each channel of a tunable fixture [DLC]. For leaving spectra to makers: DALI dropped certification of its color primaries because calibration is maker-specific [DALI]; horticultural requirements leave color names to LED makers [DLC]; LED bins and blends make fixed spectra unrealistic; and Matter and DALI standardize what is controlled, such as color temperature, rather than spectra. The Project Lead decided on a middle

course. Channel kinds stay names of colors, with no standard spectra, and each kind has an informative band in `channel-kinds.json` and Table 8, with the boundary between `uv` and `violet` at 400 nm, as CIE puts the end of UV-A. Device Classes revision 0.2 already says that `peakWavelength` is the peak wavelength, not the dominant one. Its revision 0.4 lets each channel of a light publish its spectral photon distribution, `spectrum`, and its photon flux at 100 %, `photonFlux`, in the unit of photon flux that Data Model revision 0.3 adds, `umol/s`. Programs stay portable by kind, and revision 0.3 uses what lights publish: a controller proposes mixes made from a light's spectra, so that a color temperature program without mixes runs on a light without the `color-temperature` feature (Sections 8.4.5 and 8.5.6); it estimates a light's photosynthetic photon flux, its photons over a day, and from them a vessel's daily light integral (Section 19); and it compares programs by their photons (Section 18.2.4). Moonlight given in light rather than percent, which also needs what lights publish, is deferred beyond revision 1.0. An `amber` kind can be added to Device Classes when lights have one.

2. **Function type.** *Resolved in revision 0.3, with Automation revision 0.3.* The Automation specification carried lighting functions under a provisional extension type, with their saved state under `state.extensions["org.openaquariumproject.lighting"]`. Automation revision 0.3 lists `lighting` among its function types, keeps each hosted function's saved state as `state.functions["<function id>"]` (its Section 11.10 and Table 14), describes a lighting function's demand on a light, its program (its Section 8.3.2 and Table 25), and points its `light` preset to the new type (its Table 28). Sections 7.2 and 7.7 now use the type and the saved state of that revision, and a controller still reads the names of revisions 0.1 and 0.2 (Section 7.2.8).
3. **The `light` class.** *Resolved in revisions 0.2 and 0.3.* Device Classes revision 0.2 answered what revision 0.1 asked of it: `program` as a fixture program or a reference to one, reported only once it is loaded and checked; `runProgram` and `programRunning`; even fades; `revertAfter` for `setColorTemperature`; channels' `minLevel` and `resolution`; `overtemperature` while `thermalDimming` is true, and `thermalScale`; `peakWavelength` as the peak wavelength; and the device fault `clock-invalid` (its Section 9.4 and Appendix D, issue 20). Revision 0.3 added fades on reverts (its Section 7.4.12), which Section 13.2.5 relies on, and the fade up to restored levels (its Section 8.3.3). Revision 0.4 added the cap of a light's program, `programMaxLevel`, fades of color temperature evenly in mireds, spectra, and fades to fail-safe states that light a light (its Sections 7.4.10, 7.13.7, 7.15, 8.3.5, and 8.3.6), and revision 0.5 latched stops and reverts that restore a program's effects (its Sections 7.4.13 and 7.13.8). Sections 8, 12, 13, 16, 17, and 19 now use them, and keep `org.openaquariumproject.run-program` and `org.openaquariumproject.program-running` only for lights of revision 0.1 (Section 16.4.4). Its issue 9, on groups, is answered by Section 15: a group is the lights of one lighting function, which exchange no messages.
4. **Program files over the bindings.** *Resolved in revision 0.3, with Module Bus revision 0.4 and Device Protocol revision 0.2.* This Specification asked the Module Bus specification to say that a controller serves `oap/lighting/` files while a light holds a reference to one, and the Device Protocol specification that writing a reference offers its file to the light (Section 16.2). Module Bus revision 0.4 says so (its Section 11.8; Appendix C, issue 14), as Device Protocol revision 0.2 did (its Section 13.2.2). Section 16.2.3 now cites both, and the paths are no longer provisional.
5. **Daily light integral.** *Resolved in revision 0.2.* Data Model revision 0.2 adds the parameter `daily-light-integral`, in `mol/m2/d` (its Section 9.2), which answers its issue 5. Section 19.2 now uses it, and reads the provisional extension of revision 0.1 as it.
6. **One astronomy.** *Resolved in revision 0.3, with Automation revision 0.3.* The Automation specification was asked to name the method `oap-1` for its sun and moon events, find moonrise and moonset from the moon's altitude of Section 9.4, take the solar day of Section 9.3 for a shifted location, add `noonAt` and `shiftDays` to its `astronomy`, and limit latitudes as Section 9.1.6 does, so that schedules and lights agree to the second. Automation revision 0.3 does all of these (its Section 10.7; Appendix D, issue 26). Section 9.1.3 now reads the configuration's `noonAt` and `shiftDays`, a fixture program carries them (Section 16.1.3), and `noonAt` is allowed with a seasonal table that has a `location`, as Automation allows it with a `sunTable` (Table 10 and Section 9.5.4).
7. **Fail-safe configuration of lights the controller drives.** *Resolved in revision 0.2.* With `continue`, a light the controller drives keeps its last levels when the controller fails: midday light through the night, or darkness through the day (Section 17.2). `state` with every channel off would give darkness instead, which suits

some livestock better and others worse. The Project Lead decided on 15 October 2026 to keep Fail-Safe Behavior Table 3 as it is, with `continue` and `last` for every light, and that a light restoring a level after power-up fades up to it (Fail-Safe Behavior Appendix C, issues 11 and 15). Table 24 and Section 17.4.1 now say so: a light that runs its own program fades up with its restore fade, and a light the controller drives over 900 s, until its controller continues the restore fade (Device Classes Section 8.3.3), so Section 13.4.3 holds for every light that dims. Fail-Safe Behavior Appendix C, issue 3, asked whether devices should be told when the system is in power-loss mode; its revision 0.2 decides that they are told only through what their controller commands, such as a cap on a light's own program (its Section 9.5).

8. **Lightning by default.** *Open: to be confirmed with a specialist in photosensitive epilepsy.* This Specification turns lightning off unless the owner turns it on for a light, after a warning (Section 11.5.7), and limits flashes to three a second, at least 360 ms apart, in bursts at least 5 s apart, never in red, with a warning before lightning is turned on for a second light (Section 11.5.9). The defaults and limits should be confirmed with a specialist in photosensitive epilepsy, and the Project Lead should then confirm the default.
9. **Smaller lights.** *Decided by the Project Lead on 16 October 2026; resolved in revision 0.2: one conformance level.* A light must accept 15 curves of 48 points and fixture programs of 65536 bytes. Commercial lights hold fewer points, some as few as 4 a channel (Appendix C). A smaller conformance level, with a controller simplifying programs for such lights, would let them keep a simpler day by themselves, but the same program would then run differently on different lights, which this Specification exists to prevent, and every controller and test would have to handle two levels. The largest program takes less than 16 kB once parsed into numbers, which a light can keep in its flash memory, so the limits ask for design rather than for memory. The Project Lead decided on one conformance level: a light that cannot run programs up to the limits is driven by its controller (Section 16.1.5).
10. **Numbers to confirm.** *Open until the bench tests of the breadboard controller and prototype lights, and experienced reefkeepers and planted-tank keepers, confirm them.* The tolerances, fades, defaults of clouds and storms, flash timing, the restore fade, and the acclimation defaults are provisional. So are the numbers that revision 0.2 adds: the 10 s that the fades of clouds and storms should last at least (Sections 11.3.6 and 11.4.6), the limit of one command a second to a light (Section 14.4.4), and the 900 s over which a light the controller drives fades up after power-up, which Device Classes Section 8.3.3 sets. Revision 0.3 adds the 60 s that a restore fade should last at least (Section 13.4.6).

Decided by the Project Lead

18. **Lightning across a group.** *Resolved in revision 0.2; extended in revision 0.3, as the Project Lead decided on 17 October 2026.* The Project Lead decided on 15 October 2026 that only one light of a lighting function flashes, and that the validator warns when lightning is on for more than one. Section 11.5.9 says so, with the reason: the flashes of lights that keep their own clocks can fall up to a second apart, and together break the limit of three flashes a second. Lights of different lighting functions, such as a display's and a refugium's, can do the same, and the text read as though a group could never flash more than three times a second, while the rule is a recommendation, with warnings. The Project Lead decided on 17 October 2026 to keep the rule as applied, to extend the controller's warning to turning lightning on for a second light on the same controller, whatever its lighting function, and to reword the text that read as a guarantee. Revision 0.3 does so in Sections 2.4, 6.4, 11.5.9, and 22.5, and the validator also warns about a configuration with lightning on for more than one light. Issue 8, on confirming the limits with a specialist, stays open.

The following issues arose in revision 0.2:

19. **A cap that a scene does not end.** *Decided by the Project Lead on 16 October 2026; resolved in revision 0.3, with Device Classes revision 0.4 (its Section 7.13.7; Appendix D, issues 44 and 47).* `runProgram` was the only way to change the cap of a light's program, and, without `revertAfter`, it cancels a pending revert, so it would end a scene (Section 12.3.5). While a scene was shown, the controller therefore sent the scene again with the new cap, and sent the light's program its cap after the scene; if the controller failed during the scene, the light returned to its program with the cap it had before. The Project Lead decided that the Device Classes specification is to give a light's program a cap that a controller changes with a config attribute, without taking the light off what it shows, and that `runProgram`'s `maxLevel` also sets, so that

dimming on a condition reaches every light at once, and stays after a scene's revert. Device Classes revision 0.4 gives it as `programMaxLevel`. Section 16.4.5 now writes it to a light of revision 0.4 or later that shows anything but its program, so that the cap stays after a scene's revert even if the controller fails, and keeps the earlier procedure for lights of earlier revisions, every `runProgram` to which now carries the cap in force.

20. **Fades of color temperature.** *Decided by the Project Lead on 16 October 2026; resolved in revision 0.3, with Device Classes revision 0.4 (its Section 7.4.10; Appendix D, issue 45).* Device Classes revision 0.3 said that `setColorTemperature` fades evenly, and that a fade changes each level in a straight line (its Section 7.4.10), but not how the color temperature changes. The Project Lead decided that it is to say that the color temperature changes evenly in mireds, as Section 8.4.3 does, and the intensity evenly, so that a scene of a color temperature program fades in the same way on the light and in Section 14.2. Device Classes revision 0.4 says so, and Sections 8.4 and 14.4.2 now rely on it for lights of that revision or later.

The following issue records the choices of revision 0.3:

21. **Choices of revision 0.3.** *Resolved in revision 0.3: the Project Lead accepted each choice on 18 October 2026.* Revision 0.3 made these choices where the decisions it applies, and the findings it resolves, left them open:
- a light member's `mixes`, which a controller proposes from the spectra of the light's channels for the owner to confirm, by a method that this Specification describes but does not require, let a color temperature program without mixes run on a light without the `color-temperature` feature, and every product computes the same levels from them, because they are configuration (Sections 8.4.5, 8.5.5, and 8.5.6);
 - where every channel of a light publishes its spectrum, a controller shows the light's photosynthetic photon flux and its photons over a day, compares programs by their photons, and may estimate a vessel's daily light integral over its water's surface, which it shows as an estimate and records as no observation (Sections 18.2.4, 19.3, and 19.4);
 - a validator warns about lightning on more than one light of a configuration, as well as of a lighting function (Section 11.5.9);
 - a controller sends `runProgram` when every scene's fade-out begins, rather than relying on the revert alone, so that a scene fades back over its own fade after a cap, and a light of Device Classes revision 0.4 or earlier shows its effects again (Section 13.2.6);
 - a light shows no lightning during a restore fade, as during a scene's fades (Sections 11.1.3 and 14.1);
 - a restore fade should last at least 60 s (provisional), and a validator warns about a shorter one (Section 13.4.6);
 - `noonAt` is allowed with a seasonal table that has a `location`, so that every configuration's astronomy fits in a fixture program (Table 10; Sections 9.5.4 and 16.1.3); and
 - the test "Mixes from spectra" passes when the measured color temperature is within 5 % (provisional) of the program's (Table 29).