Open Aquarium Project

OPEN AQUARIUM PROJECT SPECIFICATION

Flow Programs

PRE-DRAFT

PRE-DRAFT

Revision	0.3
Status	Pre-Draft. A candidate document submitted for consideration.
Date	2026-10-18
Identifier	flow-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 `flow-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, patterns 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, patterns file, schema, test vectors, and examples under the Apache License 2.0. Every pattern is defined by an equation of local time, and random patterns by a published random number generator, so that anyone can compute what a program makes a pump do at any moment (Section 8). A reference implementation, with test vectors, is published with the validator, so that makers can check that their pumps and controllers agree (Section 20.4). Every requirement has a published test procedure (Section 20.5).

2.2 Interoperability

A program names patterns from a shared library, such as a pulse or a gyre, rather than the modes of one product, and gives speeds as shares of whichever pump runs it (Section 8). The same program runs the same way on a controller and on a pump that runs it by itself, to within half a percentage point and 50 ms (Section 7.9). Pumps from different makers in one group stay in step, in sync, in anti-sync, or in sequence, because each computes its pattern from the same local time; they exchange no messages (Section 11). Programs written by one product can be shared with owners of another, as program documents (Section 17). Flow is a function type of the Automation specification, `flow`, so it moves between controllers with the rest of an owner's configuration. It uses the Device Classes `pump`, `level-output`, and `switch` classes, the Automation conditions and modes, the Data Model `flow-rate` parameter and its units, Fail-Safe Behavior terms, Alarms and Notifications condition codes, and the time zones, random number generator, and program files of the Lighting Programs specification [OAP-LP], and adds no parallel vocabulary.

2.3 Local Control

Programs run on the controller or on the pump itself, with no network service, account, or vendor. A pump that runs its own program keeps its patterns, its schedule, and its place in its group when its controller or the network is down (Section 16.6). Constant flow runs on the controller from a local flow sensor (Section 13). Owners can export every program they run as a file that any conforming product reads (Section 17).

2.4 Safety

Moving water keeps an aquarium alive. It carries oxygen to fish and corals and carbon dioxide away, and reef water at 25 to 28 °C holds only about 6.4 to 6.8 mg/L of oxygen [Garcia], about 2.5 mg/L more than the 4 mg/L at which fish are distressed [FA002]. Flow can also do harm. A wave pattern on a return pump makes the sump level rise and fall, waves at a tank's resonant period grow until water splashes out or an overflow surges, reversing a spinning motor strains it, and strong flow blows food into the overflow and corals off the rock. This Specification limits each of these:

- return pumps run only steady patterns that keep them running, never continue with their last command when their controller is lost, and keep at least the owner's lowest speed (Section 16.1);
- the owner's caps apply to everything a pump runs: its patterns, its triggers, and its transitions (Section 10.2);
- every change of pattern ramps, every pump that sets its speed returns to its program gradually, after a power cut too, and triggered patterns, such as feeding, end on the pump by themselves, even when its controller has failed (Section 12);
- a pump stops, and pauses, before it reverses, and nothing that spares a pump or its relay delays a stop (Sections 10.5 and 16.3);
- swells build their waves gradually, and controllers warn about patterns near a tank's resonance (Section 8.6);
- losing a flow sensor, the clock, the network, or the controller never stops the water moving (Sections 13.4 and 16.4 to 16.6); and
- low flow raises an alarm (Section 13.5).

2.5 Sustainability

Because programs are portable, owners keep their pumps when they change controllers, and keep their programs when they change pumps. A pump's power falls with roughly the cube of its speed where there is little static head, as in the display, and its bearings last longer at lower speeds [HI]; constant flow, night patterns, and the owner's caps let pumps run below their maximum (Sections 10.2, 12.5, and 13). Turnover and measured flow let owners give their livestock the flow it needs rather than the most a pump can give, and show a clogging pump or filter before it fails (Sections 13.5 and 14).

Revision History

Revision	Date	Description	Authors
0.1	2026-10-15	Initial Pre-Draft.	Trent Scholl
0.2	2026-10-16	Triggered patterns layered on a pump that runs its own program end with its schedule when the controller fails, because `runProgram` with `revertAfter` keeps a pending revert's target, and, from Device Classes revision 0.3, ramp when the revert falls due; every pump that restores a speed after power-up ramps up to it; return pumps run only steady patterns that keep them running; the pump members `maxSpeed` and `minSpeed` renamed `speedLimitHigh` and `speedLimitLow`, with the old names still read, and a floor above the cap wins, with a warning; a pump stops, and pauses for its `reversePause`, before it reverses; a switch's `minSwitchTime` never delays a stop from a higher layer; a pump whose program is not loaded never driven with a pattern that stops it while it continues on loss of contact; the flow control's limit of 1 percentage point a second applied to its own corrections only; clocks corrected whenever a binding gives the time, and valid as Fail-Safe Behavior defines; `runProgram` and `programRunning` of Device Classes revision 0.2, with the provisional names kept for pumps of revision 0.1, and its `rampTime`, `flowCurve`, `flowRateSource`, and `clockUncertainty`; `flow-low`, `flow-control-limit`, and `program-not-loaded` raised for each pump, as conditions whose delays are in `conditions.json`; faults that a pump reports not raised again for its function; a pattern	Trent Scholl

Revision	Date	Description	Authors
		type that this revision does not define unsupported rather than invalid; the Data Model's `turnover`, and vessel dimensions proposed for a flow function; resonant periods stored to six digits, so that Table 10 rounds once; the limit of one command a second stated as this Specification's own; a row of Table 16 that no requirement backed removed; the claim about program files swapped on the way limited to the Device Protocol; examples of Device Classes revision 0.2; requirement keywords for every safety check; a test procedure, mapped to its requirements, for every requirement; references cited by tag; one sentence per line; the decided issues resolved, with the limit on return pumps' patterns kept and one conformance level for pumps, and every other issue given a status.	
0.3	2026-10-18	The Project Lead's decisions of 17 October 2026 applied: the function type `flow`, in place of the provisional name, which controllers still read, with storms kept as the extension kind `org.openaquariumproject.storm`; the schema's `$id https://openaquariumproject.org/schemas/flow-programs/0.3`. The decision of 16 October 2026 that Fail-Safe Behavior revision 0.3 and Device Classes revision 0.4 apply, used: a pump ramps to a fail-safe state that runs it, from 0 % after power-up and from the speed it runs at when its controller fails. Also: a network pump in a group sets its clock with the Device Protocol's time exchange, and the clock accuracy of both bindings cited; the Module Bus's program files cited; a triggered pattern sent with `revertAfter` until the end of the mode it follows, when that comes before its `maxDuration`, and a trigger that follows a mode no longer than the mode, with a warning; on pumps of Device Classes revision 0.1 and 0.2, a trigger's revert falls due a transition after the trigger, so that the controller ramps the pump back; a pump's `reversePause` beyond a pattern's stop an exception to determinism, with a warning for gyres that pause for less; latched stops; a pump driven once `program-not-loaded` has lasted its delay; the stops that reach a pump under local control as Automation revision 0.3 lists them; a restore ramp of at least 10 s (provisional), with a warning; power-up ramps limited to pumps that were following their program; why a return pump never continues, now that stops with no known end are latched; the Matter Mapping specification listed as a reference; examples of Device Classes revision 0.5; no requirement keyword in an informative section; "A pass criterion that checks what a SHOULD asks applies to a product that does it"; corrected tests, among them restore ramps of a driven pump with `last`; references to the final revisions of the other specifications; issues 8 to 11, 13, and 15 deferred beyond revision 1.0.	Trent Scholl

PRE-DRAFT

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 **8**
- 5.1 Requirements Language 8
- 5.2 Terms and Definitions 8
- 5.3 Notation 9

6 Introduction **10**
- 6.1 Purpose 10
- 6.2 Design Principles 10
- 6.3 Overview 10
- 6.4 Example 12

7 Common Requirements **12**
- 7.1 Where Programs Run 12
- 7.2 The Flow Function 13
- 7.3 Pumps and Outputs 13
- 7.4 Speed and Direction 14
- 7.5 Local Time and the Clock 14
- 7.6 Holds 14
- 7.7 Faults and Alarms 15
- 7.8 Visibility 15
- 7.9 Determinism 16

8 Patterns **16**
- 8.1 Program Members 16
- 8.2 Pattern Time 17
- 8.3 The Pattern Library 17
- 8.4 Speeds of the Patterns 20
- 8.5 Random Patterns 21
- 8.6 Resonance and Swells 21
- 8.7 Custom Waveforms 22
- 8.8 Safety Checks 22

9 Schedules **23**
- 9.1 Entries 23

10 Pumps **23**
- 10.1 Members of a Pump 23
- 10.2 Scale, Caps, and Floors 24
- 10.3 Computing Speeds 24
- 10.4 Driving Pumps from the Controller 25
- 10.5 Level Outputs and Switches 25
- 10.6 Safety Checks 25

11 Groups **26**
- 11.1 Pumps That Run Together 26
- 11.2 Roles 26
- 11.3 Sequences 26
- 11.4 Timing Between Pumps 26
- 11.5 Safety Checks 27

12 Transitions and Modes **27**

PRE-DRAFT

12.1 Transitions ... 27
12.2 Triggers ... 28
12.3 Restore Ramps ... 29
12.4 Feed ... 29
12.5 Night ... 29
12.6 Storms ... 29
12.7 Battery and Other Modes ... 30
12.8 Safety Checks ... 30

13 Constant Flow ... 30
13.1 Flow Inputs ... 30
13.2 Control ... 31
13.3 An Example Controller (Informative) ... 31
13.4 Lost Flow Inputs ... 31
13.5 Low Flow ... 31
13.6 Safety Checks ... 32

14 Turnover ... 32
14.1 The Turnover Index ... 32
14.2 Flows ... 32
14.3 Showing and Recording Turnover ... 33
14.4 Typical Turnover (Informative) ... 33

15 Pumps That Run Their Own Programs ... 33
15.1 Pump Programs ... 33
15.2 Writing a Pump Program ... 34
15.3 Running a Pump Program ... 34
15.4 Under a Controller ... 35
15.5 Safety Checks ... 36

16 Safety and Failures ... 36
16.1 Return Pumps ... 36
16.2 Fail-Safe Configuration ... 36
16.3 Direction and Speed Changes ... 37
16.4 Lost Inputs ... 37
16.5 Power, Restarts, and the Clock ... 38
16.6 A Failed Controller ... 38
16.7 Safety Checks ... 38

17 Sharing ... 39
17.1 Program Documents ... 39
17.2 Importing and Exporting ... 39
17.3 Licenses, Credit, and Privacy ... 40

18 History ... 40

19 Recommended Values ... 40

20 Conformance ... 41
20.1 Controllers ... 41
20.2 Pumps ... 42
20.3 Documents ... 42
20.4 Schema and Tools ... 42
20.5 Test Procedures ... 42

21 Security Considerations ... 49

Appendix A — Approval Checklist ... 49

Appendix B — Examples (Informative) ... 49

Appendix C — Commercial Features (Informative) ... 50

Appendix D — Open Issues (Informative) ... 51

PRE-DRAFT

3 Scope

This Specification defines:

- flow programs: patterns from a library, each defined by an equation, custom waveforms of up to 25 hours, and a schedule of patterns through the day;
- random patterns that repeat exactly on every product;
- the resonant period of a tank, and swells and pulses that use it;
- groups of pumps that run one program in sync, in anti-sync, or in sequence, and the timing they need;
- constant flow, held from a flow sensor;
- feed, night, storm, and battery behavior, started by Automation conditions and modes;
- a turnover index;
- the programs that pumps run by themselves, and how a controller works with them;
- how flow interacts with fail-safe behavior, and the rules for return pumps;
- 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 flow in, which the Automation specification defines;
- the `pump`, `level-output`, and `switch` classes, which the Device Classes specification defines [OAP-DC];
- the hydraulic design of pumps, plumbing, and overflows;
- how much flow a system needs, beyond the informative ranges of Section 14.4;
- top-off, water changes, and dosing, which their own specifications define, though they use pumps; or
- how alarms are delivered, which the Alarms and Notifications specification defines.

4 References

4.1 Normative References

- **[RFC 2119]** Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- **[RFC 8174]** Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, May 2017.
- **[RFC 8259]** Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, December 2017.
- **[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.
- **[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`.
- **[OAP-DC]** Open Aquarium Project, "Open Aquarium Device Classes", Revision 0.5, `specs/device-classes/spec.md`.
- **[OAP-FS]** Open Aquarium Project, "Open Aquarium Fail-Safe Behavior", Revision 0.4, `specs/fail-safe-behavior/spec.md`.
- **[OAP-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`.

PRE-DRAFT

- **[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`.
- **[OAP-LP]** Open Aquarium Project, "Open Aquarium Lighting Programs", Revision 0.3, `specs/lighting-programs/spec.md`.

4.2 Informative References

- **[OAP-MM]** Open Aquarium Project, "Open Aquarium Matter Mapping", Revision 0.3, `specs/matter-mapping/spec.md`.
- **[SplitMix]** Vigna, S., "splitmix64.c", public domain, https://prng.di.unimi.it/splitmix64.c. The generator of Section 8.5 is this SplitMix64, as the Lighting Programs specification uses it.
- **[Dodge]** Dodge, F. T., "The New 'Dynamic Behavior of Liquids in Moving Containers'", Southwest Research Institute, an update of NASA SP-106, https://www2.swri.org/www2/fluid-slosh/dodge-sp-106-update.pdf. The resonant period of Section 8.6 is that of its first antisymmetric sloshing mode of a rectangular tank.
- **[HI]** Hydraulic Institute, Europump, and US Department of Energy, "Variable Speed Pumping: A Guide to Successful Applications", Executive Summary, https://www.energy.gov/sites/prod/files/2014/05/f16/variable_speed_pumping.pdf.
- **[Garcia]** Garcia, H. E. and Gordon, L. I., "Oxygen solubility in seawater: better fitting equations", Limnology and Oceanography 37(6), 1307–1312, 1992, https://doi.org/10.4319/lo.1992.37.6.1307. Its equations give the oxygen that seawater of salinity 35 holds in Section 2.4.
- **[FA002]** Francis-Floyd, R. and Pouder, D. B., "Dissolved Oxygen for Fish Production", FA002, UF/IFAS, https://ask.ifas.ufl.edu/publication/FA002.
- **[Shashar]** Shashar, N., Cohen, Y., and Loya, Y., "Extreme diel fluctuations of oxygen in diffusive boundary layers surrounding stony corals", Biological Bulletin 185, 455–461, 1993, https://doi.org/10.2307/1542485.
- **[Åström]** Åström, K. J. and Murray, R. M., "Feedback Systems", Chapter 10, "PID Control", https://www.cds.caltech.edu/~murray/books/AM08/pdf/am08-pid_28Sep12.pdf.
- **[Simmonds]** Simmonds, J., et al., "Revisiting the Gage–Bidwell law of dilution in relation to the effectiveness of swimming pool filtration", Water 13, 2350, 2021, https://doi.org/10.3390/w13172350.
- **[Johansen]** Johansen, J. L., "Quantifying water flow within aquatic ecosystems using load cell sensors", PLoS ONE 9(1), e83240, 2014, https://doi.org/10.1371/journal.pone.0083240.
- **[Reidenbach]** Reidenbach, M. A., Koseff, J. R., Monismith, S. G., Steinbuck, J. V., and Genin, A., "The effects of waves and morphology on mass transfer within branched reef corals", Limnology and Oceanography 51(2), 1134–1141, 2006, https://doi.org/10.4319/lo.2006.51.2.1134.
- **[Rodgers]** Rodgers, E. M. and Gomez Isaza, D. F., "The growth-promoting effects of exercise in finfish: a systematic review and meta-analysis", Reviews in Aquaculture 16, 942–953, 2023, https://doi.org/10.1111/raq.12879.

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, demand, layer, and temporary state, and in the Alarms and Notifications specification [OAP-AN], such as condition code, have the same meaning here.

Pump: One output that a flow function drives: an endpoint of the `pump` class, or a `level-output` or `switch` endpoint that controls a pump (Section 7.3).

Return pump: A pump that returns water from a sump to the display, identified by the equipment category `return-pump` (Section 16.1).

Speed: How fast a pump runs, from 0 % (stopped) to 100 % (its maximum), as the `speed` attribute of the `pump` class [OAP-DC Section 9.3].

Direction: Which way a pump with the `reverse` feature pumps: forward, or in reverse.

Program: What a group of pumps runs through the day: its patterns and its schedule (Section 8).

Pattern: A way of changing a pump's speed, and its direction, over time, such as a pulse or a gyre (Section 8.3).

Library: The patterns that this Specification defines (Table 8).

Custom waveform: A pattern given as points of speed over a period (Section 8.7).

Period: The time after which a pattern repeats.

Phase period: The period by which the roles of a group shift a pattern (Section 11.2).

Pattern time: Local time, counted in seconds from 1970-01-01 00:00 local time, from which every pump computes its patterns (Section 8.2).

Edge: A moment at which a pattern's speed or direction changes at once, such as the start of a pulse without a ramp.

Schedule: The patterns that a program runs at each time of day (Section 9).

Entry: One item of a schedule: a pattern, and the time of day at which it starts.

Group: The pumps of one flow function, which run one program together (Section 11).

Role: How a pump runs the program with the rest of its group: in sync, in anti-sync, or in sequence (Section 11.2).

Transition: The change from one pattern to another, over a set time (Section 12.1).

Trigger: A condition that makes a flow function run another pattern of its program while the condition holds, for feeding, the night, a storm, or a power cut (Section 12.2).

Restore ramp: The ramp with which a pump returns to its program after a power-up, a stop, or a change of program (Section 12.3).

Resonant period: The period of the slowest standing wave that the water in a tank can hold along one of its sides (Section 8.6).

Flow input: An Automation input that measures the flow of a pump (Section 13.1).

Constant flow: Control of a pump's speed so that a measured flow stays at a target (Section 13).

Turnover: How many times an hour pumps move the volume of the water they serve (Section 14).

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

Pump program: A program as a pump runs it by itself, with that pump's settings, written to its `program` attribute (Section 15.1).

Program document: A file of programs that owners and products share (Section 17).

5.3 Notation

JSON member names, function types, pattern types, condition codes, and the names of attributes, commands, and faults appear in `monospace`. Durations and periods are in seconds, and speeds in %. 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 8.3.2" means item 2 of Section 8.3. Numbers marked provisional are to be confirmed with the breadboard controller, prototype pumps, and owners' experience (Appendix D). In

PRE-DRAFT

equations, cos takes degrees, and "*x* mod *m*" gives a result from 0 up to, but not including, *m*, also when *x* is negative.

6 Introduction

6.1 Purpose

The wave pumps and controllers on the market run flow patterns, and their patterns are much alike: constant flow, pulses, random flow, sine waves, gyres, pumps in sync and in anti-sync, feeding and night modes, and a wave tuned to the tank. Yet no two products define them the same way. One maker's pulse duration is the time on, another's the time on and off; one maker's anti-sync runs the second pump at the opposite speed, another's runs it half a period later; none publishes its random patterns. A program cannot move from one product to another, and pumps from two makers cannot run in step unless a controller drives both, which cannot time a pulse of a second over a network.

This Specification defines each pattern by an equation of local time. Every pump computes its pattern from the same clock, so pumps from different makers stay in step without messages between them, and the same program means the same thing on every product. Programs can be shared, and moved between controllers and pumps, with the owner's caps, the rules for return pumps, and fail-safe behavior applied the same way everywhere.

6.2 Design Principles

1. **Patterns, not products.** Programs name patterns from a library, each with an equation, and give speeds as shares of whichever pump runs them (Section 8).
2. **The clock is the conductor.** Every pump computes its pattern from pattern time, so a group stays in step without messages, and anti-sync and sequences are offsets of the same clock (Section 11).
3. **The water keeps moving.** Losing a sensor, the clock, the network, or the controller never stops circulation, and return pumps never stop because a program asked them to (Section 16).
4. **The owner's limits are the most.** Caps apply to everything a pump runs (Section 10.2).
5. **Nothing sudden.** Patterns change with transitions, pumps return to their programs with ramps, pumps stop before they reverse, and swells build gradually (Sections 8.6, 12, and 16.3).
6. **Fast patterns run on the pump.** A controller drives only what it can time; faster patterns run on pumps that run their own programs (Section 10.4).

6.3 Overview

Figure 1 shows how a pump's speed is made from its program and its settings.

PRE-DRAFT

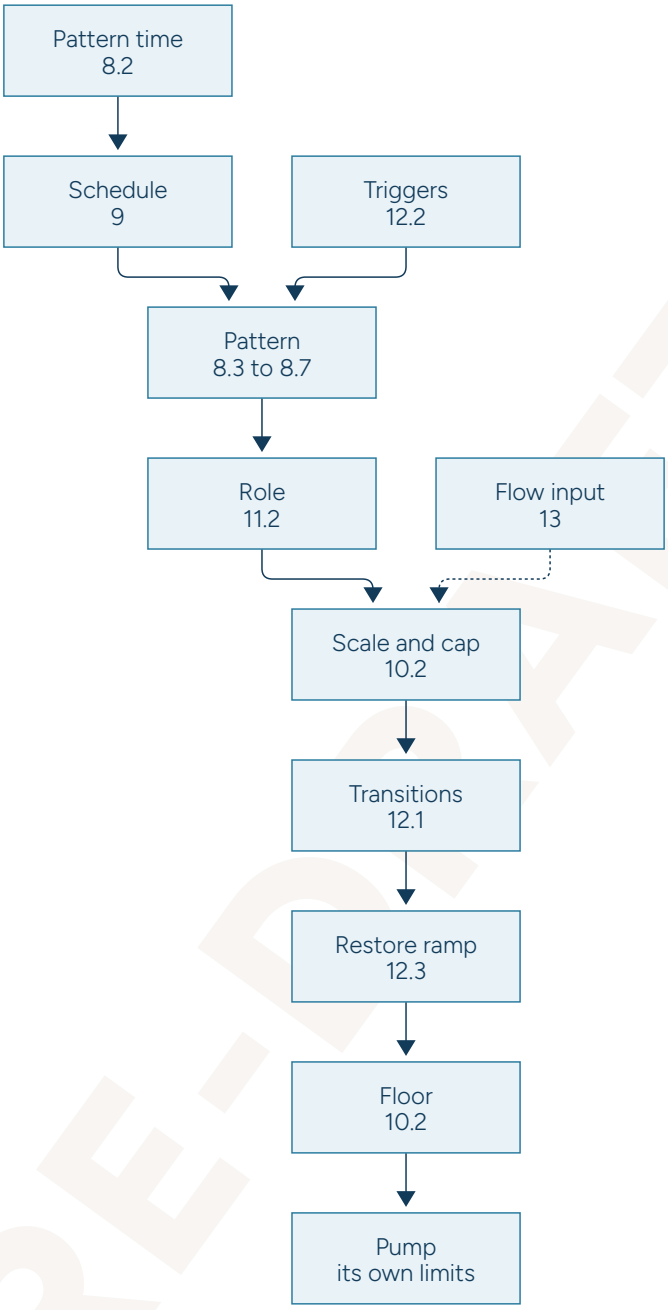

Figure 1 — How a pump's speed is made

Table 1 — Parts of this Specification

Part	What it does	Section
Common requirements	Where programs run, the flow function, the outputs it drives, and how flow fits Automation.	7
Patterns	Pattern time, the library, random patterns, resonance, and custom waveforms.	8
Schedules	The patterns of each time of day.	9
Pumps	Each pump's settings, how its speed is computed, and what a controller can drive.	10
Groups	Pumps in sync, in anti-sync, and in sequence, and the timing between them.	11
Transitions and modes	Changes between patterns; feed, night, storm, and battery behavior.	12
Constant flow	Holding a measured flow.	13
Turnover	The turnover index.	14
Pump programs	Pumps that run their own programs, under a controller or without one.	15

PRE-DRAFT

Part	What it does	Section
Safety and failures	Return pumps, fail-safe behavior, reversing, lost inputs, power, and the clock.	16
Sharing	Program documents, importing, licenses, and privacy.	17

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

6.4 Example

An owner in Brisbane keeps a mixed reef with a 400 L display, 120 cm long, with water 50 cm deep, and a 100 L sump. Two wave pumps from different makers sit on the side walls, a back-wall pump is dimmed by a 0–10 V output, and the return pump has a flow sensor in its line. The owner imports a program that another reefkeeper wrote, "Reef day" `examples/reef.json`.

1. The program suggests two pumps, one in sync and one in anti-sync. The controller proposes these roles, and the owner scales the stronger right-hand pump to 85 % (Sections 10.2, 11.2, and 17.2).
2. The program's nutrient export pulses at the tank's resonant period. The controller asks for the tank's length and water depth, and computes 1.334 s (Section 8.6).
3. Both wave pumps run the program by themselves, from pump programs that the controller writes to them (Section 15).
4. At 11:30 the schedule changes from reef-crest random flow to a midday pulse, over 10 s. The left pump's pulses start on the minute and every 5 s after it; the right pump's, in anti-sync, 2.5 s after each, within 50 ms, with no message between them (Section 11.4).
5. At 18:00 the owner presses Feed. The feed mode stops the return pump for 10 minutes, and the flow function's trigger sends both wave pumps their feeding pattern, a steady 10 %, for the same 10 minutes, each command with `revertAfter`, so the pumps return to their schedule even if the controller fails (Sections 12.2 and 12.4).
6. The return pump holds 2400 L/h from its flow sensor. Over the weeks its prefilter clogs, and its speed creeps up from 62 %; when it has run at its highest allowed speed, 85 %, for 5 minutes without reaching 2400 L/h, the controller raises `flow-control-limit`, and the owner changes the prefilter (Section 13).
7. From an hour after sunset, a night condition turns the return pump down to 1800 L/h, while the wave pumps' own schedule turns to gentle random flow at 21:30 (Sections 9 and 12.5).
8. The controller shows a display turnover of about 41 times an hour in the morning, marked as estimated, because the wave pumps' flows are estimated from their speeds, and a return turnover of 4.8 times an hour, from the flow sensor (Section 14).
9. One afternoon the owner starts the storm mode: for 20 minutes both wave pumps run fast random flow within their own caps, and then return to their schedule over 20 s (Section 12.6).
10. Overnight the Wi-Fi router fails. Both wave pumps keep running their programs by their own clocks, which drift apart by less than a second through the night, and are in step again within seconds of the router's return; the return pump, on the wired module bus, keeps its flow (Section 16.6).

The configuration is `examples/reef-system.json`.

7 Common Requirements

7.1 Where Programs Run

1. A program runs either on a controller, which sends each pump its speeds, or on a pump with the `wave` feature, which runs a pump program written to its `program` attribute [OAP-DC Section 9.3]. A pump member's `runOn` says which (Table 13).
2. The same rules apply to both, so a pump that runs a program by itself does what a controller would have commanded (Sections 7.9 and 15.3).
3. Only `pump` endpoints run programs by themselves; a `level-output` or `switch` output is always driven by its controller.

4. A controller MUST run flow functions in its control process, as Automation Section 17.1.3 requires, so that if they stop, the pumps apply their fail-safe behavior.
5. A controller SHOULD let pumps with the `wave` feature run their own programs, so that they keep their patterns when it is gone (Section 16.6). Patterns faster than a controller can drive run only on such pumps (Section 10.4).

7.2 The Flow Function

1. A flow function is a function of the type `flow`, which the Automation specification hosts [OAP-AU Section 11.10]. It keeps no saved state: its triggers follow their conditions after a restart (Section 12.2.4).
2. A flow function runs one program on a group of pumps (Section 11). It has the members of Table 2, and each of its pumps the members of Table 13.

Table 2 — Members of a flow function

Member	Type	Presence	Description
`id`, `type`, `name`, `notes`, `preset`			As for every function [OAP-AU Section 11.1]. `preset` is `circulation-pump` or `return-pump`.
`outputs`	array	REQUIRED	The output of every pump.
`program`	object	REQUIRED	The program the pumps run (Section 8).
`pumps`	array	REQUIRED	The pumps, one for each output, 1 to 16 (Section 10.1).
`triggers`	array	OPTIONAL	Conditions that run other patterns of the program, up to 32 (Section 12.2).
`restoreRamp`	number (s)	OPTIONAL	The restore ramp, from 0 to 600 s. Default 30 s (Section 12.3).
`dimensions`	object	OPTIONAL	The inside `length` and `width` of the vessel the pumps are in, and the `depth` of its water, in cm (Section 8.6).
`extensions`	object	OPTIONAL	Extensions, as in the Automation specification [OAP-AU Section 7.7].

3. `outputs` MUST list the `output` of every pump, and nothing else, so that a controller that does not support flow functions can still find their outputs, and treat them as Automation Section 19.2 requires for an unsupported function.
4. A flow function defines its own behavior when an input is unavailable (Section 16.4), so it MUST NOT have `onUnavailable`.
5. A member described as a condition names a condition of the configuration, or an input that is true or false [OAP-AU Section 10].
6. A configuration document that contains flow functions MUST be valid against the Automation schema, and each flow function MUST be valid against `$defs/flow` in `schema/flow-programs.schema.json`, a JSON Schema [JSON Schema].
7. A configuration MAY have several flow functions, for example one for the display's wave pumps, one for a back-wall pump, and one for the return pump. An output belongs to at most one function [OAP-AU Section 11.1].
8. The pumps of a flow function MUST all be return pumps, or none of them, because return pumps run only steady patterns (Section 16.1).
9. Revisions 0.1 and 0.2 named the function type `org.openaquariumproject.flow`. A controller reads it as this revision's type [OAP-AU Section 7.8], and writes `flow`.

7.3 Pumps and Outputs

Table 3 — Outputs a flow function drives

Class	How it runs a program	Notes
`pump`	Its speed, and its direction with the `reverse` feature, through `set` with `running`, `speed`, and `direction`; or its own program, with the `wave` feature (Section 15).	The usual class for DC wave pumps, cross-flow pumps, and return pumps.
`level-output`	Its speed as `level`, through `set` (Section 10.5).	A pump with a 0–10 V or PWM speed input. Driven by the controller only.

Class	How it runs a program	Notes
`switch`	On while its speed is above `threshold`, through `set` with `on` (Section 10.5).	A pump that can only be switched. Driven by the controller only, no more often than its `minSwitchTime`.

1. Each pump member names one output of a class in Table 3, or of an extension class.
2. A pattern with `reverse true`, and a gyre, run a pump in reverse. A controller MUST run them only on a `pump` endpoint with the `reverse` feature, and MUST tell the owner why it will not run them on another.
3. The equipment category of a pump's output SHOULD be `circulation-pump` or `return-pump` [OAP-DM Section 9.7].

7.4 Speed and Direction

1. A speed is a percentage of a pump's maximum, as the `speed` attribute of the `pump` class. A speed of 0 % means the pump is stopped: `running` is `false`.
2. A program gives speeds as shares of whichever pump runs it, so the same speed gives different flow on different pumps. A pump member's `scale` suits a program to a stronger or weaker pump, and its `fullFlow` relates speed to flow (Sections 10.2 and 14.2).
3. A pump SHOULD document its flow at each speed with no head, and its reverse flow, as its `flowCurve` [OAP-DC Section 9.3], so that owners can estimate its turnover.
4. A speed above 0 % and below a pump's lowest speed is raised to that speed (Section 10.2).
5. Speeds are numbers, and are not rounded by this Specification. A pump runs them as finely as its driver allows.
6. A pump's forward direction is the one its maker documents. A pattern runs forward unless it says otherwise.

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 pump program carries the time zone as `posixTimeZone`, a POSIX `TZ` string [POSIX], because a pump has no other way to know it (Section 15.1).
3. A pump that runs its own program MUST implement POSIX `TZ` strings as Lighting Programs Section 7.5.3 requires of lights.
4. Durations, such as transitions and ramps, are elapsed time.
5. A pump'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 pump started, or kept by a battery-backed clock, and cannot have drifted by more than 60 s since. It stays valid when the pump loses contact with its controller, and when a binding reports the pump's timestamps as unknown, until it could have drifted that far [OAP-FS Section 10.5.5], so a pump keeps its program when its controller fails.
6. While a controller's clock is not valid [OAP-FS Section 10.5.1], or a pump's, the water keeps moving on elapsed time (Section 16.5).

7.6 Holds

1. Interlocks, modes, overrides, and step programs act on pumps as on any other output, in the order of Automation Table 25. A flow function is the pump's control function, at layer 10.
2. The flow function's demand on a pump is its program: the speeds of Section 10.3 for a pump the controller drives, and the pump's own program for a pump that runs one.
3. A higher layer can stop a pump, run it at its `runState`, or set it to a level, which is a speed [OAP-AU Table 5]. A higher layer's demand replaces the program for as long as it lasts, and the flow function then returns the pump to its program with the restore ramp (Section 12.3). A stop with no known end that an interlock or an emergency mode demands, such as a return pump's stop while a leak sensor is wet, is latched [OAP-AU Section 17.5.4]: the pump keeps it, and its program starts nothing, even when its controller fails, until a controller ends the stop [OAP-DC Section 7.4.13].

PRE-DRAFT

4. Triggers are part of the flow function's demand, so a mode, override, or interlock that stops a pump also stops its triggered pattern.
5. The Automation presets stop circulation pumps in feed, maintenance, and water-change modes, leave them to their function in emergency mode, and run them on battery [OAP-AU Table 29]. A mode that should slow pumps, or change their pattern, rather than stop them, leaves them out of its `outputs`, and the flow function follows the mode with a trigger (Section 12.2).
6. A mode that a trigger follows acts on the pumps 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 Faults and Alarms

1. A controller raises the faults that a flow function detects, which Table 4 lists, with the function as their subject [OAP-AN Section 8.2], naming the pump. `program-not-loaded`, `flow-low`, and `flow-control-limit` concern one pump, so their subject is the pump's output, as for `command-refused` [OAP-AU Section 17.9], and each pump has its own alarm [OAP-AN Section 8.3].
2. Faults that a pump reports itself, such as `stall`, `dry-run`, `overcurrent`, `overtemperature`, and `communication-lost` (Section 16.6), raise alarms with the pump'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 7.8).
3. Table 5 gives the codes that this Specification adds, which are listed with their default severities and their delays in `conditions.json`. 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 flow functions raise

Code	Defined by	When	Subject	Latches
`input-unavailable`	Automation	A flow input or a trigger's condition is unknown (Sections 13.4 and 12.2).	The function	No
`command-refused`	Automation	A pump refused a command, such as a pattern it cannot run (Section 15.4).	The pump's output	No
`unsupported`	Automation	The controller does not support flow functions, or a member of one, such as a pattern's type (Section 20.1).	The function	No
`program-not-loaded`	Lighting Programs	A pump does not report the pump program its controller wrote (Section 15.2).	The pump's output	No
`flow-low`	This Specification	A running pump's measured flow is below its `minFlow` (Section 13.5).	The pump's output	No
`flow-control-limit`	This Specification	Constant flow holds its pump at a limit without reaching its target (Section 13.5).	The pump's output	No

Table 5 — Conditions this Specification adds

Code	Meaning	Default	Critical for	Notes
`flow-low`	The measured flow of a pump that runs at its program's speed is below the `minFlow` its owner set (Section 13.5).	`warning`	`vital`	Raised after 60 s. A clogged pump or prefilter, an air lock, a closed valve, or a low sump. A return pump with low flow starves the equipment it feeds.
`flow-control-limit`	A constant-flow pattern holds its pump at its highest or lowest speed without bringing the flow to its target (Section 13.5).	`warning`	—	Raised after 300 s. Usually a filter or pump that is clogging. The pump keeps running at its limit.

7.8 Visibility

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

1. its program's name, and, for each pump, its role, and whether it runs the program itself or the controller drives it;
2. the pattern each pump runs now, and what decided it: an entry of the schedule, a trigger, or a higher layer, such as "feeding pattern, 6 minutes left" or "stopped by maintenance mode";

3. each pump's speed and direction now, and what holds it below its pattern, if anything: its cap, a restore ramp, or a higher layer;
4. for a pump under constant flow, its target, its measured flow, its speed, and whether it is at a limit (Section 13);
5. the turnover indices of its system, marked as estimated where they are (Section 14.3);
6. the pumps it cannot drive with their program, or that cannot run a pattern of it, and why (Sections 7.3 and 10.4); and
7. the faults its pumps report, such as `stall` or `dry-run` (Section 7.7.2).

7.9 Determinism

1. Given the same program, pump settings, triggers, and clock, two conforming products MUST compute the same speed and direction for each pump at every moment, within 0.5 percentage points of speed (provisional).
2. The exceptions are moments within 50 ms of an edge, of the start of an entry, or of the start or end of a trigger, where a change may be taken a little earlier or later; the speeds of constant flow, which depend on measurements (Section 13); and the time a reversible pump stays stopped for its `reversePause` beyond the stop that a pattern or a transition gives it (Section 16.3).
3. A pump that runs its own program MUST meet item 1 against the reference implementation, so that a pump and its controller agree (Section 15.3).

This makes programs portable: the same program does the same on every product, and pumps from two makers in one tank stay in step. What a pump's motor actually does lags its commanded speed by the time the motor takes to change speed; this Specification defines the commanded speed.

8 Patterns

8.1 Program Members

1. A program has the members of Table 6, and MUST be valid against `$defs/program` in `schema/flow-programs.schema.json`.
2. Pattern identifiers are unique within a program. Every entry of the schedule, and every trigger, names a pattern of the program.
3. Members whose values are their defaults MAY be left out.
4. A pattern MAY have a `purpose`: `day`, `feed`, `night`, `storm`, `maintenance`, `battery`, `other`, or an extension name. Controllers use it to propose triggers (Section 12.2), to find a pattern for a one-touch action, and to check night patterns (Section 12.5).

Table 6 — Members of a program

Member	Type	Presence	Description
`id`	identifier	REQUIRED	Identifies the program, as an Automation identifier [OAP-AU Section 7.3].
`name`	string	REQUIRED	Name shown to the owner.
`description`	string	OPTIONAL	What the program is for, and what it suits, including where its pumps should be placed.
`author`	string	OPTIONAL	Who wrote the program, as they wish to be named.
`license`	string	OPTIONAL	An SPDX license expression [SPDX], under which others may use the program, such as `CC0-1.0` (Section 17.3).
`revision`	string	OPTIONAL	The author's revision of the program.
`created`	timestamp	OPTIONAL	When the program was written.
`water`	array	OPTIONAL	The Data Model water types the program suits: `marine`, `brackish`, `freshwater`.
`tags`	array	OPTIONAL	Data Model system tags the program suits, such as `reef` or `planted` [OAP-DM Section 9.7].
`seed`	integer	OPTIONAL	The seed of the program's random patterns, from 0 to 4294967295 (Section 8.5). Default 0.

PRE-DRAFT

Member	Type	Presence	Description
`roles`	array	OPTIONAL	The roles the program's author meant its pumps to have, one for each pump, up to 16, such as `["sync", "anti-sync"]` (Section 11.2).
`patterns`	array	REQUIRED	The patterns, 1 to 32 (Section 8.3).
`schedule`	array	REQUIRED	The entries of the schedule, 1 to 24 (Section 9).
`extensions`	object	OPTIONAL	Extensions, as in the Data Model [OAP-DM Section 7.6].

8.2 Pattern Time

1. Every pattern is a function of pattern time T: the local time at the moment, counted in seconds from 1970-01-01 00:00 local time. For a moment u, given as POSIX time in seconds, and the local UTC offset $z(u)$ at that moment, in seconds, $T = u + z(u)$.
2. Patterns repeat, so what a pump does depends on T modulo the pattern's period. Every pump of a group computes the same T from its clock, so they agree without messages (Section 11).
3. When the clocks change for daylight saving time, T jumps by the change, and every pump's pattern jumps with it, together. So a custom waveform of a day keeps to local time, and the hour that occurs twice when the clocks go back runs the same patterns twice.
4. Implementations SHOULD compute T and its remainders in whole milliseconds, as 64-bit integers, or in binary64 floating point. T is near 1.8×10^9, and binary32 floating point cannot hold it to a second.

For example, at 12:00 on 14 October 2026 in Brisbane, at UTC+10, u is 1 791 943 200 and T is 1 791 979 200. `test-vectors.json` gives further pattern times, across changes of daylight saving time.

8.3 The Pattern Library

1. Each pattern MUST be valid against `$defs/pattern` in `schema/flow-programs.schema.json`, and has the members of Table 7, and those of Table 9 that Table 8 gives for its type. A later revision may add pattern types, so a pattern of another type is valid, with whatever members its type has, but unsupported by a product that does not implement that type. A product MUST NOT run a program with a pattern of a type it does not implement: a controller treats its flow function as unsupported (Section 20.1.4), and a pump rejects it as a program it cannot run in full (Section 15.1.4).
2. Rules that the schema cannot express:
   - `min` is not above `max`;
   - for a `pulse`, `pulseWidth` is at least `ramp`, and `pulseWidth` plus `ramp` is at most `period`;
   - for a `square`, `ramp` is at most half the `period`;
   - for a `random`, `ramp` is at most `period`;
   - for a `gyre`, twice `ramp`, plus `pause`, is at most half the `period`;
   - for a `nutrient-export`, `surge` is less than `period`;
   - for a `custom`, the times of the points increase, and are less than `period` (Section 8.7); and
   - for a `constant-flow`, `fallback` is from `min` to `max`.
3. Periods, pulse widths, and ramps are numbers of seconds, and need not be whole.
4. "Period" always means one whole repetition: a pulse's time on and its time off together. Commercial products count differently (Appendix C).

Table 7 — Members of every pattern

Member	Type	Presence	Description
`id`	identifier	REQUIRED	Identifies the pattern within the program.
`type`	string	REQUIRED	The pattern's type: one of Table 8.
`name`	string	OPTIONAL	Name shown to the owner.
`purpose`	string	OPTIONAL	What the pattern is for (Section 8.1.4).
`reverse`	boolean	OPTIONAL	Whether the pump runs the pattern in reverse. For a `gyre`, whether it starts in reverse. Default `false`. Needs the `reverse` feature (Section 7.3).
`extensions`	object	OPTIONAL	Extensions.

Table 8 — The pattern library

Pattern	What it does	Members	Roles shift	In anti-sync	A controller drives it when
`constant`	One steady speed.	`speed`	—	—	Always
`pulse`	`max` for `pulseWidth` at the start of each `period`, and `min` for the rest of it, with a ramp of `ramp` at each change.	`max`, `min`, `period`, `pulseWidth`, `ramp`	`period`	Half its phase period later	`pulseWidth` at least 2 s; the rest of the `period` at least 2 s
`square`	A pulse that lasts half the `period`: `max` and `min` in turn, for half the period each.	`max`, `min`, `period`, `ramp`	`period`	Half its phase period later	`period` at least 4 s
`sine`	Rises smoothly from `min` to `max` and falls back over each `period`.	`max`, `min`, `period`	`period`	Half its phase period later	`period` at least 30 s
`triangle`	Rises steadily from `min` to `max` over the first half of each `period`, and falls steadily back over the second.	`max`, `min`, `period`	`period`	Mirrored	`period` at least 10 s
`random`	A new speed from `min` to `max` every `period`, drawn by the generator of Section 8.5, and reached with a smooth change over `ramp`.	`max`, `min`, `period`, `ramp`, `seed`	`period`	Mirrored	`period` at least 5 s
`gyre`	Forward at `max` for the first half of each `period`, and in reverse at `reverseMax` for the second, with a ramp of `ramp` at each change and a stop of `pause` before each reversal. Needs the `reverse` feature.	`max`, `reverseMax`, `period`, `ramp`, `pause`	`period`	Half its phase period later	`period` at least 20 s
`swell`	Waves of `wavePeriod`, from `min` towards `max`, whose height rises and falls over each `period`. With `wavePeriod resonance`, the waves match the tank's resonant period (Section 8.6).	`max`, `min`, `period`, `wavePeriod`	`wavePeriod`	Half its phase period later	`wavePeriod` at least 30 s
`nutrient-export`	Pulses of `pulsePeriod` whose strength grows from `min` towards `max` over each `period`, less its `surge`, and then a surge at `max` for `surge`. With `pulsePeriod resonance`, the pulses match the tank's resonant period (Section 8.6).	`max`, `min`, `period`, `pulsePeriod`, `surge`	`pulsePeriod`	Half its phase period later	`pulsePeriod` at least 4 s
`custom`	Points of speed over a `period` of up to 25 hours, joined by `interpolation` (Section 8.7).	`period`, `points`, `interpolation`	`period`	Mirrored	Points at least 2 s apart
`constant-flow`	The speed that keeps a flow input at `flowRate`, from `min` to `max` (Section 13).	`flowRate`, `unit`, `max`, `min`, `tolerance`, `fallback`	—	—	Always; only a controller can run it

Table 9 — Members of the patterns

Member	Patterns	Type	Presence	Range	Default	Description
`speed`	`constant`	number (%)	REQUIRED	0 to 100	—	The speed.
`max`	`pulse`, `square`, `sine`, `triangle`, `random`, `gyre`, `swell`, `nutrient-export`, `constant-flow`	number (%)	REQUIRED, except for `constant-flow`	above 0 to 100	100 for `constant-flow`	The highest speed of the pattern, or, for `constant-flow`, the highest speed its control may set.
`min`	`pulse`, `square`, `sine`, `triangle`,	number (%)	OPTIONAL	0 to 100	0	The lowest speed of the pattern, or,

PRE-DRAFT

Member	Patterns	Type	Presence	Range	Default	Description
	`random`, `swell`, `nutrient-export`, `constant-flow`					for `constant-flow`, the lowest speed its control may set. 0 stops the pump.
`period`	`pulse`, `square`, `sine`, `triangle`, `random`, `gyre`, `swell`, `nutrient-export`, `custom`	number (s)	REQUIRED	0.5 to 3600 for `pulse`, `square`, `random`; 1 to 86400 for `sine`, `triangle`; 10 to 86400 for `gyre`; 30 to 86400 for `swell`; 60 to 86400 for `nutrient-export`; 1 to 90000 for `custom`	—	The time of one repetition. For `random`, the time between new speeds; for `swell` and `nutrient-export`, the time over which the waves or pulses rise and fall, and repeat.
`pulseWidth`	`pulse`	number (s)	REQUIRED	0.2 to 3600	—	How long each pulse lasts, from the start of its rise to the start of its fall. At most `period` less `ramp`, and at least `ramp`.
`ramp`	`pulse`, `square`, `random`, `gyre`	number (s)	OPTIONAL	0 to 600 for `pulse`, `square`, `gyre`; 0 to 3600 for `random`	0 for `pulse` and `square`; `period` for `random`; 3 for `gyre`	How long each change of speed takes.
`reverseMax`	`gyre`	number (%)	OPTIONAL	above 0 to 100	`max`	The speed of a gyre's reverse half. Many pumps move less water in reverse, so owners often set it higher than `max`.
`pause`	`gyre`	number (s)	OPTIONAL	1 to 600	2	How long a gyre stays stopped before each reversal.
`wavePeriod`	`swell`	number (s), or `resonance`	REQUIRED	0.5 to 60, or `resonance`	—	The period of a swell's waves, or `resonance` for the tank's resonant period along its length (Section 8.6).
`pulsePeriod`	`nutrient-export`	number (s), or `resonance`	REQUIRED	0.5 to 60, or `resonance`	—	The period of a nutrient export's pulses, on for the first half and off for the second, or `resonance` for the tank's resonant period along its length (Section 8.6).
`surge`	`nutrient-export`	number (s)	OPTIONAL	0 to 3600	60	How long the surge at the end of each `period` of a nutrient export lasts. Less than `period`.

PRE-DRAFT

Member	Patterns	Type	Presence	Range	Default	Description
`points`	`custom`	array	REQUIRED	2 to 1440 items	—	2 to 1440 points, each a `time`, in seconds from the start of the period, and a `speed` (Section 8.7).
`interpolation`	`custom`	string	OPTIONAL	`linear`, `cosine`, `step`	`linear`	`linear`, `cosine`, or `step`: how the speed changes between points (Section 8.7).
`seed`	`random`	integer	OPTIONAL	0 to 4294967295	The program's `seed`	The seed of the generator, from 0 to 4294967295 (Section 8.5).
`flowRate`	`constant-flow`	number	REQUIRED	above 0	—	The flow to hold, in `unit`.
`unit`	`constant-flow`	string	OPTIONAL	`L/h`, `L/min`, `gal/h`	`L/h`	A Data Model unit of flow: `L/h`, `L/min`, or `gal/h`.
`tolerance`	`constant-flow`	number (%)	OPTIONAL	1 to 50	5	How far, as a share of `flowRate`, the measured flow may be from it before the speed changes (Section 13.2).
`fallback`	`constant-flow`	number (%)	OPTIONAL	0 to 100	`max`	The speed used while the flow input is unavailable and no recent speed is known (Section 13.4).

`patterns.json` holds Tables 8 and 9 in machine-readable form, and the schema holds the ranges.

8.4 Speeds of the Patterns

For a pump at pattern time T, with its offset o from its role (Section 11.2), let $x = T - o$ and $p = x$ mod `period`. A product MUST compute each pattern's speed and direction as this section says. Each pattern's weight w, from 0 to 1, gives its speed as `min` + (`max` − `min`) × w:

- **`constant`:** the speed is `speed`.
- **`pulse`:** with W = `pulseWidth` and r = `ramp`, w is p / r while $p < r$; 1 while $p < W$; $1 - (p - W) / r$ while $p < W + r$; and 0 otherwise. With $r = 0$, w is 1 while $p < W$, and 0 otherwise.
- **`square`:** as `pulse`, with W = `period` / 2.
- **`sine`:** $w = (1 - \cos(360° \times p / \texttt{period})) / 2$, which starts at `min`.
- **`triangle`:** $w = 2 \times p / \texttt{period}$ while $p <$ `period` / 2, and $2 - 2 \times p / \texttt{period}$ otherwise.
- **`random`:** as Section 8.5.
- **`gyre`:** with H = `period` / 2, $L = H -$ `pause`, r = `ramp`, and $y = p$ mod H, w is y / r while $y < r$; 1 while $y < L - r$; $(L - y) / r$ while $y < L$; and 0 from L to H. The speed is `max` × w while $p < H$, forward, and `reverseMax` × w while $p \geq H$, in reverse; `reverse true` swaps the two directions. So a gyre ramps up, runs, ramps down, and stops for `pause` before every change of direction.
- **`swell`:** with the envelope $E = (1 - \cos(360° \times (T \bmod \texttt{period}) / \texttt{period})) / 2$ and the wave $V = (1 - \cos(360° \times (x \bmod \texttt{wavePeriod}) / \texttt{wavePeriod})) / 2$, $w = E \times V$. The envelope follows T, not x, so every pump of a group swells together, while roles shift its waves.

- **`nutrient-export`:** with $e = T \bmod$ `period` and $S =$ `surge`: while $e \geq$ `period` $- S$, the speed is `max`, the surge; otherwise $w = e / ($`period` $- S)$ while $(x \bmod$ `pulsePeriod`$) <$ `pulsePeriod` $/ 2$, and 0 otherwise. So the pulses grow from `min` towards `max` through each period, and the period ends with every pump of the group surging together, which sweeps lifted detritus to the overflow.
- **`custom`:** as Section 8.7.
- **`constant-flow`:** as Section 13.

For example, a `pulse` with `min` 30, `max` 100, `period` 5, `pulseWidth` 1.5, and `ramp` 0.2 runs at 65 % at $p = 0.1$, at 100 % from $p = 0.2$ to 1.5, at 65 % at $p = 1.6$, and at 30 % from $p = 1.7$ to 5. A `gyre` with `max` 60, `reverseMax` 80, `period` 1200, `ramp` 10, and `pause` 3 ramps up forward for 10 s, runs at 60 % until 587 s, ramps down to a stop at 597 s, rests 3 s, and then does the same in reverse at 80 %.

8.5 Random Patterns

The generator is SplitMix64 [SplitMix], as Lighting Programs Section 11.2 defines it, with all arithmetic on unsigned 64-bit integers, modulo 2^{64}.

1. A random pattern's seed S is its `seed`, or the program's `seed`. Its steps are numbered $k = \text{floor}(x /$ `period`$)$, so that step k starts at pattern time $k \times$ `period` $+ o$.
2. The speed of step k MUST be $v(k) =$ `min` $+ ($`max` $-$ `min`$) \times (d \bmod 1001) / 1000$, where d is the first draw of the generator started at $x_0 = ((S \bmod 2^{32}) \times 2^{32} + (k \bmod 2^{32}))$ XOR $((4 \times \text{0x9E3779B97F4A7C15}) \bmod 2^{64})$, with a draw as Lighting Programs Section 11.2.2 defines it. The stream number 4 keeps random flow apart from the clouds, storms, and lightning of a lighting program with the same seed, which use the streams 1 to 3.
3. With $f = x - k \times$ `period`, the speed is $v(k - 1) + (v(k) - v(k - 1)) \times (1 - \cos(180° \times f /$ `ramp`$)) / 2$ while $f <$ `ramp`, and $v(k)$ after. So the speed glides to each new value along a cosine, and with `ramp` equal to `period`, the default, it never rests.
4. Each step's speed comes from its number alone, so any product can compute any moment without computing the moments before it, and every pump of a group draws the same speeds.

For example, a random pattern with seed 4207, `min` 40, `max` 80, and a `period` of 6 s, at 12:00 on 14 October 2026 in Brisbane (pattern time 1 791 979 200), is at the start of step 298 663 200. Its generator starts at 0x78DDF68AECE4CD74, and its first draw is 8810609125443665205, which is 666 modulo 1001, so the step's speed is 66.64 %. The step before drew 44.08 %, so the pump runs at 44.08 % at 12:00:00, 55.36 % at 12:00:03, and 66.64 % at 12:00:06. `test-vectors.json` gives further draws and speeds.

8.6 Resonance and Swells

Water in a tank sloshes most readily at its resonant periods. For water of depth h along a side of length L, the slowest standing wave, with the water rising at one end while it falls at the other, has the period $T_1 = 2\pi / \sqrt{(g \times k \times \tanh(k \times h))}$ seconds, where $k = \pi / L$, $g = 9.80665$ m/s², and L and h are in metres [Dodge]. The shallow-water estimate $2L / \sqrt{(g\,h)}$ is too short for aquariums, whose water is deep for their length: by about 7 % for a long, shallow tank, and by nearly 40 % for a short, deep one (Table 10).

Table 10 — Resonant periods of some tanks

Water length (cm)	Water depth (cm)	Resonant period (s)	Shallow-water estimate (s)
45	25	0.783	0.575
60	30	0.916	0.700
60	40	0.890	0.606
60	50	0.882	0.542
90	45	1.121	0.857
120	50	1.334	1.084
120	60	1.295	0.989
150	55	1.533	1.292
180	60	1.719	1.484
240	65	2.109	1.901

Water length (cm)	Water depth (cm)	Resonant period (s)	Shallow-water estimate (s)
300	70	2.480	2.290

1. A flow function MAY give `dimensions`: the inside `length` and `width` of the vessel its pumps are in, and the `depth` of its water, in cm. The tank's resonant period along its length is then T_1 for `length` and `depth`, and along its width, T_1 for `width` and `depth`. A controller MAY propose them from the Data Model `dimensions` of the vessel that its pumps' equipment is in [OAP-DM Section 8.2], for the owner to confirm.
2. A `swell` whose `wavePeriod` is `resonance`, or a `nutrient-export` whose `pulsePeriod` is `resonance`, uses the resonant period along the tank's length. A flow function whose program has such a pattern MUST have `dimensions`. A program document can carry such a pattern, which then makes waves in whatever tank it runs in.
3. Pumps that push water at a resonant period make waves that grow with every push. A smooth tank damps them very little, so they can grow until water splashes out, an overflow on an end wall surges more than its sump can take, or the tank's seams are strained. For these reasons:
   - a swell builds its waves over half its `period`, and lets them die away over the other half (Section 8.4);
   - the owner SHOULD watch the first swells, and lower `max` if water comes near the top of the tank, or the overflow surges; and
   - a controller with `dimensions` SHOULD warn the owner when the `period` of a `pulse`, `square`, `sine`, or `triangle` that its pumps run is within 10 % (provisional) of a resonant period, or of twice or three times one, at which a pulse train's harmonics drive the same wave.
4. Rockwork, the overflow, and the pumps' places shift the real resonance a little. A controller MAY let the owner adjust the computed period, as products that search for the wave by hand do; it then writes the adjusted period to pump programs (Section 15.1).

For example, the 120 cm display of Section 6.4, with water 50 cm deep, has a resonant period of 1.334 s along its length, and 0.882 s across its 60 cm width.

8.7 Custom Waveforms

1. A custom waveform has a `period`, from 1 s to 90 000 s, so that a whole tidal day of 24 hours 50 minutes fits; 2 to 1440 `points`, each a `time`, in seconds from the start of the period, and a `speed`; and an optional `interpolation`.
2. The times of the points MUST increase from one point to the next, and MUST be less than `period`.
3. With $q = x \bmod$ `period`, let a be the last point whose time is at or before q, or, if there is none, the last point, a period earlier; and let b be the next point after a, or the first point, a period later. With f the share of the time from a to b that has passed:
   - `linear` (the default): a's speed plus (b's speed − a's speed) × f;
   - `cosine`: as `linear`, with f replaced by $(1 - \cos(180° \times f)) / 2$; and
   - `step`: a's speed.
4. The `min` and `max` of a custom waveform, for its role in anti-sync, are the lowest and highest speeds of its points (Section 11.2).
5. A waveform with a `period` of 86 400 s follows local time: its point at 0 falls at local midnight. A waveform whose period does not divide a day, such as a semidiurnal tide of 44 712 s, drifts through the days, as tides do.

For example, `examples/tidal.json` has a tidal current of 44 712 s, with six points and `cosine` interpolation, for two pumps in sequence, so that each runs its flood while the other rests.

8.8 Safety Checks

Table 11 — Checks on patterns

Check	Presence	Applies to	What it requires	Section
Patterns by their equations	REQUIRED	Controllers, pumps	Every pattern's speed and direction come from pattern time by the equations of Sections 8.4 to 8.7, and random speeds from the generator of Section 8.5, the same on every product.	8.4

Check	Presence	Applies to	What it requires	Section
Programs checked before use	REQUIRED	Controllers, pumps	A product rejects a program document or a pump program that is not valid, without changing what runs, and a pump rejects a program it cannot run in full.	17.2
Reversing needs the reverse feature	REQUIRED	Controllers	A controller runs a gyre, or a pattern with `reverse`, only on a pump with the `reverse` feature, and tells the owner why it will not on another.	7.3
Resonance	RECOMMENDED	Controllers	With a tank's dimensions, a controller runs a swell whose `wavePeriod` is `resonance` at the tank's resonant period along its length, and warns about pulses, squares, sines, and triangles whose period is within 10 % of a resonant period, or of twice or three times one.	8.6

9 Schedules

9.1 Entries

Table 12 — Members of an entry

Member	Type	Presence	Description
`start`	time of day	REQUIRED	When the entry starts, each day, as an Automation time of day [OAP-AU Section 7.6].
`pattern`	identifier	REQUIRED	The pattern the entry runs.
`transition`	number (s)	OPTIONAL	How long the change to this entry's pattern takes, from 0 to 600 s. Default 10 s (Section 12.1).

1. A schedule MUST have 1 to 24 entries, listed in the order of their starts, no two at the same time.
2. On each local date, each entry starts at its `start`, and runs until the next entry starts; the last runs past midnight, until the first entry of the next date.
3. A schedule with one entry runs its pattern all day.
4. When an entry starts, the pump changes from the previous entry's pattern to its own over the entry's `transition` (Section 12.1), unless both entries name the same pattern.
5. Patterns keep to pattern time across entries: an entry that starts a pattern does not restart it, so that a pulse that runs again in the afternoon is in step with the morning's, and with the other pumps of its group.

For example, the schedule of `examples/reef.json` runs reef-crest random flow from 08:00, a pulse from 11:30, reef-crest flow again from 14:30, a nutrient export from 21:00, and gentle random flow from 21:30 until 08:00.

10 Pumps

10.1 Members of a Pump

Table 13 — Members of a pump

Member	Type	Presence	Description
`output`	identifier	REQUIRED	The output the pump member drives.
`runOn`	string	OPTIONAL	`controller`, which sends the pump its speeds, or `pump`, which runs a pump program by itself (Section 15). Default `controller`.
`role`	string	OPTIONAL	`sync`, `anti-sync`, or `sequence` (Section 11.2). Default `sync`.
`scale`	number (%)	OPTIONAL	The share of each pattern's speeds the pump runs. Default 100.
`speedLimitHigh`	number (%)	OPTIONAL	The cap: the highest speed the owner allows the function to run the pump at (Section 10.2). Default 100.
`speedLimitLow`	number (%)	OPTIONAL	The floor: the lowest speed the owner allows the function to run the pump at, other than stopped (Section 10.2). Default 0.
`flowInput`	identifier	OPTIONAL	An Automation input that measures the pump's flow (Section 13.1).

PRE-DRAFT

Member	Type	Presence	Description
`minFlow`	number	OPTIONAL	The lowest flow the owner allows while the pump runs, in the unit of `flowInput` (Section 13.5).
`fullFlow`	number (L/h)	OPTIONAL	The pump's flow at 100 % in this installation, measured or estimated by the owner. Default the endpoint's `ratedFlowRate` (Section 14.2).
`threshold`	number (%)	OPTIONAL, `switch` only	The switch is on while the pump's speed is above `threshold`. Default 0 (Section 10.5).
`minSwitchTime`	number (s)	OPTIONAL, `switch` only	The least time the switch stays on, and stays off, from 5 to 3600 s. Default 60 s (Section 10.5).

10.2 Scale, Caps, and Floors

1. `scale` multiplies every speed of every pattern but `constant-flow`, whose target is a flow. It suits one program to pumps of different strengths, such as two wave pumps of different sizes in anti-sync.
2. A pump's cap is the lower of its member's `speedLimitHigh` and its endpoint's `speedLimitHigh` [OAP-DC Section 9.3]. Its speed MUST NOT exceed its cap, whatever pattern, trigger, transition, or restore ramp asks for more, except as item 4 says.
3. A pump's floor is the highest of its member's `speedLimitLow`, its endpoint's fixed `minSpeed`, and its endpoint's `speedLimitLow`. A speed above 0 % and below the floor MUST be raised to the floor. A speed of 0 % stops the pump. Sensorless DC pump drivers cannot run their motors near zero speed, so every pump has a lowest speed.
4. A member's `speedLimitLow` MUST NOT be above its `speedLimitHigh`. If a pump's floor is above its cap anyway, for example because its endpoint's `minSpeed` of 30 % is above a member's `speedLimitHigh` of 20 %, no speed but 0 % meets both. The floor then wins, so that the water keeps moving (Section 6.2.3): the pump runs at its floor wherever its pattern asks for a speed above 0 %, and a controller MUST warn the owner, who lowers the floor or raises the cap.
5. A member's `speedLimitHigh` and `speedLimitLow` are the owner's limits for the flow function: a controller applies them to what the function asks of the pump, and writes them to pump programs (Section 15.1). The endpoint's own `speedLimitHigh` and `speedLimitLow`, which the device enforces for every command, are a separate layer [OAP-DC Section 7.8], and a controller MAY set them to the same values. Revision 0.1 called the members `maxSpeed` and `minSpeed`, which a product reads as `speedLimitHigh` and `speedLimitLow`, in pump members and pump programs; a validator warns about them, and a member or program MUST NOT give both names of one limit.
6. Caps are the owner's statement of the most flow that is safe for their tank: for sand that blows, for soft corals, for small fish, or for a pump that is too strong for the tank.

10.3 Computing Speeds

For a pump at a moment u, with pattern time T (Section 8.2), a product MUST find the speed and direction in these steps:

1. **Patterns:** the pattern of the schedule's entry in effect at u, and that of the entry before it while the entry's transition lasts (Section 9.1), and the pattern of each trigger that applies at u, or whose end is within its transition before u (Section 12.2).
2. **Role:** each pattern's speed and direction at $x = T - o$, with the offset o of the pump's role, and mirrored for a pump in anti-sync where Table 8 says so (Section 11.2).
3. **Scale and cap:** each pattern's speed times `scale` / 100, except for `constant-flow`, and then at most the cap (Section 10.2.2).
4. **Transitions:** the schedule's patterns blended over the entry's transition (Section 12.1), and then each trigger's pattern blended over that, in the order the triggers started, by its weight (Section 12.2).
5. **Restore ramp:** during a restore ramp, the speed of Section 12.3, with P the result of step 4.
6. **Floor:** a speed above 0 % and below the pump's floor is raised to it (Section 10.2.3).

The pump then applies its own limits. `tools/flow_reference.py` computes these steps.

10.4 Driving Pumps from the Controller

1. For a pump it drives, a controller MUST send commands that keep its commanded speed within 1 percentage point of the speed that Section 10.3 gives at some moment within 1 s of it, and its direction the same, except within 2 s of an edge (provisional).
2. It uses `set` with `speed`, and `running true`, for a `pump`, with `direction` when the pump has the `reverse` feature; `set` with `running false` for a speed of 0; `set` with `level` for a `level-output`; and `set` with `on` for a `switch` (Section 10.5). It MAY give a pump's `set` a `rampTime`, and a `level-output`'s a `fadeTime`, to the speed of a coming moment, so that a slow pattern changes smoothly with few commands [OAP-DC Section 7.4.10].
3. It SHOULD NOT send a pump more than one command a second (provisional), except a stop, which it never delays, as Automation Section 17.4.4 recommends for repeated commands, and it evaluates its configuration about once a second [OAP-AU Section 17.1.1], so it cannot time faster patterns. A controller MUST NOT drive a pump itself with a pattern whose timing is shorter than Table 8 allows a controller, and MUST tell the owner that the pattern needs a pump that runs its own program.
4. It changes a pump's direction only while the pattern has the pump stopped, as every pattern and transition of this Specification does (Section 16.3).
5. It sends nothing to a pump 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.
6. A pump the controller drives SHOULD have a fail-safe configuration of `run` (Section 16.2).

The limits of Table 8 come from the controller's one command a second: a pulse of less than 2 s, or a sine whose period is less than 30 s, would be ragged. Over the module bus, a controller could send commands faster, which this revision leaves to a later one.

10.5 Level Outputs and Switches

1. A `level-output` pump runs at a level equal to its speed (Section 10.3).
2. A `switch` pump is on while its speed, before the floor, is above its `threshold`, and off otherwise. A controller MUST keep it on, and keep it off, for at least `minSwitchTime` after each change of its flow function's demand; a change that falls due sooner waits until `minSwitchTime` has passed. This applies only to the flow function's own changes: a stop from a higher layer, such as an interlock, a mode, or `stop-all`, is never delayed [OAP-AU Section 17.4.4; OAP-FS Section 8.2].
3. A switch protects its relay and the pump's motor this way: a relay rated for 100 000 operations lasts about 70 days switched once a minute, and less than 12 days switched every 10 s (provisional).
4. A validator warns when a pattern on a switch repeats in less than twice its `minSwitchTime`.
5. A switch cannot hold a flow, so a `constant-flow` pattern never runs on one.

10.6 Safety Checks

Table 14 — Checks on pumps

Check	Presence	Applies to	What it requires	Section
Caps over everything	REQUIRED	Controllers, pumps	No pump runs faster than its cap, the lower of its member's and its endpoint's `speedLimitHigh`, from any pattern, trigger, transition, or restore ramp; where its floor is above its cap, the floor wins and the controller warns.	10.2
No running below the floor	REQUIRED	Controllers, pumps	A speed above 0 and below a pump's floor, the highest of its member's `speedLimitLow` and its endpoint's `minSpeed` and `speedLimitLow`, is raised to it, and a speed of 0 stops the pump.	10.2
Only what a controller can time	REQUIRED	Controllers, configurations	A controller drives a pump itself only with patterns whose timing Table 8 allows a controller; faster patterns run on pumps with the `wave` feature.	10.4
Switches spared	REQUIRED	Controllers	A pump on a `switch` stays on, and stays off, for at least `minSwitchTime` after each change of its flow function's demand; a stop from a higher layer is never delayed.	10.5

11 Groups

11.1 Pumps That Run Together

1. The pumps of a flow function are a group: they run one program, each with its own settings (Table 13).
2. A group's pumps need not come from one maker, share a binding, or be the same size. Some may run the program themselves while the controller drives others.
3. Pumps of a group never need to exchange messages: each computes the program from the same members, the same pattern time, and its role.
4. A tank whose pumps should run different programs, such as wave pumps and a back-wall pump, has a flow function for each.

11.2 Roles

Table 15 — Roles

Role	The pump's offset *o*	What it does
`sync`	0	Runs the pattern as written.
`anti-sync`	Half the pattern's phase period, for the patterns that Table 8 shifts; 0 for the patterns it mirrors, whose speeds become `min` + `max` − the speed	Pushes when the pumps in sync rest.
`sequence`	*i* / *n* of the pattern's phase period, where *n* is the number of pumps of the group whose role is `sequence`, and *i* is the pump's place among them, in the order of `pumps`, counting from 0	Takes its turn after the pump before it.

1. Every pump of a group MUST run each pattern at the offset of its role in Table 15, and mirrored where Table 8 says so, from pattern time (Section 8.2), without messages between the pumps. A pattern's phase period is the member that Table 8 names: `period` for most patterns, `wavePeriod` for a swell, and `pulsePeriod` for a nutrient export. Steady patterns, `constant` and `constant-flow`, are the same in every role.
2. In anti-sync, a pump runs a pulse, square, sine, gyre, swell, or nutrient export half its phase period later, so that pulses alternate between the pumps, as two pumps at opposite ends need to build a standing wave. For a square and a sine, this is the same as mirroring. A pump in anti-sync runs a triangle, a random pattern, or a custom waveform mirrored, high when the others are low, as most products define anti-sync.
3. A gyre half a period later runs the opposite way at each moment. Two cross-flow pumps facing each other from opposite ends, one in anti-sync, therefore push the water the same way, and turn it together.
4. Roles act on every pattern, including those of triggers: patterns change, roles stay.
5. A program's `roles` gives the roles its author meant; a controller SHOULD propose them when the owner assigns the program (Section 17.2).

For example, two pumps facing each other across a tank, running a square in anti-sync, push the water one way and then the other; four pumps around a tank, in sequence, running a pulse, send the flow around it; and two pumps at the ends of a tank, running a swell at the tank's resonance in anti-sync, make a standing wave.

11.3 Sequences

1. The pumps of a sequence MUST run the pattern *i* / *n* of its phase period apart, in the order of the function's `pumps`, so that with three pumps and a pulse of 6 s, their pulses start 2 s apart.
2. Pumps in sync or anti-sync may share a group with a sequence; they are not counted in *n*.
3. With two pumps, a sequence is the same as one pump in sync and one shifted by half a period, also for the patterns that anti-sync mirrors; `examples/tidal.json` uses this for flood and ebb currents.

11.4 Timing Between Pumps

Table 16 — Timing between pumps

What	Agreement	Section
A pump's commanded speed and the reference implementation	0.5 percentage points, except within 50 ms of an edge	7.9

What	Agreement	Section
The edges of two pumps of a group that run their own programs	50 ms	11.4
The clock of a pump that runs its own program, and its controller's	20 ms (RECOMMENDED)	15.3
The shortest period, and the shortest pulse	0.5 s, and 0.2 s	8.3

1. A pump that runs its own program SHOULD keep its clock within 20 ms of its controller's (provisional), so that the clocks of two pumps agree within 40 ms, and SHOULD report how closely it does as `clockUncertainty` [OAP-DC Section 8.1].
2. Two pumps of a group whose clocks are within 20 ms of their controller's MUST start the edges of the same pattern, at their offsets, within 50 ms of each other (provisional). 50 ms is a tenth of the shortest period, and a quarter of the shortest pulse; two equal waves 50 ms apart at a period of 0.5 s lose about 5 % of their combined strength. It leaves 10 ms for a pump to act on its pattern (Section 15.3.4).
3. On the module bus, a controller publishes network time once a second, and a module keeps its clock within 1 ms (provisional) of it while it receives the synchronization messages [OAP-MB Section 9.4.6].
4. Over the Device Protocol, the `time` in `control.supervise` reaches a device after a network delay of up to hundreds of milliseconds on Wi-Fi [OAP-DP Section 12.3]. A network pump in a group SHOULD set its clock with the time exchange `control.time`, which measures the round trip, keeping the result of the shortest of several exchanges [OAP-DP Sections 12.3.4 to 12.3.6]. On Ethernet, or on Wi-Fi without power saving, it then knows its controller's time within about 5 ms.
5. A pump's clock drifts between corrections. Two clocks that are each accurate to 20 parts per million can drift 144 ms apart in an hour, so a pump in a group MUST correct its clock at least every 60 s (provisional): on the module bus, from each synchronization message, which comes every second [OAP-MB Section 9.4]; and over the Device Protocol, from a time exchange, or, without one, from each `control.supervise`, which comes at least every 30 s [OAP-DP Section 12.2.2]. Without its controller, a group slowly loses step; this weakens alternating pulses but harms nothing, and the pumps are in step again once contact returns (Section 16.6).

11.5 Safety Checks

Table 17 — Checks on groups

Check	Presence	Applies to	What it requires	Section
Pumps in step	REQUIRED	Controllers, pumps	The pumps of a group run the program at the offsets of their roles, computed from pattern time, without messages between them.	11.2
Sequences	REQUIRED	Controllers, pumps	The pumps of a sequence run the pattern at offsets of i / n of its phase period, in the order of the function's pump members.	11.3
Clocks together	RECOMMENDED	Pumps	A pump that runs its own program keeps its clock within 20 ms of its controller's.	11.4

12 Transitions and Modes

12.1 Transitions

1. A pump MUST change from a pattern A to a pattern B over a transition of t seconds, as this item says. With f the share of the transition that has passed, and both patterns computed at the moment itself, so that both keep moving:
   - where A and B run the same direction, the speed is $(1 - f) \times A + f \times B$; and
   - where they run in different directions, the speed is $A \times (1 - 2f)$ in A's direction for the first half, and $B \times (2f - 1)$ in B's direction for the second, so that the pump stops before it reverses.
2. An entry's change uses the entry's `transition`; a trigger's start and end use the trigger's `transition`.
3. A transition of 0 changes at once, which is an edge.

4. The default transition of 10 s (provisional) neither jolts the water nor startles fish, and lets an owner see a change at once.

12.2 Triggers

Table 18 — Members of a trigger

Member	Type	Presence	Description
`when`	condition	REQUIRED	The condition while which the trigger's pattern runs, such as a `mode` condition of a feed mode, or a `schedule` condition for the night.
`pattern`	identifier	REQUIRED	The pattern of the program that runs.
`maxDuration`	number (s)	OPTIONAL	The longest the trigger applies at a time, from 60 to 86 400 s. Default 3600 s.
`transition`	number (s)	OPTIONAL	How long the changes to and from the pattern take, from 0 to 600 s. Default 10 s.

1. A controller MUST apply a trigger from the moment its `when` condition becomes true until its condition becomes false or unknown, or until `maxDuration` has passed, whichever is first. It applies again only after its condition has been false or unknown.
2. A trigger's pattern comes in over its `transition` after the trigger starts, and goes out over its `transition` after it ends. Where several triggers apply, their patterns are layered in the order the triggers started, the latest on top: the speed is the schedule's, blended with each trigger's pattern in turn by the trigger's weight, which rises from 0 to 1 over its transition after it starts, and falls back to 0 over its transition after it ends. So a feed during the night runs the feeding pattern, and the night pattern returns after it.
3. An unknown condition ends a trigger: the program's own schedule is the safe side, and the controller raises `input-unavailable`.
4. Triggers are not saved. After a restart, a trigger whose condition is true applies from the restart, with its `maxDuration` counted from then.
5. For a pump that runs its own program, the controller MUST send the pattern on top with `runProgram`, with `pattern`, `transition`, and `revertAfter`: the time left until the trigger ends at the latest, when its `maxDuration` has passed, or, for a trigger whose `when` is a `mode` condition of a mode with a known end [OAP-AU Section 10.8], when that mode ends, whichever is first (Section 15.4). When that end changes, as when the owner extends the mode, the controller sends the pattern again with the new `revertAfter` [OAP-AU Section 17.5.2]. When another trigger's pattern comes on top, it sends that pattern in the same way, and its `revertAfter` keeps the target of the revert already pending, the pump's schedule [OAP-DC Section 7.4.6]. When the trigger on top ends, the controller sends the pattern that is then on top, with the `revertAfter` of its own trigger, or, if none is left, `runProgram` with neither `pattern` nor `revertAfter`, which returns the pump to its schedule and cancels the pending revert. So every triggered pattern ends on the pump, even if the controller fails [OAP-AU Section 17.5]: when the revert falls due, the pump returns to its schedule, whatever triggers were layered, changing over the `transition` of the latest command [OAP-DC Section 7.4.12]. For example, the trigger of a storm mode of 30 minutes sends the storm pattern with `revertAfter` 1800, and, 10 minutes later, the trigger of a feed mode of 10 minutes sends the feeding pattern with `revertAfter` 600; if the controller then fails, the pump returns to its schedule 600 s later, which ends the storm early, rather than going back to a storm that would never end.
6. For a pump the controller drives, it sends the speeds of Section 10.3. If the controller fails, the pump takes its fail-safe behavior (Section 16.2).
7. When the owner assigns a program, a controller SHOULD propose triggers from its patterns' purposes: the feeding pattern for each feed mode, the night pattern for the night, the storm pattern for a storm mode, and the battery pattern for the power-loss mode.
8. A trigger whose `when` is a `mode` condition of a mode with a `duration` SHOULD have a `maxDuration` no longer than that `duration`, so that the trigger's pattern ends with the mode, and a validator warns about a longer one.

12.3 Restore Ramps

1. A pump MUST return to its program with the restore ramp, evenly over `restoreRamp` from what it shows to the program's speed:
   - after power-up: from 0 % for a pump that runs its own program, or that restores a speed, and otherwise from what the pump shows when its controller resumes control (Section 16.5);
   - when a higher layer's stop, run, or level ends (Section 7.6.3);
   - when its program is replaced; and
   - when its clock becomes valid again (Section 16.5).

   During the ramp, the speed is $S_0 + (P - S_0) \times s$ / `restoreRamp`, where S_0 is what it showed when the ramp began, P the program's speed now, and s the seconds since the ramp began.
2. If the program's direction differs from the pump's, the ramp starts from a stop.
3. A pump under constant flow returns to the speed at which it last held its target, and its control resumes from there (Section 13.2).
4. The default of 30 s (provisional) brings pumps back gently, so that food and detritus that settled during a stop are not blasted up at once, and a sump's level follows a return pump as it restarts.
5. A flow function's `restoreRamp` SHOULD be at least 10 s (provisional), and a validator warns about a shorter one: a shorter ramp starts a pump almost at once.
6. A `switch` pump cannot ramp: it switches on when the speed of its ramp passes its `threshold` (Section 10.5).

12.4 Feed

1. Strong flow carries food into overflows and filters before fish and corals take it, so owners stop or slow pumps while feeding, usually for 5 to 10 minutes.
2. A controller does this in one of two ways:
   - the feed mode lists the pump in its `outputs`, with `stop` or a level, which the Automation preset proposes for circulation pumps, and which reaches the pump with `revertAfter` [OAP-AU Section 17.5]; or
   - the feed mode leaves the pump out, and the flow function has a trigger whose `when` is a `mode` condition of the feed mode, with a pattern whose purpose is `feed`, such as a steady 10 %, which keeps some movement to spread food.
3. A feeding trigger's `maxDuration` SHOULD be the feed mode's `duration` (Section 12.2.8).
4. A feed mode SHOULD last 10 minutes (provisional), and SHOULD stop the return pump, as the Automation preset does, so that food is not carried to the sump.
5. A return pump slowed below the speed at which it lifts water to the display delivers nothing, but keeps its line full; some owners use this for feeding instead of a stop.

12.5 Night

1. Many owners give their tanks gentler flow at night, and some products slow pumps at night by default.
2. A flow function changes its pumps' flow at night in one of two ways:
   - an entry of the program's schedule at a fixed time, as in `examples/reef.json`, which a pump that runs its own program follows even without its controller; or
   - a trigger whose `when` is a condition, such as a `schedule` condition from an hour after sunset to sunrise [OAP-AU Section 10.6], or a `mode` condition of a mode of an extension kind, with a pattern whose purpose is `night`, and a `maxDuration` as long as the night.
3. Oxygen is lowest at night, after hours without photosynthesis, and the water next to corals can lose its oxygen entirely when it does not move [Shashar]. A night pattern SHOULD keep at least half of the mean speed of the day's patterns (provisional), and a validator warns otherwise.

12.6 Storms

1. A storm is strong, quick, varied flow for a while, such as a `random` pattern from 60 % to 100 % with a `period` of 2 to 3 s, whose purpose is `storm`.

2. The owner starts it with a mode of an extension kind, such as `org.openaquariumproject.storm` (Appendix D, issue 2), or a schedule starts the mode; the flow function follows the mode with a trigger.
3. A storm SHOULD last no more than 45 minutes (provisional), the longest storm of the Lighting Programs specification, and a validator warns about a storm trigger whose `maxDuration` is longer. Strong flow for longer stresses corals that are not used to it.
4. A storm never exceeds the owner's caps (Section 10.2).

12.7 Battery and Other Modes

1. On battery, the Automation preset keeps circulation and return pumps running [OAP-AU Table 29], and a power-loss mode MAY set them to a lower level, to make the battery last [OAP-FS Section 9.5]. For pumps that run their own programs, a power-loss mode MAY instead leave them out, and the flow function follow the mode with a trigger whose pattern's purpose is `battery`, which keeps a gentle pattern going; products that run on battery commonly run pumps at 10 % to 50 %.
2. The Automation presets stop circulation and return pumps in maintenance and water-change modes, and stop the return pump in an emergency, while circulation within the display continues [OAP-AU Table 29].
3. When any of these modes ends, the flow function returns its pumps to their programs with the restore ramp (Section 12.3).

12.8 Safety Checks

Table 19 — Checks on transitions and modes

Check	Presence	Applies to	What it requires	Section
Transitions	REQUIRED	Controllers, pumps	A change from one pattern to another blends them over its transition, and a change of direction passes through a stop.	12.1
Triggers end by themselves	REQUIRED	Controllers	A trigger applies while its condition is true, for at most `maxDuration`, ends when its condition becomes false or unknown, and reaches pumps that run their own programs with `revertAfter` until it ends at the latest, its mode's end included, so that a pump returns to its schedule when the revert falls due, however many triggers were layered.	12.2
Triggers end with their modes	RECOMMENDED	Configurations	A trigger that follows a mode with a `duration` has a `maxDuration` no longer than it; a validator warns about a longer one.	12.2
Restore ramps	REQUIRED	Controllers, pumps	A pump returns to its program evenly over `restoreRamp` after a power-up, at the end of a higher layer's demand, and after a new program; a pump that was following its own program, or restores a speed, ramps up from 0 % after power-up, and, from Device Classes revision 0.4, so does one whose fail-safe state runs it.	12.3
Gentle restore ramps	RECOMMENDED	Configurations	A flow function's `restoreRamp` lasts at least 10 s, so that a pump does not start almost at once; a validator warns about a shorter one.	12.3
Night keeps the water moving	RECOMMENDED	Program documents	A night pattern keeps at least half of the mean speed of the day's entries of the schedule, so that the water keeps exchanging gases when its oxygen is lowest.	12.5

13 Constant Flow

13.1 Flow Inputs

1. A pump member's `flowInput` names an Automation input of the Data Model parameter `flow-rate`: a flow sensor, as a `sensor` endpoint, often with the `pulse` feature [OAP-DC Section 11.1], or the pump's own `flowRate` attribute, read through an endpoint input with `attribute flowRate` [OAP-AU Section 9.2].
2. A flow input SHOULD measure the flow of that pump alone, for example in its return line.
3. A pump's own `flowRate` may be estimated rather than measured, as its `flowRateSource` says [OAP-DC Section 9.3], so a controller SHOULD prefer a sensor, or a `flowRate` whose `flowRateSource` is `measured`.

4. A pump member whose program has a `constant-flow` pattern, in its schedule or in a trigger, MUST have a `flowInput`, and its `runOn` MUST be `controller`, because a pump cannot read Automation inputs.

13.2 Control

1. While a `constant-flow` pattern runs, the controller sets the pump's speed so that its measured flow, converted to the pattern's `unit`, stays within `tolerance` of `flowRate`, keeping the speed from the pattern's `min` to its `max`, and within the pump's cap and floor (Section 10.2).
2. The control MUST NOT change the pump's speed by more than 1 percentage point a second (provisional). So the control is smooth, and a noisy or slow sensor cannot make the pump hunt. The limit applies to the control's own corrections: transitions, restore ramps, and stops change the speed as Sections 12.1 and 12.3 say, and a stop is never delayed.
3. It SHOULD leave the speed alone while the measured flow is within half of `tolerance` of `flowRate`.
4. It starts from the pump's current speed, or, after a stop, from the speed at which it last held the target (Section 12.3.3).
5. Where the target can be reached from `min` to `max`, a controller MUST bring the measured flow within `tolerance` of it within 120 s (provisional) of a change of the target, or of the system, such as a valve opened or closed.

The speed of a return pump does not map simply to its flow: a pump lifting water against static head delivers nothing below a speed that can be 60 % or more of its maximum, and its flow rises steeply above it [HI]. A pump's head rises with the square of its speed, so it lifts water only at speeds above √(lift / shutoff head) of its full speed, where the shutoff head is the most it lifts at full speed: about 59 % for a lift of 1.2 m against a shutoff head of 3.5 m, and 75 % for 1.35 m against 2.4 m. Constant flow sidesteps this by measuring the flow, and makes the flow independent of a clogging prefilter or a wearing impeller.

13.3 An Example Controller (Informative)

A rate-limited integral controller meets Section 13.2, and is what `tools/flow_reference.py` gives as `flow_step`:

1. Every second, take the relative error e = (`flowRate` − measured) / `flowRate`, from the mean of the readings of the last few seconds.
2. If $|e| \times 100$ is at most half of `tolerance`, leave the speed alone.
3. Otherwise change the speed by $10 \times e$ percentage points, but by at most 1 percentage point, and keep it from `min` to `max`.

Because the controller changes the speed itself, rather than an integral that the speed follows, it cannot wind up at a limit [Åström]. With a flow sensor of 450 pulses per litre, a common paddle-wheel type, a 1 s window resolves 2000 L/h to about 0.4 %, but such sensors are accurate only to about 10 %; the control holds what the sensor reads, so where the absolute flow matters, the sensor needs calibrating against a measured volume.

13.4 Lost Flow Inputs

1. While a flow input is unavailable, its pump MUST hold its mean speed of the last 60 s (provisional) in which the input was available, and the controller raises `input-unavailable`. Without such a speed, the pump runs at the pattern's `fallback`, or at its `max`.
2. Holding gives about the flow the pump had, so losing a sensor neither stops a return pump nor sends it to full speed.
3. When the input is available again, the control resumes from the held speed.

13.5 Low Flow

1. A pump member with a `flowInput` MAY have `minFlow`, in the input's unit. The condition `flow-low` holds while the pump's measured flow is below `minFlow`. It is judged only while the pump runs at its program's speed, outside transitions and restore ramps, and keeps its last state at other times, so that neither a stop nor a pump that is starting raises or clears it. A controller MUST raise its alarm once the condition has lasted its delay, 60 s (provisional), which `conditions.json` holds (Section 7.7.3), and the alarm clears as Alarms and Notifications Section 12.3 says, once the flow is at or above `minFlow`.

PRE-DRAFT

2. The condition `flow-control-limit` holds while a `constant-flow` pattern holds its pump at its highest allowed speed with the flow below `flowRate` less `tolerance`, or at its lowest with the flow above `flowRate` plus `tolerance`. A controller MUST raise its alarm once the condition has lasted its delay, 300 s (provisional), and the alarm clears once the flow is within `tolerance` [OAP-AN Section 12.3].
3. Low flow is the first sign of a clogged pump or prefilter, an air lock, a closed valve, a blocked intake, or a sump that is running low, and a return pump with low flow starves the equipment it feeds, such as a chiller or a UV sterilizer.

13.6 Safety Checks

Table 20 — Checks on constant flow

Check	Presence	Applies to	What it requires	Section
Smooth flow control	REQUIRED	Controllers	Under a constant-flow pattern, a pump's speed changes by at most 1 percentage point a second, stays within the pattern's `min` and `max`, and brings the flow within `tolerance` of its target within 120 s where it can.	13.2
Holding on a lost input	REQUIRED	Controllers	While a flow input is unavailable, its pump holds its mean speed of the last 60 s in which the input was available, and `input-unavailable` is raised.	13.4
Low flow raises an alarm	REQUIRED	Controllers	`flow-control-limit` is raised when a constant-flow pattern has held its pump at a limit for 300 s without reaching its target, and `flow-low` when a pump's measured flow has been below its `minFlow` for 60 s, each for its pump.	13.5

14 Turnover

14.1 The Turnover Index

1. The turnover of a volume of water V, by pumps whose mean flows over the last hour are Q_1 to Q_n, is $(Q_1 + \ldots + Q_n) / V$, in times an hour, with flows in L/h and volumes in L.
2. A controller computes two indices for each system:
   - **display turnover:** the flows of the pumps whose equipment is in the system's display vessels, and of the system's return pumps, over the volume of the display vessels; and
   - **return turnover:** the flows of the system's return pumps over the system's water volume, the sum of the volumes of its circulating vessels [OAP-DM Section 8.2].
3. A pump's mean flow over the hour counts the time it was stopped, by a feed mode or anything else, as no flow.

For example, in the system of Section 6.4, the left wave pump moves a mean of 5400 L/h, the right 5610 L/h, the back-wall pump 3150 L/h, and the return pump 2400 L/h, so the display turnover is 16 560 / 400 ≈ 41 times an hour, and the return turnover 2400 / 500 = 4.8 times an hour.

14.2 Flows

1. A pump's flow is, in order of preference:
   - what its flow input measures (Section 13.1);
   - its endpoint's `flowRate`, from the `telemetry` feature; or
   - an estimate: its member's `fullFlow` times its speed / 100, with `fullFlow` the endpoint's `ratedFlowRate` if the member does not give one, or, where the member gives no `fullFlow` and the endpoint has a `flowCurve`, the flow of that curve at its speed, interpolated linearly.
2. An estimate assumes that flow rises in proportion to speed, which is fair for a wave pump in the open water of the display, but wrong for a return pump lifting water to the display [HI]. Rated flows are measured with no head; a return pump can deliver half its rating or less. An owner who has measured a pump's flow at 100 % SHOULD set `fullFlow`.
3. For an estimate, a controller uses the pump's mean speed over the hour.

14.3 Showing and Recording Turnover

1. A controller SHOULD show each system's display and return turnover, updated at least every 10 minutes, marked as estimated when any flow in it is estimated, and the flow of each pump, measured or estimated.
2. A controller MAY record the turnover each hour as observations of the Data Model parameter `turnover`, in `1/h` [OAP-DM Section 9.2]. An observation of display turnover gives the display as its `vessel`; one of return turnover gives no vessel. Revision 0.1 recorded it as the extension parameter `org.openaquariumproject.turnover`, in the extension unit `org.openaquariumproject.per-hour`, which a consumer reads as `turnover` in `1/h`.
3. Before it runs a program it has imported, a controller SHOULD show the display turnover it expects from it: each pump's `fullFlow` times its scale and the mean speed of the program over a day, from the mean speed of each pattern weighted by the time its entry runs.
4. In a well-mixed tank, one turnover brings only about 63 % of the water past the pumps, and three turnovers about 95 % [Simmonds], which is why turnovers of several times an hour are usual.

14.4 Typical Turnover (Informative)

Turnover is a rough measure: corals need flow at their surfaces, which depends on where pumps point as much as on how much water they move. Guidance is mostly rules of thumb, which Table 21 gathers.

Table 21 — Typical turnover

System	Typical turnover	Basis
Reef, general	10 to 20 times an hour	Common hobby guidance, without studies behind it.
Reef, small-polyp stony corals	20 to 50 times an hour	Hobby guidance; one maker grades its turnover index from fish-only to high-flow stony corals.
Return pump through a sump	3 to 5 times an hour, within the overflow's capacity	Hobby guidance; the display's flow comes from wave pumps rather than the return.
Planted tank with added CO2	About 10 times an hour	Hobby convention; about 3 to 4 times without CO2.
Fish-only and freshwater community tanks	No authoritative figure found	—

Measured on real reefs, water moves from about 1 to 80 cm/s [Johansen], in waves with periods of about 2 to 10 s, and oscillating flow carries 1.6 to 2.9 times as much to a coral as a steady current of the same speed, at wave periods from 5 s down to 1 s [Reidenbach]. Fish are comfortable swimming at 1 to 2 of their body lengths a second for long periods [Rodgers].

15 Pumps That Run Their Own Programs

15.1 Pump Programs

1. A pump with the `wave` feature runs the pump program in its `program` attribute [OAP-DC Section 9.3], written as JSON text [RFC 8259] without whitespace between tokens, or a reference to it (Section 15.2). An empty `program` means that the pump follows `set` commands.
2. A pump program has the members of Table 6 that say what the program does (`seed`, `patterns`, `schedule`, and `extensions`), and MAY have its `id` and `name`; the members of Table 13 that apply to the pump (`scale`, `speedLimitHigh`, and `speedLimitLow`); and the members of Table 22. It MUST be valid against `$defs/pumpProgram` in `schema/flow-programs.schema.json`.
3. A pump program never has a `wavePeriod` or `pulsePeriod` of `resonance`: the controller writes the tank's resonant period in seconds, to the millisecond, because the pump does not know the tank (Section 8.6). It never has a `constant-flow` pattern (Section 13.1.4).
4. A pump MUST reject a write of a program it cannot run in full, such as a gyre on a pump without the `reverse` feature, or a pattern of a type it does not implement (Section 8.3.1), and keep its previous program [OAP-DC Section 7.13.2].
5. There is one conformance level: a conforming pump that runs its own programs runs every pump program that the schema allows, up to 65 536 bytes (Section 20.2.1; Appendix D, issue 14). Parsed into numbers, the

largest takes less than 32 kB, which a pump can keep in its flash memory rather than its working memory. A pump that cannot run such programs does not conform as a pump that runs its own programs, and its controller drives it, as it does any pump that rejects the program written to it (Section 15.2.6).

Table 22 — Further members of a pump program

Member	Type	Presence	Description
`phase`	number	OPTIONAL	The pump's place in a sequence, as *i* / *n* (Section 11.2), from 0 up to, but not including, 1. Default 0.
`antiSync`	boolean	OPTIONAL	`true` for a pump in anti-sync. Default `false`.
`restoreRamp`	number (s)	OPTIONAL	The flow function's restore ramp (Section 12.3). Default 30 s.
`posixTimeZone`	string	REQUIRED	The local time zone, as a POSIX `TZ` string [POSIX], such as `AEST-10`, as in Automation.

15.2 Writing a Pump Program

1. For a pump whose `runOn` is `pump`, the controller MUST write its pump program within 60 s (provisional) of its differing from what the pump reports. For a pump whose `runOn` is `controller` and that has the `wave` feature, it writes an empty `program`.
2. When the text fits in one value of the pump's binding, the controller writes the text: on the module bus, a value holds 256 bytes [OAP-MB Section 11.6.1], and 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].
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 65 536 (provisional). It serves the text as a file at the path `oap/flow/` followed by the hash and `.json`, for as long as the pump holds the reference, on the module bus [OAP-MB Section 11.8] and over the Device Protocol [OAP-DP Section 13.2.2], as Lighting Programs Section 16.2.3 serves lighting programs.
4. A pump 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.
5. A pump keeps its program across power loss [OAP-DC Section 7.3.3], so that after power-up it runs it without its controller.
6. The condition `program-not-loaded` holds while a pump whose `runOn` is `pump` does not report the program its controller wrote to it. Its alarm is raised once it has lasted its delay, 60 s (provisional), which `conditions.json` holds (Section 7.7.3). Once the condition has lasted its delay, the controller MUST drive the pump itself, until the pump reports the program, with the patterns it can drive (Section 10.4), and, while the pump's `failSafeAction` is `continue`, with no pattern that stops it (Section 16.2.1). A pattern that it cannot drive, or that would stop such a pump, leaves the pump as it is: on its previous program, or at the speed it was last sent. Before then, the pump runs its previous program, so a program that takes a few seconds to load does not take the pump off its program.
7. The controller SHOULD write the flow function's `id` to the `syncGroup` attribute of each pump that runs its own program [OAP-DC Section 9.3]. A pump that also synchronizes with other pumps by means of its own, such as its maker's radio link, MUST still keep to the timing of its program.

A pump program with two patterns takes a few hundred bytes, and that of `examples/reef.json` about 1100 bytes, so on the module bus a program is usually written as a reference, and over the Device Protocol as text. `examples/pump-network.json` and `examples/pump-module.json` show both.

15.3 Running a Pump Program

1. A pump with a pump program follows Sections 8 to 12 by itself, with its own clock, and MUST meet Section 7.9 against the reference implementation.
2. With `failSafeAction continue`, a pump that was following its program when it lost contact with its controller MUST keep running it, with its schedule, its role, and its settings, 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, ramps to a

`failSafeState` that runs it from the speed it runs at [OAP-DC Section 8.3.6]. A pump that a latched stop holds keeps it in place of either [OAP-FS Section 8.4].
3. It reports `programRunning` [OAP-DC Section 9.3]: `true` while it runs its program, its schedule or a pattern that `runProgram` gave it, and `false` while it follows `set`, a stop, or its fail-safe state.
4. It SHOULD set its motor's speed from its pattern at least every 10 ms (provisional), so that the shortest pulse keeps its shape, and the edges of a group stay within Section 11.4.2.
5. It SHOULD keep its clock within 20 ms of its controller's (Section 11.4).
6. A pump cannot read Automation inputs, so triggers and holds reach it through its controller (Section 15.4).

15.4 Under a Controller

1. A controller returns a pump to its program, and sends it triggered patterns, with `runProgram` [OAP-DC Section 9.3], whose parameters are all OPTIONAL:
   - `pattern` (string): a pattern of the pump's program to run in place of its schedule;
   - `transition` (number, s): the change from what the pump runs to the program or the pattern, from 0 to 600 s; default the program's `restoreRamp`; and
   - `revertAfter` (number, s): as for `set` [OAP-DC Section 7.4].

   The pump cancels any speed from `set`, and runs its program, or the pattern. 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 pump that was following its program is its schedule, and when the revert falls due, a pump of Device Classes revision 0.3 or later changes to it over the `transition` of the latest command with `revertAfter` [OAP-DC Section 7.4.12]. A pump MUST reject a `pattern` that its program does not have.
2. The controller sends it:
   - to resume control of the pump, for example after contact returns [OAP-FS Section 7.4], with the pattern of any trigger that applies [OAP-DC Section 7.13.5];
   - when a higher layer's stop, run, or level of the pump ends, with `transition` the restore ramp (Section 7.6.3); and
   - when a trigger starts or ends, as Section 12.2.5 says.
3. A controller MUST NOT let a pump run its own program unless the pump implements `runProgram`, as every pump of Device Classes revision 0.2 or later with the `wave` feature does; it drives any other pump itself, with the patterns it can drive. Without the command, a controller could not return the pump to its program after a stop with no known end, such as a power-loss mode [OAP-AU Section 17.3.4].
4. A pump 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, and as `org.openaquariumproject.program-running` in the endpoint's `extensions` [OAP-DC Table 5]. A controller MAY let such a pump run its own program if the pump reports `org.openaquariumproject.program-running`, and then uses these names in place of `runProgram` and `programRunning`. A pump of revision 0.2 or later SHOULD NOT use them.
5. A pump of revision 0.1 or 0.2 cancels a pending revert with every `runProgram`, and returns at once when a revert falls due [OAP-DC Table 4]. Before it sends such a pump a pattern with `revertAfter` while a revert is pending, a controller sends `runProgram` with neither `pattern` nor `revertAfter`, so that the new revert's target is the schedule rather than the pattern beneath, which would otherwise have no end if the controller failed. It sends such a pump a triggered pattern with a `revertAfter` that is longer, by the trigger's `transition`, than Section 12.2.5 gives, and, when the trigger ends, `runProgram` with the `transition`, which cancels the revert, so that the pump changes back over the transition rather than at once, unless its controller has failed.
6. The controller watches `programRunning` and the pump's `speed` and `direction`, and shows the owner what the pump does (Section 7.8). A pump that runs a fast pattern reports a commanded speed that changes faster than its state reports, at most 10 a second on the module bus [OAP-MB Section 11.1]; the controller MUST NOT treat the difference between the reported speed and its own computation as a fault or a refused command.

15.5 Safety Checks

Table 23 — Checks on pumps that run their own programs

Check	Presence	Applies to	What it requires	Section
The same program on a pump	REQUIRED	Pumps	A pump that runs a pump program does what the reference implementation computes, within Section 7.9.	15.3
Writing pump programs	REQUIRED	Controllers	A controller writes each pump program within 60 s of its differing from what the pump reports: as text when it fits the binding, and otherwise as a reference to a file it serves.	15.2
Checking a program file	REQUIRED	Controllers, pumps	A pump 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 `program-not-loaded`.	15.2
Returning to the program	REQUIRED	Controllers, pumps	A controller returns a pump that runs its own program to it, and sends it triggered patterns, with `runProgram`, which cancels speeds from `set`, and a pending revert unless it carries `revertAfter`; a pump without `runProgram` does not run its own program.	15.4
Running without a controller	REQUIRED	Pumps	A pump with `failSafeAction continue` keeps running its program, with its schedule, role, and settings, when it loses its controller.	15.3

16 Safety and Failures

16.1 Return Pumps

A return pump is a pump whose output's equipment category, or whose endpoint's `loadCategory`, is `return-pump`.

1. A return pump MUST NOT have `failSafeAction continue` [OAP-FS Section 7.3; OAP-FS Table 3]: its last command may have been a stop or a slow speed with no known end that its controller did not latch, such as a power-loss mode's [OAP-AU Section 17.5.6], which it would keep if its controller failed. A controller MUST NOT set `continue` on a return pump, and MUST warn the owner about a return pump that has it. A pump whose `loadCategory` is `return-pump` SHOULD reject a write of `failSafeAction continue` (provisional). Its fail-safe state SHOULD be running at its normal speed, such as `{"running": true, "speed": 70}`.
2. A flow function MUST run only `constant` patterns whose `speed` is above 0 %, and `constant-flow` patterns whose `min` and `fallback` are above 0 %, on return pumps. A wave pattern on a return pump makes the water in transit through the overflow rise and fall, so that the sump's level swings, the overflow surges and gurgles, and the equipment the return pump feeds sees its flow come and go. A pattern that can stop a return pump would stop it with no end, and a constant-flow pattern whose `min` is 0 % would stop it when its flow sensor reads high; a return pump is stopped instead by modes, overrides, interlocks, and `stop-all`, whose temporary stops carry their own ends [OAP-AU Section 17.5].
3. Changes of a return pump's speed SHOULD take at least 30 s (provisional), with an entry's or a trigger's `transition`, and the restore ramp, so that the sump's level follows gently.
4. A return pump's member's `speedLimitLow` SHOULD be at least the lowest speed at which it still delivers the flow that the equipment it feeds needs, such as a chiller, a UV sterilizer, or a reactor. Automation `requires` interlocks stop such equipment when the return pump stops, not when it slows [OAP-AU Section 13.3], and below a certain speed a return pump moves no water at all (Section 13.2).
5. A return pump with a flow input SHOULD have `minFlow` (Section 13.5).
6. When a return pump stops, the water above the overflow, in the overflow, and in a submerged return line drains to the sump, which needs room for all of it; that is a matter for the system's design rather than for this Specification.

16.2 Fail-Safe Configuration

Table 24 gives the RECOMMENDED fail-safe configuration [OAP-FS Section 13] of pumps. It follows the Fail-Safe Behavior recommendations for the categories `circulation-pump` and `return-pump` [OAP-FS Table 3].

Table 24 — Recommended fail-safe configuration of pumps

Pump	On loss of contact	After power-up	On battery	Notes
Circulation pump that runs its own program	`continue`	`fail-safe`	`run`	It keeps its program, its role, and its schedule. A triggered pattern ends when its revert falls due, with a ramp on a pump of Device Classes revision 0.3 or later. After power-up, it runs its program, beginning with the restore ramp from 0 %.
Circulation pump driven by the controller	`run`	`fail-safe`	`run`	With `continue`, it would keep whatever speed it was last sent: a stop at the bottom of a pulse, or a stop with no known end that is not latched, such as one of a mode that ends only by `end.when`. From Device Classes revision 0.4, it ramps to its fail-safe speed over 30 s, after power-up and when its controller fails.
Return pump	`run`	`fail-safe`	`run`	Never `continue` (Section 16.1). A power-loss mode may slow it. From Device Classes revision 0.4, it ramps to its fail-safe speed over 30 s, after power-up and when its controller fails.

1. A controller MUST NOT drive a pump whose `failSafeAction` is `continue` with a pattern that stops it, and SHOULD propose `run` instead.
2. With `failSafeAction continue` and `powerOnBehavior fail-safe`, a pump that runs its own program runs it after power-up without its controller [OAP-FS Section 9.1].
3. "On battery" is the Automation power-loss mode (Section 12.7).

16.3 Direction and Speed Changes

1. A pump MUST come to a stop before it changes direction, and MUST stay stopped for at least its `reversePause`, or, if it gives none, for a pause its manufacturer documents, whatever it is commanded, as the `reverse` feature requires [OAP-DC Section 9.3]. Reversing a spinning motor drives large currents through it and its driver.
2. Every pattern and transition of this Specification changes direction only through a stop: a gyre stops for its `pause`, by default 2 s (provisional), and a transition between directions passes through a stop (Section 12.1). A pump that needs a longer pause than a pattern gives takes it, as item 1 requires, and lags the reference meanwhile (Section 7.9.2), which a transition between directions, whose stop lasts only an instant, always makes it do. A controller SHOULD warn the owner when a gyre's `pause` is shorter than the `reversePause` of a pump that runs it, and a validator warns about a pump program whose gyre pauses for less than its pump's `reversePause`.
3. A pump MAY limit how quickly its speed changes, to protect its motor; its commanded speed still follows its program, and its motor follows as quickly as it can.

16.4 Lost Inputs

Flow defines what it does when an input is unavailable, as Fail-Safe Behavior Section 10.3 requires. In every case, the water keeps moving.

Table 25 — What flow functions do when an input is unavailable

Input	Behavior	Section
The controller's clock	Pumps the controller drives keep running the pattern of the entry in effect when the clock was last valid, or the first entry, on elapsed time; triggers on schedules end.	16.5
A pump's clock	The pump runs as the controller does without a clock.	16.5
A trigger's condition	The trigger ends, and `input-unavailable` is raised.	12.2
A flow input	The pump holds its recent mean speed, and `input-unavailable` is raised.	13.4
A pump's device	Its endpoints take their fail-safe behavior; the rest of the group continues, and the controller shows the pump as lost.	16.6
A pump program's file	The pump keeps its previous program; `program-not-loaded` is raised.	15.2

16.5 Power, Restarts, and the Clock

1. After power-up, a pump follows its `powerOnBehavior` [OAP-FS Section 9.1]:
   - a pump 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`, MUST run it with the restore ramp from 0 %;
   - a pump that restores a speed, because its `powerOnBehavior` is `last`, MUST ramp up to it from 0 %, beginning when its `powerOnDelay` ends, over its program's restore ramp if it returns to its program, and otherwise over 30 s (provisional), as Device Classes Section 8.3.3 requires; and
   - a pump whose `powerOnBehavior` is `fail-safe`, and whose `failSafeAction` is `state`, enters its `failSafeState`, as Table 24 recommends for pumps the controller drives, and, from Device Classes revision 0.4, MUST ramp up to a `failSafeState` that runs it from 0 %, beginning when its `powerOnDelay` ends, over 30 s (provisional) [OAP-DC Section 8.3.5]; a pump of an earlier revision steps to it.

   A pump that a latched stop held when it lost power stays stopped [OAP-FS Section 8.4]. Once the controller resumes control of a pump it drives, it ramps the pump from what it reports to its program's speed with the restore ramp (item 4).
2. A pump whose clock is not valid MUST run the pattern of the schedule's entry that was in effect when it last had a valid time, if it knows it, or else the first entry, with its own elapsed time from power-up in place of pattern time, and without its role's offset. It reports `clock-invalid` meanwhile [OAP-FS Section 10.5.6]. Unlike a light, it does not stop: moving water matters more than the right pattern. Its group loses step until its clock is valid again, and it then returns to its schedule with the restore ramp.
3. While a controller's clock is not valid, it MUST drive its pumps in the same way. Triggers whose conditions need the time end, because their conditions are unknown (Section 12.2.3).
4. After a restart, a controller drives its pumps with the restore ramp from what they report (Section 12.3).
5. When mains power returns after a power-loss mode, the pumps return to their programs with the restore ramp, in the order and with the delays of Fail-Safe Behavior Section 9.2.
6. A pump SHOULD keep running while it updates its firmware [OAP-FS Section 11.2], and returns to its program with the restore ramp afterwards.

16.6 A Failed Controller

When the controller fails:

1. pumps that run their own programs, with `failSafeAction continue`, keep their programs, their schedules, and their roles, and their group stays roughly in step for hours, drifting apart as their clocks drift (Section 11.4);
2. a triggered pattern ends on each pump when its revert falls due, with a ramp on a pump of Device Classes revision 0.3 or later (Section 12.2.5);
3. pumps the controller drove, and return pumps, take their fail-safe state, running, to which they ramp from Device Classes revision 0.4 (Table 24), unless a latched stop holds them [OAP-FS Section 8.4]; and
4. 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].

16.7 Safety Checks

Table 26 — Checks on safety and failures

Check	Presence	Applies to	What it requires	Section
Return pumps never continue	REQUIRED	Controllers	A controller never sets `failSafeAction continue` on a return pump, and warns the owner about a return pump that has it.	16.1
Return pumps refuse continue	RECOMMENDED	Pumps	A pump whose `loadCategory` is `return-pump` rejects a write of `failSafeAction continue`.	16.1

PRE-DRAFT

Check	Presence	Applies to	What it requires	Section
Return pumps keep running	REQUIRED	Controllers, configurations	A flow function runs on return pumps only `constant` patterns above 0 %, and `constant-flow` patterns whose `min` and `fallback` are above 0 %.	16.1
No stop kept by a failed controller	REQUIRED	Controllers	A controller does not drive a pump whose `failSafeAction` is `continue` with a pattern that stops it.	16.2
Stop before reversing	REQUIRED	Pumps	A pump stops, and stays stopped for at least its `reversePause`, or its documented pause, before it changes direction, whatever it is commanded.	16.3
The water keeps moving without a clock	REQUIRED	Controllers, pumps	Without a valid clock, a controller keeps driving its pumps, and a pump keeps running its program, with the pattern of the entry last in effect, on elapsed time.	16.5

17 Sharing

17.1 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/flow-programs.schema.json`, and meeting every requirement of Sections 8 and 9 for each of its programs.
2. Each program's `id` is unique within the document.
3. A program document describes flow, not a tank: it carries no bindings, flow inputs, triggers, or settings of particular pumps. 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
`oapFlowPrograms`	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.
`generator`	object	OPTIONAL	The product that wrote the document, as a Data Model Generator [OAP-DM Table 2].
`created`	timestamp	OPTIONAL	When the document was written.
`programs`	array	REQUIRED	The programs, at least one (Section 8).
`extensions`	object	OPTIONAL	Extensions.

17.2 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 pump 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 group of pumps, a controller MUST show the owner:
   - the pattern of each entry of the schedule;
   - the roles the program suggests, and the roles it proposes for the owner's pumps (Section 11.2.5);
   - the triggers it proposes from the patterns' purposes (Section 12.2.7);
   - the patterns that need the `reverse` feature, or a pump that runs its own program, and which of the owner's pumps cannot run them (Sections 7.3 and 10.4); and
   - the display turnover it expects (Section 14.3.3). A program with a `resonance` period needs the tank's `dimensions`; the controller MUST ask the owner for them if there are none.
4. A controller MUST NOT run an imported program on a return pump unless every pattern its schedule and triggers run is one that Section 16.1.2 allows on a return pump.
5. The owner MUST confirm an imported program before it runs [OAP-AU Section 19.2.5].

17.3 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 flow program carries nothing about the owner except what the owner puts in it, such as a name in `author`. Before it shares a program, a controller SHOULD offer to remove the author's name and the description [OAP-AU Section 21].

18 History

1. Flow measured by a flow input is recorded as Data Model observations of `flow-rate` [OAP-DM Section 9.2].
2. A pump's motor speed from the `telemetry` feature MAY be recorded as observations of `pump-speed`, in `rpm`.
3. Turnover is recorded as Section 14.3.2 says.
4. The program a flow 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].
5. Faults, such as `flow-low`, are recorded through the history of their alarms [OAP-AN].

19 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 speeds, caps, flows, and the times of the schedule, are set by the owner and the program's author.

Table 28 — Recommended values

Area	Value	Recommended	Why
Patterns	Speed agreement	0.5 %	Half a percentage point is far less than anyone can see or measure in a pump's flow, and allows for differences in arithmetic between products (Section 7.9).
Patterns	Edge tolerance	50 ms	A tenth of the shortest period, and a quarter of the shortest pulse; two equal waves 50 ms apart at a period of 0.5 s lose about 5 % of their combined strength (Section 7.9).
Patterns	Resonance warning	10 %	Waves grow when the forcing is near a resonant period; a tenth either side covers the shift that rockwork, the overflow, and a mismeasured depth cause (Section 8.6).
Patterns	`pause`	2 s	No maker publishes a pause before reversal, and reversing a spinning motor drives large currents; 2 s lets a small cross-flow or propeller pump coast to rest (Section 16.3). To be confirmed with makers.
Patterns	Gyre `ramp`	3 s	Spreads the start and stop of each half over several seconds, which spares the motor and does not startle fish.
Pumps	Controller commands	1 s	A controller evaluates its configuration about once a second (Automation Section 17.1), and sends a pump no more than one command a second (Section 10.4), which limits the patterns it can drive.
Pumps	`minSwitchTime`	60 s	A relay rated for 100 000 operations lasts about 70 days switched once a minute, and a motor started once a minute has time to cool (Section 10.5). To be confirmed.
Groups	Clock agreement	20 ms	Two pumps then agree within 40 ms, inside the 50 ms that the edges of a group may differ by; two clocks accurate to 20 parts per million drift 144 ms apart in an hour, so they need frequent correction (Section 11.4).
Groups	Shortest period	0.5 s	Shorter than the resonant period of a tank 30 cm long, about 0.6 s, the quickest wave a tank needs; products tune waves from 0.1 to 2 s by hand (Section 11.4).

PRE-DRAFT

Area	Value	Recommended	Why
Groups	Shortest pulse	0.2 s	Four times the edge tolerance, so that a pulse keeps its shape on every pump of a group (Section 11.4).
Transitions and modes	`transition`	10 s	Long enough not to jolt the water or startle fish, short enough that an owner sees a change at once (Section 12.1).
Transitions and modes	`restoreRamp`	30 s	Brings pumps back gently, so that food and detritus that settled during a stop are not blasted up at once, and a sump's level follows a return pump as it restarts (Section 12.3).
Transitions and modes	Shortest restore ramp	10 s	A ramp shorter than the default transition starts a pump almost at once, and stirs up what settled during a stop (Section 12.3).
Transitions and modes	`maxDuration`	3600 s	A trigger that ends by itself cannot be forgotten; an hour covers feeding, storms, and maintenance (Section 12.2).
Transitions and modes	Feed duration	600 s	Products feed for 5 to 10 minutes, and one pump's button for 30; 10 minutes is long enough for fish and corals to take food, and short enough that circulation is not missed (Section 12.4).
Transitions and modes	Storm duration	2700 s	The longest storm of the Lighting Programs specification, so that flow and light can storm together; one controller's example storm lasts 10 minutes. Strong flow for longer stresses corals that are not used to it (Section 12.6).
Transitions and modes	Night share	50 %	Oxygen is lowest at night, and the water next to corals loses its oxygen when it does not move [Shashar]; one maker's night mode runs at 50 % (Section 12.5).
Constant flow	Speed change	1 percentage point a second	Slow enough that a noisy or slow flow sensor cannot make the pump hunt, quick enough to correct a clogging filter within a minute (Section 13.2).
Constant flow	`tolerance`	5 %	Half the accuracy of a common paddle-wheel flow sensor, about 10 %, so that the control holds what the sensor reads without chasing its noise (Section 13.2).
Constant flow	Settling time	120 s	At 1 percentage point a second, enough to change the speed by 60 points with time to spare (Section 13.2).
Constant flow	Held speed	60 s	The mean of the last minute smooths out the control's last corrections, and gives about the flow the pump had (Section 13.4).
Constant flow	Low flow delay	60 s	Longer than a transition or a restore ramp, so that a pump that is starting does not raise an alarm (Section 13.5).
Constant flow	Control limit delay	300 s	Long enough that only a lasting shortfall, such as a clogged filter, raises an alarm (Section 13.5).
Pumps that run their own programs	Writing a program	60 s	As for lights: long enough for a module to read a file over a busy bus, short enough that an owner sees a change at once (Section 15.2).
Pumps that run their own programs	Largest pump program	65536 bytes	Room for a custom waveform of 1440 points and a full library of patterns, within the memory of a small pump's microcontroller, as for lights (Section 15.2).
Pumps that run their own programs	Speed updates	10 ms	A hundred updates a second keep the shortest pulse, of 0.2 s, in shape, and add at most 10 ms to the 40 ms by which two clocks may differ (Sections 11.4 and 15.3).
Safety and failures	Return pump changes	30 s	Lets the water in transit through an overflow change gradually, so that the sump's level follows without surging; return pumps lift against static head, so small changes of speed make large changes of flow [HI] (Section 16.1).

20 Conformance

20.1 Controllers

A conforming controller:

1. supports flow functions, and every member of programs, patterns, flow functions, and program documents that this Specification defines;

2. computes speeds as Section 10.3 says, within Section 7.9, and drives pumps as Section 10.4 says;
3. meets every requirement for controllers in Sections 7 to 18, including those for pumps that run their own programs (Section 15) and for constant flow (Section 13);
4. treats a flow function it cannot run in full, such as one with a pattern of a type 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 pumps to their fail-safe behavior; and
5. passes the test procedures of Section 20.5 that apply to controllers.

20.2 Pumps

1. A conforming pump with the `wave` feature runs pump programs as Section 15 says, within Section 7.9; implements POSIX `TZ` strings (Section 7.5), every pattern of the library, the generator (Section 8.5), and `runProgram` and `programRunning` (Section 15.4); reads references (Section 15.2); and passes the test procedures of Section 20.5 that apply to pumps.
2. Every conforming pump, whether or not it runs programs, enforces its own `speedLimitHigh`, `speedLimitLow`, and `minSpeed` (Section 10.2), and stops, and pauses, before it reverses (Section 16.3).

20.3 Documents

A conforming program document meets Section 17.1, and a conforming configuration document with flow functions meets Section 7.2 and every requirement of Sections 8 to 16 that applies to documents, including those a schema cannot express, such as the timing of patterns, the order of entries, the patterns of return pumps and their speeds, and the patterns a controller can drive.

20.4 Schema and Tools

- `schema/flow-programs.schema.json`, whose `$id` is `https://openaquariumproject.org/schemas/flow-programs/0.3`, defines the program document at its root, and the program (`$defs/program`), pattern (`$defs/pattern`), flow function (`$defs/flow`), pump program (`$defs/pumpProgram`), and program reference (`$defs/programReference`).
- `patterns.json` holds Tables 8 and 9, and `checks.json` the safety checks, their test procedures, and the recommended values.
- `tools/validate_flow.py` validates program documents, Automation configuration documents with flow functions, device descriptions of pumps with pump programs, and the test vectors, against the schema and the rules of this Specification that a schema cannot express. It warns about patterns that nothing runs, night patterns much slower than the day, storms longer than recommended, triggers that outlast the modes they follow, short restore ramps, patterns near a tank's resonance, patterns faster than a switch can follow, gyres that pause for less than a pump's `reversePause`, return pumps without `minFlow`, pumps whose floor is above their cap, pattern types that this revision does not define, and the function type of revisions 0.1 and 0.2.
- `tools/flow_reference.py` is a reference implementation of pattern time, every pattern of the library, schedules, roles, transitions, triggers, restore ramps, resonant periods, the example flow controller, mean speeds, and turnover. It uses the local time and the generator of the Lighting Programs reference implementation.
- `test-vectors.json` holds its results for pattern time, resonant periods, the generator, every pattern of the library, and the pump programs of the examples, written by `tools/make_flow_vectors.py`.

```
python tools/validate_flow.py specs/flow-programs/examples/*.json specs/flow-programs/test-vectors.json
```

20.5 Test Procedures

Tests use a controller with simulated pumps whose commanded speeds, commands, and refusals the tester can watch, whose power the tester can remove, and whose clock the tester can set or stop, a flow sensor whose reading the tester can set, and a pump with the `wave` 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 11, 14, 17, 19, 20, 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 21 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
Patterns by their equations (8.4)	Controllers, pumps	7.9.1, 8.4, 8.5.2	Compute every entry of `random` and `patterns` in `test-vectors.json` on a controller, and as pump programs on a pump.	Every draw matches. Every speed is within 0.5 percentage points of the vector, and every direction is the same, except within 50 ms of an edge.
Programs checked before use (17.2)	Controllers, pumps	8.1.1, 8.3.1, 8.7.2, 15.1.2, 15.1.4, 17.1.1, 17.2.2, 21.1	Import program documents with a pulse whose `pulseWidth` is longer than its `period`, a custom waveform of 1441 points, a custom waveform whose times do not increase, and a gyre whose ramps and pause are longer than half its period, and write each as a pump program to a pump. Then write a gyre to a pump without the `reverse` feature, and a program of 70 000 bytes.	Each is rejected. The program that ran before still runs, and the pump reports its previous `program`.
Reversing needs the reverse feature (7.3)	Controllers	7.3.2	Assign the gyre-swell example to a pump without the `reverse` feature, and to one with it.	The controller refuses the first, and says why. It runs the second.
Resonance (8.6)	Controllers	8.6.2, 8.6.3	Run the gyre-swell example on two pumps at the ends of a tank 120 cm long and 60 cm wide, with water 50 cm deep, in anti-sync. Then add a pulse with a period of 1.3 s.	The swell's waves have a period of 1.334 s, within 1 ms, and build and fall over the swell's `period`. The controller warns about the pulse, whose period is within 10 % of the resonant period.
Caps over everything (10.2)	Controllers, pumps	10.2.2, 10.2.4	Run the reef example with a `speedLimitHigh` of 60 on a pump member, through its midday pulse and a storm, on a controller and as a pump program. Then give a pump whose endpoint's `minSpeed` is 30 % a member `speedLimitHigh` of 20.	The pump never runs faster than 60 %. The storm's random speeds above 60 % are held at 60 %. The controller warns about the second pump, which runs at 30 % wherever its pattern asks for more than 0 %.
No running below the floor (10.2)	Controllers, pumps	10.2.3	Run a sine from 0 to 100 % with a period of 60 s on a pump whose member's `speedLimitLow` is 25, on a controller and as a pump program, and then on a pump whose endpoint's `minSpeed` is 30.	The pump stops at the start of each period, and otherwise never runs below 25 %, or, on the second pump, 30 %.
Only what a controller can time (10.4)	Controllers, configurations	10.4.3	Assign the reef example, whose midday pulse has a period of 5 s, to a pump that the controller drives, and to one that runs its own program.	The controller refuses to drive the first with the pulse, and says why; the validator reports an error. The second runs the pulse.
Switches spared (10.5)	Controllers	10.5.2	Run a square with a period of 20 s on a `switch` whose `minSwitchTime` is 60 s. Then, 10 s after it switches on, start an interlock that stops it.	The switch changes no more often than every 60 s for the square. The interlock's stop is sent at once. The validator warns that the pattern is

PRE-DRAFT

Test	Applies to	Requirements	Method	Pass criteria
				faster than the switch can follow.
Pumps in step (11.2)	Controllers, pumps	11.2.1, 11.4.2, 15.2.7	Run the reef example on two pumps from different makers, one in sync and one in anti-sync, both running their own programs, and one also keeping in step by its maker's radio link, through the midday pulse, and record each pump's commanded speed, or its power with a fast meter.	With both pumps' clocks within 20 ms of their controller's, the anti-sync pump's pulses start half a period after the sync pump's, within 50 ms, throughout the entry.
Sequences (11.3)	Controllers, pumps	11.3.1	Run a pulse with a period of 6 s and a `pulseWidth` of 2 s on three pumps in sequence, two running their own programs and one driven by the controller with a period of 60 s and a `pulseWidth` of 20 s.	The pulses start 2 s apart, in the order of the pump members, within 50 ms; with the slower pulse, 20 s apart, within 1 s.
Clocks together (11.4)	Pumps	11.4.1	Run a pump with its controller for 7 days, on the module bus and over the Device Protocol, and compare their clocks every hour.	They never differ by more than 20 ms.
Transitions (12.1)	Controllers, pumps	12.1.1	Run the reef example across each entry of its schedule and the start and end of a trigger, and the gyre-swell example across its change from gyre to swell, on a controller where it can drive them and as pump programs.	Speeds match the reference implementation within 0.5 percentage points through each transition. The pump stops before it changes direction.
Triggers end by themselves (12.2)	Controllers	12.2.1, 12.2.3, 12.2.5	In the reef-system example, start the storm mode, and the feed mode 5 minutes later, and stop the controller 2 minutes after that. Restart it, start the feed mode again, and end it after 3 minutes. Start the storm mode again, and stop the controller at once. Then, during the night, restart the controller with its clock not valid.	The first time, the pumps run the feed pattern over the storm, and return to their schedule 10 minutes after the feed mode started, although the controller is stopped and the storm had time left. The second time, they return after 3 minutes. The storm pattern is sent with `revertAfter` 1200, the storm mode's duration, and the pumps return to their schedule 20 minutes after it started. The night pattern ends, because its condition is unknown.
Triggers end with their modes (12.2)	Configurations	12.2.8, 12.4.3	Validate a copy of the reef-system example whose storm trigger has a `maxDuration` of 1800 s, and one whose feeding trigger has none.	The validator warns about each trigger, which can outlast its mode.
Restore ramps (12.3)	Controllers, pumps	12.3.1, 16.5.1	Remove and restore a pump's power; start and end a maintenance mode that stops it; and replace its program. Do this with the pump running its own program, with the pump driven by the controller and `powerOnBehavior` `fail-safe`, and with a driven pump of Device Classes revision 0.3 whose `powerOnBehavior` is `last`. Then remove and restore the power, with no controller, of a pump of Device Classes revision 0.5 whose `powerOnBehavior` is `fail-`	Each time, the pump's speed rises from what it showed to its program's speed evenly over its `restoreRamp`, by default 30 s; after power-up, the pump that runs its own program rises from 0 %, the pump with `last`

Test	Applies to	Requirements	Method	Pass criteria
			`safe` and whose `failSafeState` runs it at 70 %, and of a pump that a latched stop holds.	ramps from 0 % to the speed it had over 30 s, and each driven pump rises from what it reports when the controller resumes control. The pump of revision 0.5 ramps from 0 % to 70 % over 30 s, and the latched pump stays stopped.
Gentle restore ramps (12.3)	Configurations	12.3.5	Validate a copy of the reef-system example whose display flow function has a `restoreRamp` of 2 s.	The validator warns about the restore ramp.
Night keeps the water moving (12.5)	Program documents	12.5.3	Validate a copy of the reef example whose night pattern runs at a constant 10 %.	The validator warns that the night pattern's mean speed is less than half the schedule's.
Smooth flow control (13.2)	Controllers	13.2.2, 13.2.5	Run the reef-system example's return pump at its constant flow; partly close a valve on its line, so that its flow falls by 20 %; then open it again.	The speed rises and falls by no more than 1 percentage point in any second, and the flow is within 5 % of its target within 120 s of each change.
Holding on a lost input (13.4)	Controllers	13.4.1	Disconnect the return pump's flow sensor in the reef-system example, and reconnect it 10 minutes later.	The pump keeps its speed within 1 percentage point. `input-unavailable` is raised while the sensor is disconnected, and control resumes when it is back.
Low flow raises an alarm (13.5)	Controllers	13.5.1, 13.5.2	Restrict the intake of the reef-system example's return pump until its flow falls below its target with the pump at its `max`, and then below its `minFlow`; start and end the feed mode; then free the intake.	`flow-control-limit` is raised after the pump has run at its `max` for 300 s, and `flow-low` 60 s after the flow falls below `minFlow`, each with the pump's output as its subject. The feed mode's stop and restart neither raise nor clear `flow-low`. Both clear once the intake is free.
The same program on a pump (15.3)	Pumps	7.9.3, 15.3.1	Write the pump program of each entry of `pumps` in `test-vectors.json` to a pump, set its clock to each sample time, and read its commanded speed and direction.	Every speed is within 0.5 percentage points of the vector, and every direction is the same.
Writing pump programs (15.2)	Controllers	15.2.1, 15.2.2, 15.2.3	Assign the reef example to a pump over the Device Protocol and to one on the module bus, then change one pump's `scale`.	Within 60 s, the network pump holds its pump program as text, and the module a reference whose file the controller serves. Both hold their new programs within 60 s of the change.
Checking a program file (15.2)	Controllers, pumps	15.2.4, 15.2.6	Write a pump whose `failSafeAction` is `continue` a reference whose file has another hash, then one whose file is cut short, then a	After each of the first two, the pump keeps its previous program and reports its

PRE-DRAFT

Test	Applies to	Requirements	Method	Pass criteria
			correct one, while its schedule runs a pulse whose `min` is 0.	previous `program`, and the controller raises `program-not-loaded` 60 s later, with the pump's output as its subject; meanwhile it does not drive the pump with the pulse. After the third, the pump runs the new program, and the alarm clears.
Returning to the program (15.4)	Controllers, pumps	15.3.3, 15.4.1, 15.4.2, 15.4.3, 15.4.4, 15.4.5, 15.4.6	Stop a pump that runs its own program with a maintenance mode, and end the mode; start a feed mode that the function's trigger follows; send a `pattern` that its program does not have; and cut contact for longer than the fail-safe timeout, then restore it. Then assign the program, with `runOn pump`, to a pump of Device Classes revision 0.1 that does not report `org.openaquariumproject.program-running`. Last, start and end the feed mode with a pump of Device Classes revision 0.2 that runs its own program.	Each time, the controller sends `runProgram`, and the pump returns to its program, or runs the feed pattern, with the transition the command gives; it rejects the unknown pattern. `programRunning` is `false` while the pump is stopped, and `true` after. The controller drives the pump of revision 0.1 itself. The pump of revision 0.2 is sent the feed pattern with a `revertAfter` 10 s longer than the feed mode, and, when the mode ends, `runProgram` with a `transition` of 10 s, and changes back over 10 s.
Running without a controller (15.3)	Pumps	15.3.2	Stop the controller of two pumps that run the reef example by themselves, in sync and in anti-sync, with `failSafeAction continue`, for 24 hours. Repeat while an interlock's latched stop holds one of them.	Each pump follows its schedule and patterns by its own clock, which stays valid, matching the reference implementation, and reports `communication-lost`. The latched pump stays stopped.
Return pumps never continue (16.1)	Controllers	16.1.1	Bind a return pump whose endpoint has `failSafeAction continue`; then try to set `continue` on a return pump with the controller.	The controller warns about the first, and refuses the second. The validator reports an error for the first pump's device description.
Return pumps refuse continue (16.1)	Pumps	16.1.1	Set a pump's `loadCategory` to `return-pump`, and write `failSafeAction continue` to it.	The write is rejected, and `failSafeAction` keeps its value.
Return pumps keep running (16.1)	Controllers, configurations	7.2.8, 16.1.2, 17.2.4	Assign the reef example to a return pump; then a program whose only pattern is `constant` at 0 %; then one whose `constant-flow` pattern has a `min` of 0.	The controller refuses each, and says why. The validator reports an error for each.
No stop kept by a failed	Controllers	16.2.1	Assign a pulse whose `min` is 0 to a pump that the controller drives, whose `failSafeAction` is `continue`.	The controller refuses, says why, and proposes `run`.

PRE-DRAFT

Test	Applies to	Requirements	Method	Pass criteria
controller (16.2)				
Stop before reversing (16.3)	Pumps	16.3.1	Send a reversible pump that runs forward at 80 % a `set` with `direction` `reverse` and `speed` 80, and measure its motor speed.	The motor stops, and stays stopped for at least the pump's `reversePause`, or its documented pause, before it turns the other way.
The water keeps moving without a clock (16.5)	Controllers, pumps	7.5.6, 16.5.2, 16.5.3	Power up a pump that runs its own program with its clock not valid and no controller. Restart a controller with its clock not valid, with a pump that it drives.	Neither pump stops. Each runs the pattern of the entry in effect when it last had the time, or the first entry, the pump reports `clock-invalid`, and each returns to its schedule with its restore ramp once the clock is valid.
Invalid configurations	Controllers, configurations	7.2.3, 7.2.4, 7.2.6, 9.1.1, 10.2.5, 13.1.4	Import configurations that each break one rule for documents: a flow function whose `outputs` list an output that has no pump member; a flow function with `onUnavailable`; a function that fails its schema; a swell at `resonance` without `dimensions`; a schedule with two entries at the same time; a `constant-flow` pattern on a pump without a `flowInput`, and on a pump whose `runOn` is `pump`; a pump member whose `speedLimitLow` is above its `speedLimitHigh`; and one with both `speedLimitHigh` and `maxSpeed`.	The controller and the validator reject each one, saying why, and the running configuration is unchanged.
Flow in the control process	Controllers	7.1.4	With a pump the controller drives, whose fail-safe configuration is `run`, stop the controller's evaluations while its communication task keeps running.	The controller stops supervising the pump, which reaches its fail-safe state, running, within its `failSafeTimeout` plus 2 s.
Local time on a pump	Pumps	7.5.3	Compute every entry of `patternTime` in `test-vectors.json` on a pump that runs its own program, with its POSIX `TZ` string, and run the gyre-swell example's pump program across the end of daylight saving time of 25 October 2026.	Every pattern time matches, and the pump's speeds match the vectors across the change.
What the owner sees	Controllers	7.8	With the reef-system example running, look at each flow function in every app the controller offers, during the feed mode, with the return pump's flow sensor reading low and a wave pump reporting `stall`.	Each app shows every item of Section 7.8: the program, each pump's role and where it runs, the pattern each runs and why, its speed and what holds it, the return pump's target, flow, speed, and limit, the turnover marked as estimated, the patterns a pump cannot run, and the pump's `stall`.
Pattern types a product does not implement	Controllers, pumps	8.3.1, 15.1.4, 20.1.4	Import a configuration whose flow function's program has a pattern of the type `org.example.wave` in its schedule, and write the same program, as a pump program, to a pump.	The controller tells the owner before it accepts the document, raises `unsupported`, and runs none of the function's

PRE-DRAFT

Test	Applies to	Requirements	Method	Pass criteria
				pumps, which take their fail-safe behavior. The pump rejects the program and keeps its previous one. The validator warns about the type.
Speeds by the steps	Controllers	10.3, 10.4.1	Compute every entry of `pumps` in `test-vectors.json` on a controller, and drive a pump that runs no program of its own with the back-wall entry.	Every speed is within 0.5 percentage points of the vector, and the pump's commanded speed is within 1 percentage point of the vector at some moment within 1 s.
Clock corrections	Pumps	11.4.5	Watch when a pump that runs its own program in a group corrects its clock, on the module bus, and over the Device Protocol with a controller that sends only `control.supervise` every 20 s, with and without time exchanges by the pump.	On the module bus, the pump corrects its clock every second. Over the Device Protocol, a pump that makes time exchanges makes one at least every 60 s and corrects its clock from it, and one that makes none corrects it at every `control.supervise`.
Pauses shorter than a pump's	Controllers, pumps	7.9.2, 16.3.2	Assign the gyre-swell example to a reversible pump whose `reversePause` is 5 s, on a controller and as a pump program, and validate a device description of the pump with that program.	The controller warns that the gyre's `pause` of 2 s is shorter than the pump's `reversePause`. The pump stays stopped for 5 s at each reversal, and then follows the gyre, lagging the reference meanwhile. The validator warns about the pump program.
Export and import	Controllers	17.2.1	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.
What an import shows	Controllers	17.2.3, 17.2.5	Import the reef example for a group of a pump that runs its own program and a pump without the `wave` feature, into a configuration whose flow function has no `dimensions`.	Before it runs, the controller shows the pattern of each entry, the suggested and proposed roles, the proposed triggers, that the pump without the `wave` feature cannot run the midday pulse, and the expected turnover; it asks for the tank's dimensions, and waits for the owner to confirm.
Authors and licenses kept	Controllers	17.3.2	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.

PRE-DRAFT

21 Security Considerations

1. **Programs are data.** A program contains no code, and a product parses it with the limits the schema sets: 32 patterns, 24 entries, 1440 points, 16 pumps, 32 triggers, and pump programs of at most 65 536 bytes. A pump 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; starting a mode that a trigger follows, or ending one other than an emergency mode, the `start-mode` operation; and ending an emergency mode the `end-emergency-mode` operation [OAP-SA Table 6]. Pumps accept pump programs, and `runProgram`, only from their controller [OAP-FS Section 7.5].
3. **Shared programs.** A program from someone else can ask for strong flow, waves at a tank's resonance, or long storms. It can never exceed the owner's caps, run a wave pattern on a return pump or stop one, or reverse a pump that cannot reverse, and the controller shows it, and the turnover it expects, before it runs (Section 17.2).
4. **Program files.** A pump reads a program file only from its controller, and checks its size and hash before it runs it (Section 15.2.4). 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. **Flow as a hazard.** Someone who could change a program could stop circulation or build resonant waves. Authorization, caps, the rules for return pumps, the owner's confirmation, fail-safe behavior, and the alarms for low flow limit what they could do.
6. **Privacy.** A flow program carries no location, and nothing about the owner but what the owner puts in it (Section 17.3).

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 20.5, including a pump that runs its own program and a controller.	Not started	
Pumps from two makers, each running its own program, have kept in step in sync and in anti-sync, and a program written by one implementation has run on a pump of another.	Not started	
Every normative reference is publicly available.	Done	
The checks file, patterns 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 the night flow of Section 12.5 (Appendix D, issues 12 and 16).	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 `Notices.md`.	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, gentle random flow all day.
- `reef.json`: a reef program for two wave pumps, one in sync and one in anti-sync: reef-crest random flow through the day, alternating pulses at midday, a nutrient export at the tank's resonant period in the evening, and gentle random flow at night, with patterns for feeding, storms, and battery power that a controller's triggers use.
- `gyre-swell.json`: a program for two reversible cross-flow pumps facing each other: a gyre that turns the water one way and then the other every ten minutes, a swell that builds standing waves at the tank's resonant period in the early afternoon, and a slow gyre at night.
- `tidal.json`: a semidiurnal tide of 12 hours 25 minutes 12 seconds, as a custom waveform for two pumps in sequence, which drifts through the days as a tide does.
- `reef-system.json`: the Automation configuration of Section 6.4: two network wave pumps that run the reef program by themselves, in sync and in anti-sync, with feeding, storm, and battery triggers and the tank's dimensions; a back-wall pump on a 0–10 V output, driven by the controller with a slow sine and a floor of 30 %; and a return pump that holds 2400 L/h from a flow sensor by day and 1800 L/h at night, never below 40 %, with a minimum flow, which the feed mode stops.
- `pump-network.json`: a network wave pump's device description of Device Classes revision 0.5, with the pump program of the right-hand pump of the reef-system example written as text.
- `pump-module.json`: a module bus pump driver's device description of Device Classes revision 0.5, with a return pump whose fail-safe state is running, and a reversible cross-flow pump with a reference to its pump program: the right-hand pump of the gyre-swell example, in anti-sync, in a tank 90 cm long with water 45 cm deep.

Appendix C — Commercial Features (Informative)

How this Specification covers the flow features of the controllers and wave pumps of seven widely used makers.

Table 30 — Commercial flow features

Feature	Found in	This Specification
A library of patterns	Every product; from five patterns to eleven, with names of their own, often of reef zones or surf breaks; none publishes its equations	Eleven patterns, each defined by an equation (Section 8.4)
Random flow	Every product; reef-crest and lagoon modes are random flow over wide or narrow ranges	`random`, with its range and step, from a published generator (Section 8.5)
Pulses and square waves	Every product; "pulse duration" means the time on in some, the time on and off in others, and a cycle of ramps in one	`pulse` and `square`, with `period` always the whole repetition (Section 8.3.4)
Sine and triangle waves	One controller's library, and one's choice of sine or square waves	`sine` and `triangle` (Section 8.4)
Waves tuned to the tank	Short pulses of 0.2 to 2 s, found by sweeping until the tank resonates; one maker's tuning is a pulse at half the tank's period	`swell` and `nutrient-export` at `resonance`, computed from the tank (Section 8.6)
Gyres and alternating flow	Cross-flow pumps that reverse every 2 s to 8 h; one pump's reverse flow is about a third of its forward flow	`gyre`, with its own reverse speed and a stop before each reversal (Sections 8.4 and 16.3)
Nutrient export and tidal swell	Two-phase modes: growing waves, then a surge that sweeps detritus to the overflow	`nutrient-export`, with its surge (Section 8.4)
Custom waveforms of up to 24 hours	One controller; another chains three modes; one counts its tidal cycle as 24 h 48 min	`custom`, of up to 25 hours (Section 8.7)
Sync and anti-sync	Every product; anti-sync is the opposite speed in some, half a period later in others; master and slave pumps over a radio link or a cable	Roles, from one clock, with anti-sync defined for each pattern (Section 11.2)
Sequences	One controller's sequences and surges; one controller's four-corner pattern; one maker's timed delay	`sequence` (Section 11.3)
Sync accuracy	One maker claims a thousandth of a second; no other publishes a figure	50 ms between pumps, 20 ms between clocks (Section 11.4)

Feature	Found in	This Specification
Constant flow from a flow sensor	One controller	`constant-flow` (Section 13)
Feed modes	Every product; 5 to 30 minutes; some stop the pumps, others slow them	Automation feed modes, with stops or triggered patterns (Section 12.4)
Night modes	One maker's 50 % for 10 hours; others use schedules or a light sensor	Entries of the schedule, or triggers on conditions (Section 12.5)
Storms	One controller's thunderstorm setting; one controller's example storm of 10 minutes each hour	Storm patterns, started by a mode (Section 12.6)
Battery operation	Pumps slowed to 10 % to 50 %, or to a set constant speed	Power-loss modes, with levels or a battery pattern (Section 12.7)
Turnover index	One controller adds up its wave pumps' flows over the display's volume, and grades it from 0 to 10	Display and return turnover, measured or estimated (Section 14)
Running without the controller	Most pumps keep their own schedule; some fall back to a set speed or switch off	Pump programs with `continue`, and return pumps that run (Sections 15 and 16)
Program files	Cloud or app only; none reads another maker's	Program documents (Section 17)

Appendix D — Open Issues (Informative)

Issues 1 to 16 were open in revision 0.1, and issue 18 arose in revision 0.2. The Project Lead decided issue 17 on 15 October 2026, and issues 1, 14, and 18 on 16 October 2026, accepting the recommendation in each. On 17 October 2026 the Project Lead froze the scope of revision 1.0; revision 0.3 uses what Fail-Safe Behavior revision 0.3 and Device Classes revision 0.4 added for issue 18, and resolves issues 2, 4, and 5, which other specifications settled. Issue 19 records the choices that revision 0.3 made where those decisions left them open, which the Project Lead accepted on 18 October 2026. Issues 8 to 11, 13, and 15 are deferred beyond revision 1.0, and are tracked outside this Specification. Issues 12 and 16 stay open until experienced aquarists, 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. **Return pumps and wave patterns.** *Decided by the Project Lead on 16 October 2026; resolved in revision 0.2: the limit stays.* Section 16.1 limits return pumps to steady patterns that keep them running, and recommends 30 s for their changes, because wave patterns make sump levels swing and overflows surge. Some owners do vary their return pumps slowly, for example in a slow sine over hours, and one controller allows only ramps on its return pump. The Project Lead decided to keep the limit (Section 16.1.2), and, once owners' experience with constant flow shows a need for slow patterns, to allow in a later revision a `sine` with a `period` of at least an hour and a `min` above 0 %.
2. **Function type and mode kinds.** *Resolved in revision 0.3, with Automation revision 0.3.* The Automation specification carried flow functions under a provisional extension type, `org.openaquariumproject.flow`. Automation revision 0.3 lists `flow` among its function types (its Section 11.10 and Table 14), describes a flow function's demand on a pump, its program, and that the function returns a pump to its program when a higher layer's demand ends (its Section 8.3.2 and Table 25), points its `circulation-pump` preset to the new type, and lets its `return-pump` preset use it for constant flow (its Table 28; Appendix D, issue 27). Section 7.2 now uses the type, and a controller still reads the name of revisions 0.1 and 0.2 (Section 7.2.9). Storms stay the extension kind `org.openaquariumproject.storm` (Section 12.6), and revision 1.0 adds no standard mode kinds for storms or the night.
3. **The `pump` 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 pump program or a reference to one, reported only once it is loaded and checked; `runProgram` and `programRunning`; `syncGroup` as the pump's group; `set`'s `rampTime`; `reversePause`, with a stop before every reversal; `flowRateSource` and `flowCurve`; the device attribute `clockUncertainty`; and the device fault `clock-invalid` (its Section 9.3 and Appendix D, issue 21). Revision 0.3 added ramps on reverts (its Section 7.4.12), which Section 12.2.5 relies on, `runProgram` that keeps a pending revert's target (its Section 7.4.6), and the ramp up to a restored speed (its Section 8.3.3). Revision 0.4 added ramps to fail-safe states that run a pump (its Sections 8.3.5 and 8.3.6), and revision 0.5 latched stops (its Section 7.4.13). Sections 7.4, 7.6, 10.4, 11.4, 13.1, 14.2, 15, and 16 now use them, and keep

`org.openaquariumproject.run-program` and `org.openaquariumproject.program-running` only for pumps of revision 0.1 (Section 15.4.4). Its issue 9, on groups, is answered by Section 11: a group is the pumps of one flow function, which keep in step by their clocks and 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/flow/` files while a pump holds a reference to one, and the Device Protocol specification that writing a reference offers its file to the pump, as both are asked to for lighting programs (Section 15.2). 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 15.2.3 now cites both, and the paths are no longer provisional.
5. **Clock accuracy over the Device Protocol.** *Resolved in revision 0.3, with Device Protocol revision 0.3 and Module Bus revision 0.4.* A network pump set its clock from `control.supervise` without knowing how long the message took, so this Specification asked the Device Protocol for a time exchange that measures the round trip, and the accuracy a device can expect (Section 11.4.4). Device Protocol revision 0.3 adds the time exchange `control.time`, from which a device knows its controller's time within half the shortest round trip it measures, plus 1 ms: about 5 ms on Ethernet, or on Wi-Fi without power saving (its Sections 12.3.4 to 12.3.6; Appendix C, issue 29). Module Bus revision 0.4 keeps a module's clock within 1 ms of network time (its Section 9.4.6; Appendix C, issue 15). Section 11.4 now has a network pump in a group set its clock with the time exchange, and correct it at least every 60 s, and cites the accuracy of each binding, well within the 20 ms that a pump should keep.
6. **Turnover and tank dimensions in the Data Model.** *Resolved in revision 0.2.* Data Model revision 0.2 adds the parameter `turnover`, in `1/h`, and a Vessel's `dimensions` (its Sections 8.2 and 9.2). Section 14.3.2 now records turnover as `turnover`, and reads the provisional extension of revision 0.1 as it, and a controller may propose a flow function's `dimensions` from its vessel's (Section 8.6.1).
7. **Fail-safe configuration of driven pumps.** *Resolved in Fail-Safe Behavior revision 0.2 (its Appendix C, issue 12).* Fail-Safe Behavior Table 3 recommends `run` for circulation pumps, and allows `continue` for a pump that runs its own flow program. Its notes now say that a controller never drives a pump that continues with a pattern that stops it (Section 16.2), and that a pump whose `loadCategory` is `return-pump` should reject `continue` (Section 16.1.1).
8. **Night flow.** *Open: to be confirmed with experienced reefkeepers.* Section 12.5 asks night patterns to keep at least half of the day's mean speed, because oxygen is lowest at night; no study of night flow in aquariums was found, and the share should be confirmed with experienced reefkeepers.
9. **Smaller pumps.** *Decided by the Project Lead on 16 October 2026; resolved in revision 0.2: one conformance level.* A pump must accept 32 patterns, 1440 points, and programs of 65 536 bytes. A smaller conformance level, with a controller simplifying programs for such pumps, would let them run simpler programs by themselves, but the pumps of a group would then no longer run the same program in step, and every controller and test would have to handle two levels. The largest pump program takes less than 32 kB once parsed into numbers, which a pump can keep in its flash memory, so the limits ask for design rather than for memory. The Project Lead decided on one conformance level, as for lights (Lighting Programs Appendix D, issue 14): a pump that cannot run programs up to the limits is driven by its controller (Section 15.1.5).
10. **Numbers to confirm.** *Open until the bench tests of the breadboard controller and prototype pumps, and experienced aquarists, confirm them.* The timing tolerances, the transition and restore ramps, the gyre's pause, the switch time, the controller's limits, the flow control's rate and delays, and the night and storm recommendations are provisional. So are the numbers that revision 0.2 adds or relies on: the limit of one command a second to a pump (Section 10.4.3), the clock correction at least every 60 s (Section 11.4.5), and the 30 s over which a pump without a program ramps up to a restored speed, which Device Classes Section 8.3.3 sets. Revision 0.3 adds the 10 s that a restore ramp should last at least (Section 12.3.5), and relies on the 30 s over which a pump ramps to a fail-safe state that runs it, which Device Classes Section 8.3.5 sets.

Decided by the Project Lead

17. **Ramping after power-up.** *Resolved in revision 0.2.* The Project Lead decided on 15 October 2026 that a pump restoring a speed after power-up ramps up to it, rather than stepping (Fail-Safe Behavior Appendix

C, issue 15). Section 16.5.1 now says so: a pump that runs its own program ramps up with its restore ramp, and one that restores a speed without a program over 30 s (Device Classes Section 8.3.3), so that Section 12.3.1 holds after power-up. Issue 18 asks about pumps that enter a running fail-safe state after power-up.

The following issue arose in revision 0.2:

18. **A running fail-safe state after power-up.** *Decided by the Project Lead on 16 October 2026; resolved in revision 0.3, with Fail-Safe Behavior revision 0.3 (its Section 9.1; Appendix C, issue 22) and Device Classes revision 0.4 (its Section 8.3.5; Appendix D, issue 46).* A pump the controller drives has the recommended `fail-safe` after power-up and a `failSafeState` that runs it (Table 24), so after power-up it entered that state at once [OAP-FS Section 9.1], stepping from 0 % to its fail-safe speed, and its controller could ramp it only once it resumed control (Section 16.5.1). The Project Lead decided that an endpoint that enters a `failSafeState` that runs a pump after power-up is to ramp up to it, as one that restores a speed does (Device Classes Section 8.3.3), and one that lights a light is to fade up to it. The same revisions also have a pump ramp to a running fail-safe state that it enters at any other time, such as when its controller fails, from the speed it runs at (Fail-Safe Behavior Section 8.1.6; Device Classes Section 8.3.6; their Appendix C, issue 24, and Appendix D, issue 48). Section 16.5.1 now requires the ramp after power-up of a pump of Device Classes revision 0.4 or later, and Sections 15.3.2 and 16.6, and Table 24, describe the ramp when its controller fails; a pump of an earlier revision steps to its fail-safe speed, and its controller ramps it once it resumes control.

The following issue records the choices of revision 0.3:

19. **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 triggered pattern is sent with `revertAfter` until the earlier of its trigger's `maxDuration` and the end of the mode the trigger follows, and a trigger that follows a mode with a `duration` should last no longer, with a validator's warning (Sections 12.2.5 and 12.2.8);
    - to a pump of Device Classes revision 0.1 or 0.2, a trigger's `revertAfter` is longer by the trigger's `transition`, so that its controller ramps it back rather than letting it step (Section 15.4.5);
    - the time a pump stays stopped for its `reversePause` beyond a pattern's stop is an exception to Section 7.9, and a controller should warn about a gyre that pauses for less, as a validator does (Sections 7.9.2 and 16.3.2);
    - a network pump in a group corrects its clock at least every 60 s from a time exchange, or, without one, from each `control.supervise` (Section 11.4.5); and
    - a restore ramp should last at least 10 s (provisional), and a validator warns about a shorter one (Section 12.3.5).

PRE-DRAFT