



OPEN AQUARIUM PROJECT SPECIFICATION

Device Protocol

PRE-DRAFT

Revision	0.2
Status	Pre-Draft. A candidate document submitted for consideration.
Date	2026-10-16
Identifier	device-protocol
Authors	Trent Scholl, Project Lead, Open Aquarium Project
Editors	Trent Scholl
License	Community Specification License 1.0

Pre-Draft. This is a proposed initial draft. The Working Group has not approved it, and it will change. Several choices are provisional until the Project Lead decides them (Appendix C). Comment by opening an issue with the `device-protocol` 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 protocol definitions, JSON Schema, and example files published with this Specification are licensed under the Apache License 2.0 (`LICENSES/Apache-2.0.txt`).

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

Notes:

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

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

2 Compliance with Open Aquarium Tenets

2.1 Openness

The Specification is published under the Community Specification License 1.0, and its protocol definitions, schema, and examples under the Apache License 2.0. It is built entirely from free, public standards: multicast DNS and DNS service discovery, TLS 1.3, WebSocket, JSON, and JSON-RPC 2.0. No membership, certification, vendor identifier, or fee stands between a maker and a working device, controller, app, adapter, or relay. On Thread networks it uses only IP and the public Service Registration Protocol, not the Thread specification itself. A validator and example message exchanges are published with it, so anyone can check an implementation.

2.2 Interoperability

Any conforming device works with any conforming controller and app, whoever made them. The protocol carries the Device Classes model unchanged, so a pump or a doser is the same endpoint whether it is a module on the wired bus or a device on the network, and a controller needs no driver for a particular product. Apps from different makers can claim, share, and update devices from any maker, because pairing and firmware updates are carried the same way for everyone. Adapters bring commercial devices into the same model, and any app works with any relay.

2.3 Local Control

Everything in this Specification works on the local network with no internet connection, cloud account, or vendor. Discovery runs on the local network, and every session is authenticated by keys that the owner paired. Local and cloud access work at the same time: a cloud service is one more party with a role the owner chose,

and connecting it never turns local access off (Section 15). A controller can host its own Wi-Fi network, so that its devices do not depend on the home router (Section 14.1).

2.4 Safety

The protocol carries the supervision that the Fail-Safe Behavior specification depends on, so devices take their fail-safe behavior when their controller, the network, or a session fails (Section 12). Only messages from the controller's own key count as contact, and only one controller at a time can command a device. Commands that stop equipment are always accepted, never held back by a limit on requests, and handled first (Sections 9.5 and 11.3). A command whose response was lost is never repeated blindly, so that a lost connection never doses twice (Section 12.4). Devices report changes to outputs and safety inputs within 100 ms, and controllers notice a lost device within 30 s. Sleepy devices, which run on batteries and have only inputs, are the exception: they report changes of `leak`, `wet`, and `closed` within 1 s, but other changes up to their reporting interval late, and a controller notices that one is lost only after three times that interval, which for one that a safety function uses should be no more than 45 minutes (Section 14.3). Commercial devices that depend on a cloud service, or whose fail-safe behavior depends on an adapter, are marked, and are kept out of functions that need equipment to stop (Section 16.4). A radio link is never the only path for a safety function: Wi-Fi, Thread, and the links to bridged devices are radio links, and each safety function keeps a path that uses none, as Fail-Safe Behavior Section 13.4 requires [OAP-FS Section 13.4].

2.5 Sustainability

Makers can add the protocol to existing Wi-Fi products with a firmware update, and adapters let owners keep commercial equipment that has no open interface. Because devices never depend on their maker's service, they keep working after the maker stops supporting them. The protocol fits common single-chip Wi-Fi microcontrollers, so devices need no extra processor to take part.

Revision History

Revision	Date	Description	Authors
0.1	2026-10-14	Initial Pre-Draft.	Trent Scholl
0.2	2026-10-16	Sleepy devices report changes of <code>leak</code> , <code>wet</code> , and <code>closed</code> within 1 s, and Section 2.4 says how they differ; stops are carried out first and never refused for a limit on requests; the commands that end an operation carry no <code>revertAfter</code> , and operations that <code>device.stop</code> ends never resume; <code>device.stop</code> and the module bus's <code>COMMAND_EMERGENCY_STOP</code> compared; firmware information in the device attributes of Device Classes revision 0.2, with <code>org.openaquariumproject.firmware</code> read only from devices of revision 0.1, and one method, <code>device.write</code> with <code>install-firmware</code> , that sets the update channel; <code>firmware.revert</code> refused while dosing, filling, or in a water change, and required of devices; a device that gets a new controller puts every endpoint in its fail-safe behavior; the outputs of a lost device unknown until their <code>failSafeTimeout</code> plus 2 s; default maximum ages from the <code>maxInterval</code> a device returns; <code>control.supervise</code> , with the time, at least every 30 s whatever else the controller sends; adapters that lose their controller make their bridged equipment fail safe; a link to a bridged device a radio link, and safety functions as Fail-Safe Behavior defines them; access rules of their own for <code>control.take</code> and <code>pairing.invite</code> ; a claimed device's controller mark from the Prover's <code>kind</code> ; emergency access on a device network only as Security and Access allows; devices on a failed controller's device network; program files offered by writing a reference; resuming control only by commands that change an output; <code>specRevisions</code> and the TXT key <code>v</code> at 0.2, with the subprotocol <code>oap.0.1</code> kept; [OAP-MB] a normative reference; examples of Device Classes revision 0.3; a test procedure, mapped to its requirements, for every requirement; references cited by tag; one sentence per line; the decided issues resolved, and every other issue given a status; the Project Lead's decisions of 16 October 2026 applied: a warning, not a refusal, when run-on-loss equipment is bound to a bridged device that depends on the cloud or on its adapter, which ends the provisional status of Section 16.4, emergency access on a device network as restricted, and no change to the module bus's emergency stop; the equipment's own timeout, <code>deviceTimeout</code> , in <code>org.openaquariumproject.bridge</code> ; and sleepy devices that a safety function uses asked	Trent Scholl

Revision	Date	Description	Authors
		to report at least every 900 s, with the time a silent failure would go unnoticed shown to the owner.	

Contents

1	License	2
2	Compliance with Open Aquarium Tenets	2
2.1	Openness	2
2.2	Interoperability	2
2.3	Local Control	2
2.4	Safety	3
2.5	Sustainability	3
	Revision History	3
3	Scope	7
4	References	7
4.1	Normative References	7
4.2	Informative References	8
5	Terms, Definitions, and Conventions	9
5.1	Requirements Language	9
5.2	Terms and Definitions	9
5.3	Notation	9
6	Introduction	10
6.1	Purpose	10
6.2	Design Principles	10
6.3	Overview	11
7	Networks and Identities	12
7.1	Networks	12
7.2	Parties	12
7.3	Device Identifiers	12
8	Discovery	12
8.1	Service	12
8.2	TXT Record	13
8.3	Subtypes	13
8.4	Thread	13
8.5	Finding a Party	13
9	Sessions	14
9.1	Connections	14
9.2	WebSocket	14
9.3	Messages	14
9.4	Opening a Session	15
9.5	Requests and Responses	16
9.6	Errors	16
9.7	Value Encoding	17
9.8	Methods	17
9.9	Extensions	20
9.10	Keeping Sessions Alive	20
10	Authentication and Pairing	20
10.1	Trust	20
10.2	The Pairing Exchange	21
10.3	Codes	21
10.4	Setup Code Labels	22
10.5	Invitations	22
10.6	Pairing Lists	23
10.7	Password Sign-In	23
10.8	Setup over Bluetooth Low Energy	23
11	Carrying the Device Classes	24
11.1	Device Descriptions	24
11.2	Reading and Writing Attributes	24
11.3	Commands	25
11.4	Stopping and Identifying a Whole Device	25
11.5	Subscriptions and State Reports	25
11.6	Events	26

11.7	Changes to Configuration	27
11.8	Summary	27
12	Control and Supervision	27
12.1	The Controller of a Device	27
12.2	Contact and Supervision	28
12.3	Time	28
12.4	Resuming Control	28
12.5	Loss of a Device	29
12.6	Summary	29
13	Managing Units	30
13.1	Firmware Updates	30
13.2	File Transfer	30
13.3	Network Settings	30
13.4	Restarting, Resetting, and Erasing	31
13.5	Audit Log and Consent	31
14	Network Arrangements	31
14.1	A Controller's Own Wi-Fi Network	31
14.2	Devices That Connect to Their Controller	32
14.3	Battery-Powered Devices on Thread	32
15	Local and Cloud Access	32
15.1	Access at the Same Time	32
15.2	Services	33
15.3	Remote Access	33
15.4	Relay Protocol	33
16	Adapters for Commercial Devices	34
16.1	Adapters and Bridged Devices	34
16.2	How a Bridged Device Is Reached	34
16.3	Fail-Safe Behavior Through an Adapter	35
16.4	Keeping Dependent Equipment out of Safety Functions	35
17	Conformance	35
17.1	Devices	35
17.2	Controllers	36
17.3	Apps and Services	36
17.4	Adapters	36
17.5	Relays	36
17.6	Machine-Readable Definitions and Tools	36
17.7	Test Procedures	36
18	Security Considerations	40
Appendix A — Approval Checklist		41
Appendix B — Message Exchanges (Informative)		42
B.1	Adding a Network Doser	42
B.2	Loss of Contact and Recovery	42
B.3	A Cloud-Only Pump Behind an Adapter	43
B.4	Remote Access Through a Relay	43
Appendix C — Open Issues (Informative)		43

3 Scope

This Specification defines how controllers, network devices, and apps discover and talk to each other over IP, on Ethernet, Wi-Fi, or Thread. It defines:

- discovery with multicast DNS and DNS service discovery, and with the Service Registration Protocol on Thread;
- sessions: connections, their protection, and the format of messages;
- how the pairing of the Security and Access specification is carried, and the format of setup codes;
- how the Device Classes model is carried: descriptions, attributes, commands, state, events, and faults;
- supervision, contact, loss of a device, and one controller at a time, as the Fail-Safe Behavior specification requires;
- firmware updates, network settings, and other management of devices;
- a Wi-Fi network that a controller hosts for its devices, devices that connect to their controller, and battery-powered devices;
- local and cloud access at the same time, and the relay protocol for remote access;
- adapters for commercial devices; and
- conformance requirements and test procedures.

This Specification does not define:

- the device model, which the Device Classes specification defines;
- fail-safe behavior, which the Fail-Safe Behavior specification defines;
- roles, permissions, keys, and cryptographic protocols, which the Security and Access specification defines;
- the functions a controller offers its apps, such as history, automation configuration, modes, overrides, alarms, and users, which a separate specification is proposed to define (Appendix C, issue 1);
- an MQTT binding or Home Assistant discovery (Appendix C, issue 2);
- how devices appear in Matter, which the Matter Mapping specification defines;
- the wired module bus, which the Module Bus specification defines; or
- the format of firmware images.

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 6762]** Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, February 2013.
- **[RFC 6763]** Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, February 2013.
- **[RFC 9665]** Lemon, T. and S. Cheshire, "Service Registration Protocol for DNS-Based Service Discovery", RFC 9665, June 2025.
- **[RFC 6455]** Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.
- **[RFC 9110]** Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, June 2022.
- **[RFC 9112]** Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, June 2022.
- **[RFC 8259]** Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, December 2017.
- **[RFC 7493]** Bray, T., Ed., "The I-JSON Message Format", RFC 7493, March 2015.
- **[JSON-RPC]** JSON-RPC Working Group, "JSON-RPC 2.0 Specification", 2013-01-04, <https://www.jsonrpc.org/specification>.
- **[RFC 3339]** Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.
- **[RFC 4648]** Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

- **[RFC 5869]** Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, May 2010.
- **[RFC 9266]** Whited, S., "Channel Bindings for TLS 1.3", RFC 9266, July 2022.
- **[RFC 9383]** Taubert, T. and C. Wood, "SPAKE2+, an Augmented Password-Authenticated Key Exchange (PAKE) Protocol", RFC 9383, September 2023.
- **[SP 800-38D]** Dworkin, M., "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", NIST Special Publication 800-38D, November 2007.
- **[BT Core]** Bluetooth SIG, "Bluetooth Core Specification", Version 5.0 or later, <https://www.bluetooth.com/specifications/specs/>; anyone can read it, but its notice prohibits its use by anyone who is not a member of the Bluetooth SIG (Appendix C, issue 9).
- **[JSON Schema]** "JSON Schema: A Media Type for Describing JSON Documents" and "JSON Schema Validation", draft 2020-12, <https://json-schema.org/draft/2020-12>.
- **[OAP-DM]** Open Aquarium Project, "Open Aquarium Data Model", Revision 0.3, [specs/data-model/spec.md](https://openaquariumproject.org/specs/data-model/spec.md).
- **[OAP-DC]** Open Aquarium Project, "Open Aquarium Device Classes", Revision 0.3, [specs/device-classes/spec.md](https://openaquariumproject.org/specs/device-classes/spec.md).
- **[OAP-FS]** Open Aquarium Project, "Open Aquarium Fail-Safe Behavior", Revision 0.2, [specs/fail-safe-behavior/spec.md](https://openaquariumproject.org/specs/fail-safe-behavior/spec.md).
- **[OAP-SA]** Open Aquarium Project, "Open Aquarium Security and Access", Revision 0.2, [specs/security-and-access/spec.md](https://openaquariumproject.org/specs/security-and-access/spec.md).
- **[OAP-AU]** Open Aquarium Project, "Open Aquarium Automation", Revision 0.2, [specs/automation/spec.md](https://openaquariumproject.org/specs/automation/spec.md).
- **[OAP-MB]** Open Aquarium Project, "Open Aquarium Module Bus", Revision 0.3, [specs/module-bus/spec.md](https://openaquariumproject.org/specs/module-bus/spec.md).

4.2 Informative References

- **[P0001]** Open Aquarium Project, "Proposal 0001: Reference Controller Architecture", [proposals/0001-reference-controller-architecture.md](https://openaquariumproject.org/proposals/0001-reference-controller-architecture.md).
- **[OAP-LP]** Open Aquarium Project, "Open Aquarium Lighting Programs", Revision 0.2, [specs/lighting-programs/spec.md](https://openaquariumproject.org/specs/lighting-programs/spec.md).
- **[OAP-FP]** Open Aquarium Project, "Open Aquarium Flow Programs", Revision 0.2, [specs/flow-programs/spec.md](https://openaquariumproject.org/specs/flow-programs/spec.md).
- **[RFC 6335]** Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, August 2011.
- **[RFC 8766]** Cheshire, S., "Discovery Proxy for Multicast DNS-Based Service Discovery", RFC 8766, June 2020.
- **[AdProxy]** Cheshire, S. and T. Lemon, "Advertising Proxy for DNS-SD Service Registration Protocol", Internet-Draft draft-ietf-dnssd-advertising-proxy, work in progress.
- **[RFC 8117]** Huitema, C., Thaler, D., and R. Winter, "Current Hostname Practice Considered Harmful", RFC 8117, March 2017.
- **[RFC 8882]** Huitema, C. and D. Kaiser, "DNS-Based Service Discovery (DNS-SD) Privacy and Security Requirements", RFC 8882, September 2020.
- **[RFC 8449]** Thomson, M., "Record Size Limit Extension for TLS", RFC 8449, August 2018.
- **[RFC 9147]** Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", RFC 9147, April 2022.
- **[RFC 7252]** Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.
- **[RFC 8949]** Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, December 2020.
- **[MQTT]** OASIS, "MQTT Version 5.0", OASIS Standard, March 2019.
- **[HA Sensor]** Home Assistant, "Sensor entity", Home Assistant Developer Docs, <https://developers.home-assistant.io/docs/core/entity/sensor/>.
- **[Thread]** Thread Group, "Thread Specification", Version 1.4.1, <https://www.threadgroup.org/ThreadSpec>, available to non-members for internal use only.

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, controller, and binding, in the Fail-Safe Behavior specification [OAP-FS], such as contact, fail-safe behavior, inactive state, stop, safety function, and radio link, and in the Security and Access specification [OAP-SA], such as party, app, service, owner, role, operation, key ID, pairing, pairing window, Prover, Verifier, and relay, have the same meaning here.

Unit: A device or a controller: a party that accepts sessions and that an owner claims.

Session: A connection between two parties over which they exchange messages (Section 9).

Client: The party that opens the connection of a session.

Server: The party that accepts the connection of a session.

Message: One JSON-RPC 2.0 request, response, or notification.

Method: The name of a request or notification, such as `device.command`.

Subscription: A request that a device send state reports and events on a session (Section 11.5).

State report: A `device.state` notification.

Controller of a device: The party that has taken control of the device (Section 12.1).

Adapter: A device that presents equipment of another kind, such as a commercial pump reached through its maker's own interface, as devices of this Specification (Section 16).

Bridged device: A device that an adapter presents.

Cloud-dependent: Describes a bridged device that its adapter can reach only through a service on the internet.

Stop-on-loss equipment: Equipment of a Data Model category whose recommended behavior on loss of contact is `stop` in Fail-Safe Behavior Table 3, such as a heater, a doser, or a CO2 system.

Run-on-loss equipment: Equipment of a Data Model category whose recommended behavior on loss of contact is `run` in Fail-Safe Behavior Table 3, such as a return pump or a circulation pump.

Discriminator: A number from 0 to 4095 that helps an app find the unit whose setup code it has (Section 10.4).

Device network: A Wi-Fi network that a controller hosts for its devices (Section 14.1).

Sleepy device: A device whose radio is off for most of the time, so that it cannot always receive a message within 1 s.

5.3 Notation

Method names, JSON members, and the names of attributes, commands, events, and faults appear in `monospace`. Durations are in seconds, and sizes in bytes, unless stated. In examples, the `jsonrpc` member that every message carries is left out, and whitespace is added. References to other Open Aquarium specifications name them, as in “Fail-Safe Behavior Section 7.5”, or use their tags from Section 4, as in “[OAP-FS Section 7.5]”. A section number with a third part refers to a numbered item of that section: “Section 12.1.2” means item 2 of Section 12.1.

6 Introduction

6.1 Purpose

Equipment that is not a module on the module bus, such as a Wi-Fi wave pump, a light, a doser, or a smart power strip, connects to its controller over the local network [P0001]. Today each maker's equipment speaks its own protocol. The local interfaces of four widely used commercial controller ecosystems are undocumented, unauthenticated, read-only, or missing, and some of their functions are available only through the maker's cloud. This Specification defines one open protocol, so that any conforming device works with any conforming controller and app.

6.2 Design Principles

1. **One model everywhere.** The protocol carries the Device Classes model unchanged, as the module bus does, so the same endpoint works over either.
2. **Standard parts.** Discovery, security, framing, and encoding all come from existing open standards that run on microcontrollers, phones, and servers.
3. **One session, both ways.** A single long-lived connection carries requests, responses, state reports, and supervision in both directions, whichever party opened it. Devices that cannot accept connections, such as sleepy devices, can open them instead.
4. **Supervision from end to end.** Contact runs directly between the controller's key and the device. No broker or other intermediary sits between them.
5. **Secure by default.** Every session is authenticated and encrypted as the Security and Access specification requires.
6. **Local first, cloud too.** Everything works locally, and cloud services take part as ordinary parties alongside local ones.
7. **Honest about dependencies.** Adapters say how each commercial device is reached and who applies its fail-safe behavior, and controllers act on it.
8. **Fits a microcontroller.** Every requirement can be met by a device built on a common single-chip Wi-Fi microcontroller.

6.3 Overview

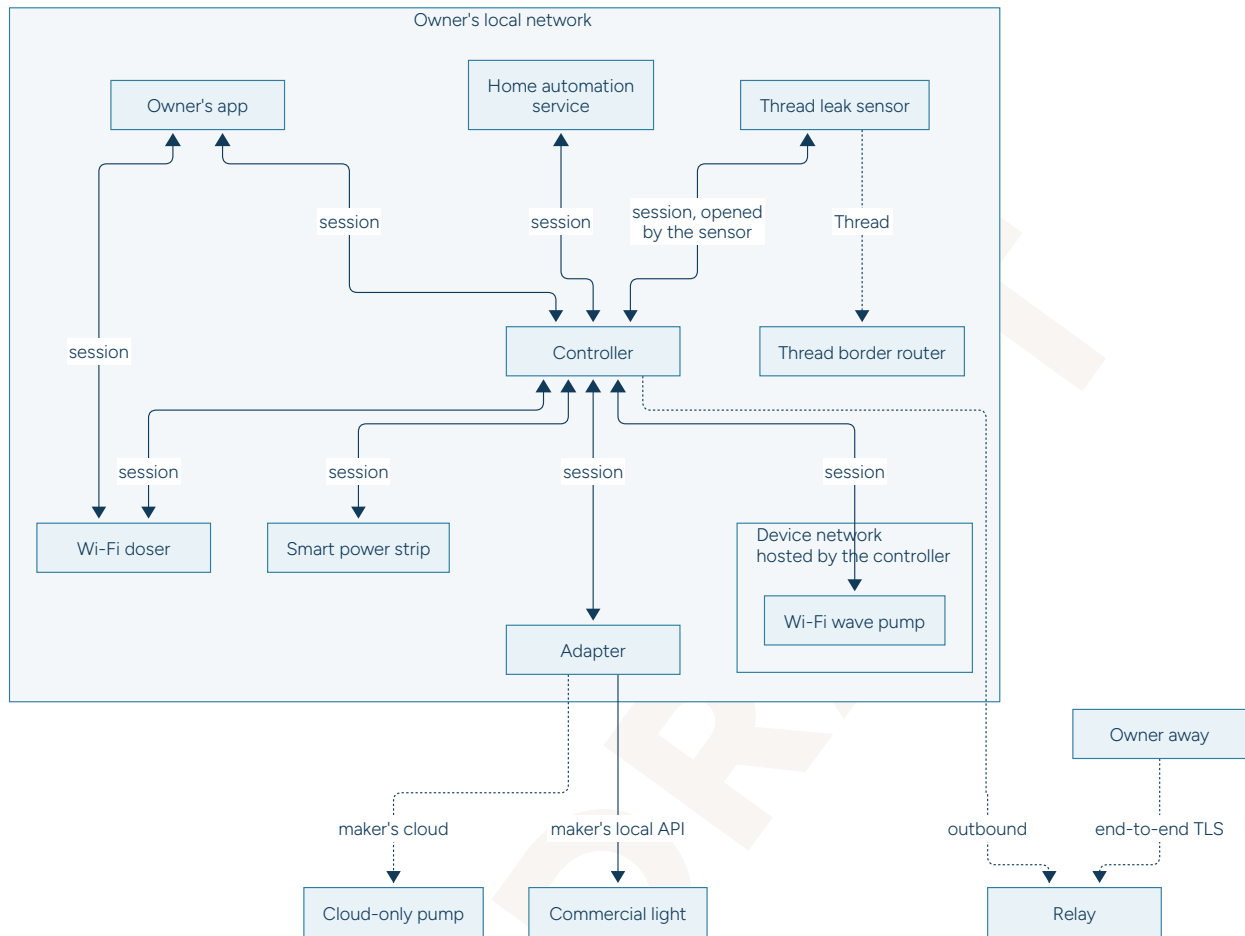


Figure 1 — Parties and sessions

A controller finds its devices with DNS service discovery and opens a session to each. Each session is a WebSocket connection inside TLS 1.3, authenticated at both ends by the keys the owner paired. Over the session, the controller reads the device's description, subscribes to its state, takes control of it, and sends commands and supervision messages. Apps and services open sessions of their own to controllers and devices, with the roles their owners gave them. Owners away from home reach their controller through a relay that forwards encrypted sessions without being able to read them.

Table 1 — Layers of the Device Protocol

Layer	Defined by	Section
Device model: descriptions, attributes, commands, state, events, faults, and fail-safe behavior	[OAP-DC] and [OAP-FS], carried as this Specification defines	11, 12
Methods: session, pairing, control, device, and unit management, and relays	This Specification	9.8
Messages: JSON-RPC 2.0 in JSON	[JSON-RPC], [RFC 8259], [RFC 7493]	9.3
Sessions: WebSocket over TLS 1.3 with mutual authentication	[RFC 6455], [OAP-SA] Section 10	9.1, 9.2
Discovery: multicast DNS and DNS-SD; SRP on Thread	[RFC 6762], [RFC 6763], [RFC 9665]	8
Network: IPv4 or IPv6 over Ethernet, Wi-Fi, or Thread	—	7.1

7 Networks and Identities

7.1 Networks

1. The protocol runs over IP, on Ethernet, Wi-Fi, and Thread. Bluetooth Low Energy is used only for setup (Section 10.8).
2. Devices on Ethernet or Wi-Fi **MUST** support IPv4 with DHCP, and **SHOULD** support IPv6. Devices on Thread use IPv6. Controllers **MUST** support both, so that they can reach devices on Thread through a border router.
3. Parties **MUST NOT** depend on a unit keeping its address. They find units by key ID (Section 8.5).
4. Every unit **MUST** work on a network that has no internet access.

7.2 Parties

Table 2 — Parties

Party	Accepts sessions	Opens sessions	Role in this Specification
Device	Yes	To its controller and configured services, optionally (Section 14.2)	Presents endpoints; follows its controller.
Adapter	Yes	As a device	A device that presents bridged devices (Section 16).
Controller	Yes, from apps, services, and devices	To devices, and to relays	Controls devices; serves apps.
App	No	To controllers, devices, and relays	Shows and operates a system for a person.
Service	No	As an app	Acts for the owner without a person, such as a home automation server.
Relay	Yes	No	Forwards remote sessions (Section 15.4).

7.3 Device Identifiers

1. A unit is identified by its key ID [OAP-SA Section 8.3].
2. Where another specification names a device by binding, such as an Automation binding [OAP-AU Section 8.5] or an audit record, a unit on this binding is `device-protocol:` followed by its key ID, for example `device-protocol:Rj4mT9wK2bX7nP1qV6sZ0cH5yD8fL3gA4eU9i02kN7s`.
3. A bridged device is `device-protocol:` followed by its adapter's key ID, a slash, and the bridged device's identifier, for example `device-protocol:Zb6RJYUIdLcoeM_Tkd7o0Htw5xBKA4dBdSErpsGxsH0/wave-1`. The identifier of a bridged device **MUST** consist of 1 to 64 lowercase letters, digits, hyphens, and underscores, **MUST** begin with a letter or digit, and **MUST** stay the same for as long as the adapter keeps its configuration.
4. A unit's key ID changes when it is factory reset and replaces its identity key [OAP-SA Section 8.5], and its bindings then no longer match. A controller that adds a device with the same manufacturer, model, and serial number as a device it has lost **SHOULD** offer to move the lost device's bindings to it, after the owner confirms.

8 Discovery

8.1 Service

1. Every unit on Ethernet or Wi-Fi **MUST** advertise one service instance of type `_oap._tcp` in the `local.` domain with multicast DNS [RFC 6762] and DNS service discovery [RFC 6763], on every interface on which it accepts sessions, while it accepts sessions. The service type is provisional until it is registered with IANA (Appendix C, issue 4).
2. The SRV record gives the port on which the unit accepts sessions (Section 9.1).
3. The instance name is for people. It **SHOULD** be the unit's model name, a space, and the first four characters of its key ID, such as `PS-4 6dNM`. It **MUST NOT** contain names the owner has set, the owner's

name, or the unit's serial number [OAP-SA Section 10.8.4]. Units resolve name conflicts as [RFC 6762] Section 9 describes, so parties MUST NOT use instance names to identify units.

4. The unit's host name MUST NOT contain names the owner has set, the owner's name, or its serial number [RFC 8117]. For example, it can be `oap-` followed by eight random hexadecimal digits.
5. A unit MUST update its TXT record within 1 s when a pairing window opens or closes, and MUST send goodbye records [RFC 6762 Section 10.1] when it stops accepting sessions.
6. Controllers, apps, and services browse for `_oap._tcp` while they need to find units, with the query intervals of [RFC 6762] Section 5.2.

8.2 TXT Record

The TXT record [RFC 6763 Section 6] carries the keys of Table 3.

Table 3 — TXT record keys

Key	Presence	Value	Description
<code>v</code>	REQUIRED	0.2, or 0.1 for a unit of revision 0.1	The revision of this Specification that the unit implements.
<code>id</code>	REQUIRED	43 characters	The unit's key ID, in the text form of Security and Access Section 8.3.
<code>kind</code>	REQUIRED	<code>controller</code> or <code>device</code>	Whether the unit is a controller or a device. An adapter is a device.
<code>pw</code>	OPTIONAL	1	Present while the unit has a pairing window open (Security and Access Section 9.3), and absent otherwise.
<code>d</code>	OPTIONAL	0 to 4095, in decimal	The unit's discriminator (Section 10.4). Present only while a pairing window is open, and only on units that have a discriminator.

1. Units MUST NOT add keys that this Specification does not define. Parties MUST ignore keys they do not know, which later revisions may add.
2. The whole TXT record MUST NOT be longer than 200 bytes, which [RFC 6763] Section 6.2 gives as the typical size.

8.3 Subtypes

1. A controller MUST also advertise its instance under the subtype `_controller._sub._oap._tcp`.
2. A unit with a pairing window open MUST also advertise its instance under the subtype `_pairing._sub._oap._tcp`, and MUST remove it when the window closes.

An app that is setting up new equipment browses for the `_pairing` subtype, and a device that connects to its controller (Section 14.2) can browse for the `_controller` subtype.

8.4 Thread

1. A device on Thread MUST register the same records, including subtypes, with the Service Registration Protocol [RFC 9665], using the registrar that its Thread network provides.
2. Thread border routers republish registered services on their other networks with multicast DNS [AdProxy], so controllers and apps find devices on Thread as they find any other. A controller that is itself a Thread border router MUST find the services registered with it as it finds services advertised with multicast DNS.
3. A device on Thread MUST update its registration within 1 s when a pairing window opens or closes.

8.5 Finding a Party

1. To find a unit whose key ID it knows, a party browses for `_oap._tcp` and connects to the instance whose `id` matches. A TXT record is only a hint: anyone can advertise any key ID, so the party relies on the key the unit proves in its TLS handshake (Section 10.1).
2. A party MAY remember the last address and port of a unit, and try them first.
3. Multicast DNS does not cross routers, so units on another network, such as another VLAN, cannot be found this way. Controllers, apps, and services MUST let the owner enter the address and port of a unit by hand.
4. Apps and services do not advertise themselves.

9 Sessions

9.1 Connections

1. A session runs over a TCP connection protected with TLS 1.3, with both parties authenticated by their identity keys, as Security and Access Section 10 requires.
2. The client opens the connection; the server accepts it. Once the session is open, which party opened it matters only for the WebSocket handshake (Section 9.2): either party may send requests (Section 9.5).
3. A unit accepts sessions on the port given in its SRV record. The port MUST NOT also serve a web interface, because a web interface must not request a client certificate [OAP-SA Section 10.2.2].
4. Parties SHOULD send the record size limit extension [RFC 8449], so that devices with little memory need smaller buffers.

9.2 WebSocket

1. When the TLS handshake is complete, the client sends an HTTP/1.1 GET request [RFC 9112] for the path `/oap` that asks to upgrade the connection to WebSocket [RFC 6455], and offers the subprotocol `oap.0.1` in `Sec-WebSocket-Protocol`. The subprotocol names the format of messages, which this revision keeps from revision 0.1, so that units, controllers, and apps of both revisions work together. A later revision that changes the format will name a new subprotocol.
2. The server MUST refuse an upgrade that does not offer `oap.0.1`, with status 400.
3. The server MUST refuse an upgrade request that has an `Origin` header, with status 403. Web browsers send one, and this Specification defines no access from web pages.
4. Parties MUST NOT negotiate WebSocket extensions. In particular, compression is not used, because compressing secrets together with data an attacker can influence can reveal the secrets.
5. Each message is sent as one WebSocket text message. Parties SHOULD send each message in a single frame, and MUST accept messages split into several frames [RFC 6455 Section 5.4]. Binary messages are used only by relays (Section 15.4); a party that receives one on a session closes the session with code 1003.
6. Parties close sessions with the codes of Table 4.

Table 4 — WebSocket close codes

Code	Meaning
1000	Normal closure.
1002	A message broke the rules of this Specification.
1003	A binary message arrived on a session.
1008	A rule of Security and Access was broken, for example an untrusted key sent repeated requests.
1009	A message was larger than the receiver accepts (Section 9.3.6).
4001	The other party's credential was revoked, or its grant expired [OAP-SA Section 10.6].
4002	A newer session from the same key replaced this one (Section 9.4.6).
4003	The unit was factory reset, or its data erased.
4004	Nothing was received for 60 s (Section 9.10).
4005	The unit is restarting.

9.3 Messages

1. Each WebSocket message carries exactly one JSON-RPC 2.0 request, response, or notification [JSON-RPC], encoded as I-JSON [RFC 7493].
2. Parties MUST NOT send batches. A party that receives one responds with a single `invalid-request` error.
3. A request's `id` is an integer from 0 to 2147483647, or a string of 1 to 64 characters. Each party chooses the identifiers of its own requests, and MUST NOT reuse one while a request with it is outstanding on the session.
4. `params` MUST be an object, and MAY be left out when a method has no parameters. A method with nothing to return returns the empty object `{}`.

5. Receivers MUST ignore members they do not know, so that later revisions can add members. Senders MUST NOT add members this Specification does not define, except in `extensions` objects (Section 9.9).
6. Each party states in `session.hello` the largest message it accepts, in bytes of UTF-8. Devices MUST accept messages of at least 8192 bytes, and controllers, apps, and services messages of at least 131072 bytes. A party MUST NOT send a message larger than the other party accepts. It sends the error `too-large` instead of a response that would be larger, and a receiver of a larger message closes the session with code 1009.
7. A receiver MAY refuse, with `invalid-request`, a message whose structures are nested more than 16 deep.
8. Every message MUST be valid against `schema/messages.schema.json` [JSON Schema].

For example, a command and its response:

```
{ "id": 7, "method": "device.command",
  "params": { "endpoint": "head_1", "command": "dose", "parameters": { "volume": 2.5 } } }
{ "id": 7, "result": {} }
```

9.4 Opening a Session

1. Each party MUST send `session.hello` as its first message, and MUST NOT send any other request until it has received the response to its own `session.hello`. A party responds to any other request that arrives before the sender's `session.hello` with `bad-state`.
2. `session.hello` has the parameters of Table 5, and its result has the members of Table 6.
3. The receiver fills in `peer` from its own records: a controller from its user list and the devices it has paired, and a device from its pairing list [OAP-SA Section 11].
4. To a party whose key it does not trust, a unit MUST NOT reveal more than `kind`, `maxMessageSize`, `claimed`, `pairingWindow`, and `peer.trusted` [OAP-SA Section 10.8.4].
5. Pairing (Section 10.2) and sign-in (Section 10.7) change how the receiver sees the sender during the session. Their results give the new role.
6. When a new session from a key has exchanged `session.hello`, a unit SHOULD close its older sessions from the same key with code 4002, so that sessions left half-open by a lost connection do not use up its sessions.

Table 5 — Parameters of `session.hello`

Member	Type	Presence	Description
<code>kind</code>	controller, device, app, or service	REQUIRED	What the sender is. An adapter is a <code>device</code> .
<code>maxMessageSize</code>	integer	REQUIRED	The largest message the sender accepts, in bytes (Section 9.3.6).
<code>name</code>	string	OPTIONAL	A label for people, of up to 64 characters, such as “Alex’s phone”. Used as the default label when the sender is paired. It MUST NOT be used to decide anything.
<code>identity</code>	string	Over Bluetooth only	The sender’s public identity key (Section 10.8.4).

Table 6 — Result of `session.hello`

Member	Type	Presence	Description
<code>kind</code>	controller or device, or as Table 5	REQUIRED	What the receiver is.
<code>maxMessageSize</code>	integer	REQUIRED	The largest message the receiver accepts.
<code>claimed</code>	boolean	REQUIRED for units	Whether the unit has been claimed.
<code>pairingWindow</code>	boolean	REQUIRED for units	Whether a pairing window is open.
<code>bridge</code>	boolean	OPTIONAL	<code>true</code> for an adapter (Section 16).
<code>peer</code>	object	REQUIRED	How the receiver sees the sender: <code>trusted</code> (boolean), and, when it is trusted, <code>keyId</code> , <code>role</code> (the sender’s role, or <code>null</code> for a device paired with a controller), <code>controller</code> (whether the sender is marked as a controller in a

Member	Type	Presence	Description
			device's pairing list), <code>inControl</code> (whether the sender is the device's controller), and <code>user</code> (on a controller, the user's <code>id</code> and <code>name</code>).
<code>relays</code>	list of objects	OPTIONAL	On a controller, for users whose grant allows remote access: the relays it listens on, each with a <code>url</code> and, where the relay is checked by key, its <code>keyId</code> (Section 15.4).

9.5 Requests and Responses

1. The receiver of a request MUST send exactly one response, with a `result` or an `error`.
2. A party MUST respond to a request within 5 s of receiving it (provisional). A party that has not received a response within 10 s SHOULD treat the request as failed. It then does not know whether the request was carried out (Section 12.4.3).
3. Responses MAY be sent in any order, but a device MUST carry out commands and attribute writes to one endpoint in the order it received them.
4. A device MUST carry out a stop, and `device.stop`, from its controller before other requests that it has received but not yet started.
5. A party MAY limit how many requests it handles at once. It MUST NOT ignore a request: when it cannot accept another, it responds with `busy`. A device MUST accept a stop, and `device.stop`, from its controller even when it has reached its limit, and MUST NOT refuse either with `busy` [OAP-FS Section 8.2].
6. Each method has an access rule (Table 9). Before it carries out a request, the receiver MUST check the rule against the key that authenticated the session, as Security and Access Section 11.3 requires, and refuse with:
 - `not-trusted` when the method needs a trusted key and the receiver does not trust the sender's;
 - `not-permitted` when the sender's role does not permit the operation; and
 - `not-controller` when the operation is accepted only from the device's controller (Section 12.1) and the sender is not it.
7. A party that receives a request for a method it does not serve responds with `method-not-found`.

9.6 Errors

1. An error has a `code`, a `message` of up to 128 characters for people, and optional `data`.
2. Table 7 lists the codes. Codes from -32768 to -32000 are those of [JSON-RPC]; codes 1 to 9 have the meanings of the status codes of `oap.Command.0.2` on the module bus [OAP-MB Section 11.3], so that adapters and controllers can pass them on unchanged.
3. Parties MUST NOT use other codes, except codes from 1000 to 1999 for extension methods (Section 9.9), whose meaning the extension defines.

Table 7 — Error codes

Code	Name	Meaning
-32700	<code>parse-error</code>	The message is not valid JSON.
-32600	<code>invalid-request</code>	The message is not a valid request, or is a batch.
-32601	<code>method-not-found</code>	The receiver does not serve this method.
-32602	<code>invalid-params</code>	The parameters do not have the structure the method defines.
-32603	<code>internal-error</code>	The receiver failed while handling the message.
1	<code>bad-endpoint</code>	The device has no such endpoint, or the adapter has no such device.
2	<code>bad-command</code>	The endpoint's class and features do not define the command, or the device does not implement it.
3	<code>bad-parameter</code>	A command parameter or attribute value is missing, unknown, of the wrong type, or out of range.
4	<code>limit</code>	Refused by a limit the device enforces (Device Classes Section 7.8).
5	<code>local-control</code>	Refused because a local control is in effect (Fail-Safe Behavior Section 12).
6	<code>not-controller</code>	The sender is not the device's controller, or may not take control of it now.
7	<code>busy</code>	The receiver cannot accept the request now, for example during <code>powerOnDelay</code> , a firmware update, or another pairing exchange.

Code	Name	Meaning
8	<code>device-error</code>	The device could not carry out a valid request because of a fault of its own.
9	<code>bad-state</code>	Valid, but not possible in the current state, such as a firmware update while dosing, or a request before <code>session.hello</code> .
10	<code>not-permitted</code>	The sender's role does not permit the operation (Security and Access Section 11.3).
11	<code>not-trusted</code>	The receiver does not trust the sender's key, and only pairing and sign-in are possible.
12	<code>bad-attribute</code>	The attribute does not exist, or is not a config attribute.
13	<code>pairing-failed</code>	The pairing exchange failed. The Verifier counts a failed attempt.
14	<code>no-pairing-window</code>	No pairing window is open for that code.
15	<code>unreachable</code>	The adapter cannot reach the device, or the relay cannot reach the controller.
16	<code>too-large</code>	The response would be larger than the receiver of the response accepts.
17	<code>sign-in-failed</code>	The user or password is wrong, or too many attempts have failed.

9.7 Value Encoding

Attribute values, command parameters, and event fields are encoded as Table 8 shows.

Table 8 — Value encoding

Device Classes type	JSON
boolean	<code>true</code> or <code>false</code> .
integer	A number without a fraction or exponent, from $-2^{53} + 1$ to $2^{53} - 1$.
number	A number, in the attribute's unit [OAP-DC Section 7.3]. Not-a-number and infinities cannot be sent; an unusable reading is shown by its <code>quality</code> .
string, enum, parameter, unit	A string.
timestamp	An RFC 3339 date and time with a UTC offset [RFC 3339], such as <code>2026-10-14T01:02:03.250Z</code> , with up to three digits of fractional seconds; or <code>null</code> when the unit's clock is not valid (Section 12.3).
date	A string <code>YYYY-MM-DD</code> .
list	An array.
object	An object.

Binary data, such as keys and parts of files, is encoded in unpadding base64url [RFC 4648 Section 5].

9.8 Methods

Table 9 lists the methods. Requests get a response; notifications do not.

Where Access names an operation, the receiver checks it as Security and Access Table 6 defines: on a device, against the sender's entry in its pairing list; on a controller, against the sender's user. An operation whose device column in that table is `controller` is permitted only from the device's controller (Section 12.1). The other access rules are:

- **any**: any party, including one whose key the receiver does not trust;
- **trusted**: any party whose key the receiver trusts;
- **offered**: only the party to which the receiver offered the file, and only for that file (Section 13.2);
- **relay**: as the relay's operator decides (Section 15.4);
- **invitation**: on a device, `open-pairing-window`; on a controller, `manage-users` to invite a new user, and `manage-own-credentials` for the sender's own new app (Section 10.5.4); and
- **controller mark**: a party marked as a controller in the device's pairing list, with the `owner` or `full-control` role (Section 12.1.2).

Which methods each kind of party serves is set out in Section 17.

Table 9 — Methods

Group	Method	Type	From	To	Access	Description
Session	<code>session.hello</code>	request	any party	any party	any	Opens the session. Each party sends it first, and the response tells the sender how the receiver sees it.
Session	<code>session.signIn</code>	request	app or service	controller	any	Signs in with a user's password (Security and Access Section 10.5).
Pairing	<code>pairing.start</code>	request	Prover	device or controller	any	Starts a pairing exchange with a setup code, an invitation, or an emergency code, and returns the salt and iteration count of its verifier.
Pairing	<code>pairing.share</code>	request	Prover	device or controller	any	Sends the Prover's SPAKE2+ share, and returns the Verifier's share and confirmation.
Pairing	<code>pairing.confirm</code>	request	Prover	device or controller	any	Sends the Prover's SPAKE2+ confirmation, and returns what the pairing granted.
Pairing	<code>pairing.invite</code>	request	app or controller	device or controller	invitation	Creates an invitation: a one-time code, and the role that pairing with it grants.
Pairing	<code>pairing.list</code>	request	app or controller	device	<code>view-access</code>	Returns the device's pairing list.
Pairing	<code>pairing.add</code>	request	app or controller	device	<code>manage-pairings</code>	Adds a key to the device's pairing list, with a role.
Pairing	<code>pairing.update</code>	request	app or controller	device	<code>manage-pairings</code>	Changes the role, label, or controller mark of an entry in the pairing list.
Pairing	<code>pairing.remove</code>	request	app or controller	device	<code>manage-pairings</code>	Removes an entry from the pairing list.
Control	<code>control.take</code>	request	controller	device	controller mark	Makes the sender the device's controller, as Fail-Safe Behavior Section 7.5 allows.
Control	<code>control.release</code>	request	controller	device	<code>command-endpoint</code>	Ends the sender's control of the device.
Control	<code>control.supervise</code>	notification	controller	device	<code>command-endpoint</code>	Shows that the controller is still in control, and gives its time.
Device	<code>device.describe</code>	request	any party	device	<code>view-configuration</code>	Returns the device description (Device Classes Section 12).
Device	<code>device.list</code>	request	any party	adapter	<code>view-configuration</code>	Returns the devices behind an adapter.
Device	<code>device.read</code>	request	any party	device	<code>view-state</code>	Returns the current values of attributes of the device or of one endpoint.
Device	<code>device.write</code>	request	app or controller	device	<code>change-device-settings</code>	Sets config attributes of the device or of one endpoint, all or none. Fail-safe settings need <code>change-fail-safe-settings</code> , and <code>updateChannel</code> and <code>automaticInstall</code> need <code>install-firmware</code> .

Group	Method	Type	From	To	Access	Description
Device	<code>device.command</code>	request	any party	device	<code>command-endpoint</code>	Sends a command to an endpoint. <code>identify</code> needs only <code>identify</code> .
Device	<code>device.stop</code>	request	controller	device	<code>stop-all</code>	Puts every output and unit endpoint of the device in its inactive state, for a while with <code>revertAfter</code> .
Device	<code>device.identify</code>	request	any party	device	<code>identify</code>	Makes the whole device show where it is, without changing any output.
Device	<code>device.subscribe</code>	request	any party	device	<code>view-state</code>	Starts state reports and events for the session, replacing any earlier subscription.
Device	<code>device.unsubscribe</code>	request	any party	device	<code>view-state</code>	Ends the session's subscription.
Device	<code>device.state</code>	notification	device	subscriber	—	Reports the state of the device or of one endpoint.
Device	<code>device.event</code>	notification	device	subscriber	—	Reports an event of an endpoint.
Device	<code>device.config</code>	notification	device	subscriber	—	Reports config attributes that changed.
Device	<code>device.listChanged</code>	notification	adapter	subscriber	—	Reports that the devices behind an adapter, or whether they can be reached, changed.
Unit	<code>firmware.update</code>	request	app or controller	device	<code>install-firmware</code>	Asks the device to install a firmware image, which it then reads from the sender with <code>file.read</code> .
Unit	<code>firmware.progress</code>	notification	device	app or controller	—	Reports the progress of a firmware update to the party that asked for it.
Unit	<code>firmware.revert</code>	request	app or controller	device	<code>install-firmware</code>	Returns the device to its previous firmware (Security and Access Section 15.4.4), and restarts it.
Unit	<code>firmware.configure</code>	request	app or controller	device	<code>install-firmware</code>	On a device of Device Classes revision 0.1 only: sets its update channel and whether automatic installation is on. Other devices set them with <code>device.write</code> (Section 13.1.8).
Unit	<code>file.read</code>	request	device	app or controller	<code>offered</code>	Reads part of a file that the receiver offered, such as a firmware image.
Unit	<code>network.read</code>	request	app or controller	device or controller	<code>view-configuration</code>	Returns the unit's network settings, without passwords.
Unit	<code>network.configure</code>	request	app or controller	device or controller	<code>change-network-settings</code>	Changes the unit's network settings, and returns to the previous settings if the new ones fail.
Unit	<code>unit.restart</code>	request	app or controller	device	<code>restart</code>	Restarts the unit.

Group	Method	Type	From	To	Access	Description
Unit	<code>unit.factoryReset</code>	request	app or controller	device	<code>factory-reset</code>	Factory resets the unit (Security and Access Section 9.8).
Unit	<code>unit.eraseData</code>	request	app or controller	device	<code>erase-data</code>	Erases history and other personal data without a factory reset.
Unit	<code>audit.read</code>	request	app or controller	device	<code>view-audit-log</code>	Returns records of the unit's audit log.
Unit	<code>consent.list</code>	request	app or controller	device	<code>view-configuration</code>	Returns each purpose for which the unit would send data off the owner's network, and whether the owner has consented.
Unit	<code>consent.set</code>	request	app or controller	device	<code>manage-consent</code>	Gives or withdraws consent for one purpose.
Relay	<code>relay.register</code>	request	controller	relay	<code>relay</code>	Asks the relay to forward connections for the controller's key ID.
Relay	<code>relay.incoming</code>	notification	relay	controller	—	Tells the controller that an app wants to connect, and which key it presented.
Relay	<code>relay.accept</code>	request	controller	relay	<code>relay</code>	Sent on a new relay session, which then carries the accepted connection.
Relay	<code>relay.refuse</code>	request	controller	relay	<code>relay</code>	Refuses an incoming connection.
Relay	<code>relay.connect</code>	request	app or service	relay	<code>relay</code>	Asks the relay to connect the session to a controller, by its key ID.

9.9 Extensions

1. An extension method is named with an extension name [OAP-DM Section 7.6], a dot, and a name in lower camel case, such as `com.example.pump.calibrateFlow`. The names of this Specification's methods have exactly one dot, and extension methods have more.
2. Extensions to the members of a parameter object or a result go in an `extensions` object within it, keyed by extension name.
3. An extension method that changes what an output does MUST be accepted only from the device's controller.

9.10 Keeping Sessions Alive

1. A party MUST answer a WebSocket Ping with a Pong [RFC 6455 Section 5.5].
2. A party that has sent nothing on a session for 20 s SHOULD send a Ping.
3. A party MAY close a session on which it has received nothing, not even a Pong, for 60 s, with code 4004.
4. When a session ends, its subscription and any pairing exchange in progress end with it.

10 Authentication and Pairing

10.1 Trust

1. Every session authenticates both parties by their identity keys, with TLS, as Security and Access Section 10 requires. Each party computes the other's key ID from the certificate it presented [OAP-SA Section 8.3].
2. A party whose key the receiver does not trust may send only `session.hello`, the pairing requests of Section 10.2, and, to a controller, `session.signIn`. The receiver refuses every other request from it with `not-trusted` [OAP-SA Section 10.3.2].

3. The key that authenticated a session stays the same for the whole session. When the receiver stops trusting it, because its entry was removed, its credential was revoked, or its grant expired, the receiver MUST close every session authenticated with it within 5 s, with code 4001 [OAP-SA Section 10.6].

10.2 The Pairing Exchange

The pairing exchange carries the SPAKE2+ pairing of Security and Access Section 9.4. The party that sends `pairing.start` is the Prover, and the unit that receives it is the Verifier.

Table 10 — Pairing requests

Request	Parameters	Result
<code>pairing.start</code>	<code>code</code> : <code>setup</code> , <code>invitation</code> , or <code>emergency</code> (Section 10.3); <code>slot</code> , from 0 to 99, for an invitation or an emergency code; <code>userName</code> (optional): the name of the user that claiming or recovering a controller creates.	<code>salt</code> : 16 to 32 bytes; <code>iterations</code> : from 1000 to 100 000.
<code>pairing.share</code>	<code>shareP</code> : the Prover's share.	<code>shareV</code> : the Verifier's share; <code>confirmV</code> : the Verifier's confirmation.
<code>pairing.confirm</code>	<code>confirmP</code> : the Prover's confirmation.	<code>role</code> : the role the pairing granted; <code>controller</code> : on a device, whether the new entry is marked as a controller; <code>user</code> : on a controller, the user's <code>id</code> and <code>name</code> ; <code>emergency</code> : <code>true</code> for an emergency session.

1. The Verifier MUST refuse `pairing.start` with `no-pairing-window` when no pairing window is open for the code it names, and with `busy` while another exchange is in progress [OAP-SA Section 9.3.4].
2. Shares are P-256 points in uncompressed form, 65 bytes, and confirmations are 32 bytes, all in base64url.
3. The SPAKE2+ inputs are those of Security and Access Section 9.4.5. The channel binding is the `tls-exporter` value of the TLS connection that carries the session [RFC 9266], and the identities are the key IDs, as 32 bytes, of the certificates the two parties presented. Through a relay, the TLS connection is the one between app and controller inside the relayed stream (Section 15.4).
4. A Prover that finds `confirmV` wrong MUST NOT send `pairing.confirm`. The Verifier counts a failed attempt [OAP-SA Section 9.3.3] for every exchange in which it received `pairing.share` but did not receive a correct `confirmP` within 30 s, and refuses a wrong `confirmP` with `pairing-failed`.
5. An exchange belongs to the session on which it started, and fails if the session ends.
6. When the exchange succeeds, the Verifier records the Prover's key as Security and Access Section 9.4.7 requires, and from then on trusts the session with the role it granted. The Prover records the Verifier's key. When the exchange claims a device with its setup code, the device MUST mark the new entry as a controller if the Prover's `kind` in `session.hello` was `controller`, and MUST NOT mark it otherwise [OAP-SA Section 9.5.4]. An entry made with an invitation is marked as the invitation's `controller` says (Section 10.5).
7. Claiming a controller creates a user with the `owner` role [OAP-SA Section 9.5.2], named `userName`, or the Prover's `name` from `session.hello` if `userName` is absent. Owner recovery [OAP-SA Section 11.7.3] does the same with a new user.
8. An exchange with an emergency code makes the session an emergency session [OAP-SA Section 14]: the session has the `override` role, limited as Security and Access Section 14.5 requires, and the controller does not keep the Prover's key.

10.3 Codes

Table 11 — Codes

Code	Form	Made by	Used for
Setup code	At least 8 digits	The manufacturer, or a unit with a display for each pairing window [OAP-SA Section 9.2]	Claiming; owner recovery on a controller; network settings after a button press [OAP-SA Section 9.6.4]
Invitation	A 2-digit slot, then at least 8 digits	The unit, or the owner's app (Section 10.5)	New users, a user's new app, and other parties on a device [OAP-SA Section 11.5]

Code	Form	Made by	Used for
Emergency code	A 2-digit slot, then at least 8 digits	The controller	Emergency access [OAP-SA Section 14.4]

1. The slot is not secret. It tells the Verifier which of its verifiers to use, because a unit can have several invitations and emergency codes at once. The password for the derivation of Security and Access Section 9.4.3 is the digits after the slot.
2. A unit **MUST NOT** have two active invitations, or two emergency codes, with the same slot.
3. Codes are shown and typed as digits in groups, such as 3018 4762 for a setup code and 27-4815-1623 for an invitation. Parties ignore spaces and hyphens in codes.

10.4 Setup Code Labels

1. The QR code on a unit's label and packaging [OAP-SA Section 9.2.2] contains the text 0AP:1:, the unit's discriminator as four decimal digits, a colon, and the setup code's digits, for example 0AP:1:2817:30184762. It uses only digits, capital letters, and colons, so that it fits the QR code's alphanumeric mode. Apps **MUST** ignore anything after a further colon, which a later revision may use.
2. The discriminator is a number from 0 to 4095, chosen at random for each unit when it is made. It is not secret and does not identify the unit, because many units share each value. A unit with a pairing window open advertises it (Section 8.2), so that an app with several units in pairing windows tries the right one first.
3. A unit that shows a new setup code for each pairing window **SHOULD** also choose a new discriminator for each window, and show both.
4. The label **SHOULD** also show the setup code as digits, in groups of four.
5. Every attempt with the wrong unit uses up one of that unit's attempts, so an app **SHOULD NOT** try a code on a unit whose discriminator differs from the code's.

10.5 Invitations

`pairing.invite` creates an invitation [OAP-SA Section 11.5].

Table 12 — Parameters of `pairing.invite`

Member	Type	Presence	Description
<code>role</code>	role	REQUIRED, unless <code>self</code> is <code>true</code>	The role that pairing with the invitation grants.
<code>label</code>	string	OPTIONAL	On a device, the label of the new entry in its pairing list.
<code>controller</code>	boolean	OPTIONAL	On a device, whether the new entry may act as its controller. Only with the <code>owner</code> or <code>full-control</code> role.
<code>user</code>	object	REQUIRED on a controller, unless <code>self</code> is <code>true</code>	The new user's <code>name</code> , <code>kind</code> (<code>person</code> or <code>service</code>), and optionally <code>grantExpires</code> (a timestamp) and <code>remote</code> (a boolean) [OAP-SA Section 11.1].
<code>self</code>	boolean	OPTIONAL	On a controller, <code>true</code> for a new credential of the sender's own user [OAP-SA Section 11.5.2].
<code>validFor</code>	number (s)	OPTIONAL	How long the invitation stays usable. Default 86 400 (24 hours), at most 604 800 (7 days) [OAP-SA Section 11.5.3].
<code>verifier</code>	object	OPTIONAL	A verifier the sender made from a code it chose: <code>w0</code> (32 bytes), <code>L</code> (65 bytes), <code>salt</code> , and <code>iterations</code> , as Section 10.2 encodes them.

The result has the invitation's `slot`, its `expires` time, and, when the unit made the code itself, the `code`: the digits that follow the slot.

1. A device **MUST** accept a `verifier` [OAP-SA Section 11.5.3], and **MAY** make the code itself when there is none. A controller **MUST** make the code itself when there is no `verifier`.
2. A unit **MUST NOT** return a code more than once, and **MUST** store only its verifier [OAP-SA Section 9.2.4].
3. The invitation opens a pairing window for its slot, which closes when it is used once or when it expires.
4. Access: on a device, `open-pairing-window`; on a controller, `manage-users` to invite a new user, and `manage-own-credentials` with `self` (the access rule `invitation` of Section 9.8).

10.6 Pairing Lists

A device's pairing list [OAP-SA Section 11.6] is managed with the requests of Table 13. Each entry has a `keyId`, a `role`, a `label`, `controller` (whether the party may act as the device's controller), and `added` (a timestamp, or `null`).

Table 13 — Pairing list requests

Request	Parameters	Result
<code>pairing.list</code>	—	<code>entries</code> : the entries, each with <code>inControl</code> , which is <code>true</code> for the device's controller.
<code>pairing.add</code>	<code>keyId</code> , <code>role</code> , <code>label</code> (optional), <code>controller</code> (optional, default <code>false</code>).	<code>{}</code>
<code>pairing.update</code>	<code>keyId</code> , and any of <code>role</code> , <code>label</code> , and <code>controller</code> .	<code>{}</code>
<code>pairing.remove</code>	<code>keyId</code> .	<code>{}</code>

1. A controller uses `pairing.add` to add the owners' keys when it claims a device, and an owner's app uses it to add the controller [OAP-SA Sections 9.5.4 and 9.5.5].
2. A device MUST refuse, with `bad-state`, a change that would leave it without an entry with the `owner` role [OAP-SA Section 11.6.5].
3. A device MUST refuse, with `bad-parameter`, a `controller` mark on an entry whose role is not `owner` or `full-control`.
4. When the entry of the device's controller is removed, or loses its mark, the device has lost contact with its controller [OAP-SA Section 10.6.2], and has no controller from then on (Section 12.1). Each of its output and unit endpoints takes its fail-safe behavior when its `failSafeTimeout` expires, or at once when another controller takes control first (Section 12.1.6).

10.7 Password Sign-In

1. A party whose key a controller does not trust MAY sign in with a user's password [OAP-SA Section 10.5] by sending `session.signIn` with `user`, the user's name or `id`, and `password`. The result gives the user's `role` and `user` (`id` and `name`).
2. The session then has the user's role until it ends. The key that authenticated the session is not added to the user's credentials.
3. Controllers MUST limit failed sign-ins as Security and Access Section 10.5.4 requires, and refuse with `sign-in-failed`.
4. Devices have no passwords, and respond with `method-not-found`.

10.8 Setup over Bluetooth Low Energy

A unit MAY let an app claim it, and give it network settings, over Bluetooth Low Energy [BT Core] [OAP-SA Section 9.6]. If it does, it MUST do so as this section describes. The UUIDs in this section are provisional (Appendix C, issue 4).

1. **Advertising.** While a pairing window is open for its setup code, or a button press has opened a window for new network settings [OAP-SA Section 9.6.4], the unit advertises the service UUID `c98e498b-93f2-4615-9979-1ad4ec7d1b40`, with service data of 4 bytes: a version (1), its discriminator in 2 bytes, least significant byte first, and flags, in which bit 0 means that the unit is not claimed and bit 1 means that it wants network settings.
2. **Service.** The unit's GATT service with that UUID has two characteristics: requests, `c98e498c-93f2-4615-9979-1ad4ec7d1b40`, which the app writes; and responses, `c98e498d-93f2-4615-9979-1ad4ec7d1b40`, on which the unit sends notifications.
3. **Fragments.** Each message is split into fragments that fit the connection's ATT MTU. Each fragment is a 1-byte header followed by part of the message. In the header, bit 0 is 1 when more fragments of the message follow, bit 1 is 1 when the message is encrypted, and the other bits are 0. A message MUST NOT be longer than 4096 bytes.
4. **Before pairing.** Messages are the JSON-RPC messages of Section 9.3, unencrypted. There is no TLS certificate, so each party's `session.hello` also carries `identity`: its public identity key, as the DER

encoding of its `SubjectPublicKeyInfo`, in base64url. The pairing exchange follows (Section 10.2), with the `Context` `OAP pairing 1` followed by the ASCII text `BLE`, and the key IDs of the two `identity` keys as the identities [OAP-SA Section 9.4.6].

5. **After pairing.** Both parties derive two 16-byte keys from the SPAKE2+ shared secret `K_shared` [RFC 9383] with HKDF-SHA256 [RFC 5869], with an empty salt and the info `OAP BLE 1 Prover` and `OAP BLE 1 Verifier`: one for the messages each sends. Every later message is encrypted with AES-128-GCM [SP 800-38D], with no additional data, and with a nonce of 4 zero bytes followed by the number of messages the sender has already sent with that key, as 8 bytes, most significant byte first. The encrypted message is the ciphertext followed by the 16-byte tag. A receiver MUST end the connection when a message fails to decrypt.
6. **Methods.** Over Bluetooth, a unit serves only `session.hello`, the pairing requests, `network.read`, and `network.configure`. The response to `pairing.confirm` is the last unencrypted message.
7. Once the unit has joined its network, everything else happens in sessions over IP, and the unit stops using Bluetooth as Security and Access Section 9.6.3 requires.

11 Carrying the Device Classes

11.1 Device Descriptions

1. `device.describe` returns the device description [OAP-DC Section 12], which MUST be valid against the Device Classes schema. For a bridged device, the request names it in `device`.
2. The description's `specRevisions` MUST include `device-protocol`, with the revision of this Specification that the device implements: `0.2` for this revision.
3. The description MUST NOT be longer than 65 536 bytes, as on the module bus [OAP-MB Section 10.2].
4. Endpoint identifiers MUST follow the rules of the module bus [OAP-MB Section 10.2.3], so that a device can move between the two bindings.
5. An endpoint's `attributes` in the description MUST NOT include state attributes. Config attribute values in the description, where given, are the manufacturer's defaults, which a factory reset restores; current values are read with `device.read`. Device Classes Section 12.1 says the same from its revision 0.3.
6. A device of Device Classes revision 0.2 or later MUST report its firmware information [OAP-SA Section 15.9] in the device attributes `updateChannel`, `securityVersion`, `firmwareSigner`, `updateFeed`, `updateKeys`, `supportEnds`, `automaticInstall`, and `lastUpdate` [OAP-DC Section 8.1], and MUST NOT carry the provisional extension `org.openaquariumproject.firmware`. The fixed ones are in its description, the others are read with `device.read`, and `securityVersion`, `firmwareSigner`, and `lastUpdate` also come in the device's state reports (Section 11.5). A device of Device Classes revision 0.1 carries its firmware information in its description instead, in `device.extensions` under `org.openaquariumproject.firmware`, and controllers read it from there [OAP-DC Section 7.11].
7. The description MUST NOT change while the device runs, except that an adapter's bridged devices change as Section 16.1 describes. Consumers read it again when `firmwareVersion` changes [OAP-DC Section 12].

11.2 Reading and Writing Attributes

1. `device.read` has the optional parameters `device`, `endpoint`, and `attributes`, a list of attribute names. It returns `attributes`: an object with the current value of each attribute named, or, when `attributes` is absent, of every config and state attribute. Without `endpoint`, it reads the device attributes [OAP-DC Section 8.1].
2. A device refuses a name it does not know with `bad-attribute`.
3. `device.write` has the optional parameters `device` and `endpoint`, and `attributes`: an object with a new value for each attribute to change. Only config attributes can be written; a device refuses any other with `bad-attribute`.
4. A write is carried out completely or not at all. If any value is outside its permitted range, the device refuses the write with `bad-parameter` and keeps every previous value [OAP-DC Section 7.3.4].
5. A list or object attribute is written whole.

6. A device MUST store the new values where they survive power loss before it responds [OAP-DC Section 7.3.3].
7. Writing `failSafeAction`, `failSafeState`, `failSafeTimeout`, `powerOnBehavior`, `powerOnDelay`, or `loadCategory` needs `change-fail-safe-settings`; writing the device attributes `updateChannel` and `automaticInstall` needs `install-firmware`; and writing any other config attribute needs `change-device-settings` [OAP-SA Table 6]. A write of attributes of more than one of these kinds needs each of their operations.
8. Writes do not count as contact, and do not resume control (Section 12.2).

11.3 Commands

1. `device.command` has the parameters `endpoint`, `command`, and `parameters`, an object of the command's parameters by name, and `device` for a bridged device. It returns `{}`.
2. The device responds when it has accepted or refused the command, not when the operation ends. Operations that take time, such as a dose, report their progress through state reports and events, as on the module bus [OAP-MB Section 11.3.2].
3. A device refuses a command with the error that applies (Table 7), and MUST NOT carry out part of a command [OAP-DC Section 7.4.2].
4. Only the device's controller may send commands, except `identify`, which any party with the `identify` operation may send [OAP-SA Table 6].
5. A device MUST NOT respond with `local-control` or `busy` to a stop from its controller, and a fade or ramp in progress never delays one [OAP-FS Section 8.2; OAP-DC Section 7.4.11]. It uses `busy` for the other commands it refuses during `powerOnDelay` [OAP-FS Section 9.1] and during a firmware update (Section 13.1).
6. A command from the device's controller counts as contact, whether or not the device accepts it. An accepted command that changes the endpoint's output resumes control of the endpoint, and one that only records, clears, shows, or silences something, such as `identify`, `silence`, `refill`, or `resetFault`, does not [OAP-FS Section 7.4.2].
7. The commands that end an operation, such as a dose, a feed, or an advance, have no `revertAfter`: the `stop` of a doser, a feeder, or a filter roller, and the `abort` of an analyzer [OAP-DC Section 7.14.2]. A device refuses one that has it with `bad-parameter`, as it refuses any unknown parameter [OAP-DC Section 7.4.3].

11.4 Stopping and Identifying a Whole Device

1. `device.stop` puts every output and unit endpoint of the device in its inactive state [OAP-FS Table 2], as one stop for each endpoint would. With the optional parameter `revertAfter`, the stops are temporary [OAP-DC Section 7.4.5], and when the revert falls due each endpoint returns to its revert target. A dose, a feed, an advance, or another operation of Device Classes Section 7.14 that the stops end MUST NOT resume, neither when the revert falls due nor at any other time [OAP-DC Section 7.14.2]. It is what a controller sends a network device for `stop-all`, with `revertAfter` for the time that `stop-all` has left [OAP-SA Section 11.8]. On the module bus, `COMMAND_EMERGENCY_STOP` stops a module in the same way, but has no `revertAfter` [OAP-MB Section 11.7], so a controller sends it for `stop-all` only as Security and Access Section 11.8.3 allows.
2. A device MUST accept `device.stop` from its controller at any time, including during a local control, during `powerOnDelay`, and during a firmware update until its final restart. It counts as a command from the controller.
3. `device.identify`, with an optional `duration` (default 10 s), makes the whole device show where it is, as `identify` does for one endpoint. It MUST NOT change any output.

11.5 Subscriptions and State Reports

1. `device.subscribe` starts state reports and events on the session. It has the optional parameters of Table 14, and returns the `minInterval` and `maxInterval` in effect.
2. A session has at most one subscription. Subscribing again replaces it, and `device.unsubscribe` or the end of the session ends it.

- After it responds to `device.subscribe`, the device sends a state report for itself and for each of its endpoints, and for each bridged device and its endpoints, and then sends reports as Table 15 requires.
- A state report, `device.state`, has the members of Table 16. It always carries every state attribute of the device or endpoint, so a report that replaces a missed or merged one loses nothing.
- A device **MUST NOT** send state reports for one endpoint, or for itself, less than 100 ms apart. Changes within that time are merged into the next report.
- A device that cannot report as often as asked **MAY** use a longer `maxInterval`, up to 60 s, or up to 3600 s for a sleepy device (Section 14.3), and **MUST** return the intervals it uses.
- A controller **MUST** subscribe to every device it controls, with a `maxInterval` of no more than 10 s unless the device returns a longer one. The `maxInterval` that the device returns is the reporting interval of its binding, and the default maximum age of each input that reads the device is at least three times that interval [OAP-FS Section 10.2.3; OAP-AU Section 9.8].

Table 14 — Parameters of `device.subscribe`

Member	Type	Default	Description
<code>minInterval</code>	number (s), 0.1 to 60	1	The least time between reports for an endpoint whose changes Table 15 allows to wait.
<code>maxInterval</code>	number (s), 1 to 3600	10	The longest time between reports for an endpoint, whether or not anything changed.
<code>devices</code>	list of strings	all	On an adapter, the bridged devices to report.
<code>eventsAfter</code>	integer	none	Send the kept events whose <code>seq</code> is greater first (Section 11.6).

Table 15 — When a device reports state

Change	Report
<code>controlSource</code> ; <code>faults</code> ; the state of an output: the attributes of its class in Fail-Safe Behavior Table 2, and any attribute that a command sets; boolean and enumerated state attributes, such as <code>wet</code> , <code>leak</code> , and <code>quality</code>	Within 100 ms of the change.
Other state attributes, such as readings and power measurements	When they change, but not sooner than <code>minInterval</code> after the endpoint's previous report.
None	At least once every <code>maxInterval</code> .

Table 16 — Members of `device.state`

Member	Type	Presence	Description
<code>device</code>	string	For bridged devices	The bridged device (Section 16).
<code>endpoint</code>	string	For endpoints	The endpoint's <code>id</code> . Absent in the report of a device's own attributes.
<code>time</code>	timestamp	REQUIRED	When the state was read, or <code>null</code> while the device's clock is not valid.
<code>attributes</code>	object	REQUIRED	The current value of every state attribute, including <code>faults</code> , and <code>controlSource</code> for output and unit endpoints.
<code>reachable</code>	boolean	For a bridged device's own report	Whether the adapter can reach the bridged device (Section 16.3).

For example:

```
{ "method": "device.state",
  "params": { "endpoint": "head_1", "time": "2026-10-14T01:02:03.250Z",
    "attributes": { "controlSource": "controller", "faults": [], "dosing": true,
      "dosedToday": 12.5, "containerRemaining": 480 } } }
```

11.6 Events

- A device reports each event of its endpoints [OAP-DC Section 7.5] to every subscribed session with `device.event`. Its members are `endpoint`; `event`; `fields`, which holds every field that Device Classes defines for the event, including `time`; `seq`; and, for a bridged device, `device`.

2. `seq` is an integer that increases with every event the unit reports, across all its endpoints and bridged devices. It MUST NOT go back while the unit keeps its identity key, not even after a restart, so that a gap shows that events were lost.
3. A unit SHOULD keep at least its 16 most recent events. When `device.subscribe` has `eventsAfter`, the unit MUST first send, in order, the kept events whose `seq` is greater, and then new events.
4. A controller SHOULD subscribe with `eventsAfter` set to the last `seq` it received from the unit, so that it records doses, fills, and results that happened while it had no session.

```
{ "method": "device.event",
  "params": { "endpoint": "head_1", "event": "doseCompleted", "seq": 118,
              "fields": { "volume": 2.5, "time": "2026-10-14T01:02:33.500Z" } } }
```

11.7 Changes to Configuration

1. When a config attribute changes, whether by a write from any party, by a command such as `completeCalibration` or `refill`, or by a local control, the device sends `device.config` to every subscribed session within 1 s. It has `device` and `endpoint` as `device.state` does, and `attributes` with the new values.
2. A controller uses it to record changes to fail-safe settings in its audit log [OAP-SA Section 12.1.2], and to show the owner changes that other parties made.

11.8 Summary

Table 17 — The Device Classes on the network

[OAP-DC] concept	Device Protocol
Device	A unit, or a bridged device of an adapter, named as Section 7.3 describes.
Endpoint	The <code>endpoint</code> member, which is the endpoint's <code>id</code> .
Classes, features, and fixed attributes	<code>device.describe</code> (Section 11.1).
Config attributes	<code>device.read</code> , <code>device.write</code> , and <code>device.config</code> (Sections 11.2 and 11.7).
State attributes, <code>controlSource</code> , and faults	<code>device.read</code> and <code>device.state</code> (Sections 11.2 and 11.5).
Commands and <code>revertAfter</code>	<code>device.command</code> (Section 11.3).
<code>identify</code>	<code>device.command</code> for an endpoint; <code>device.identify</code> for the whole device.
Events	<code>device.event</code> (Section 11.6).
Device-enforced limits (Section 7.8)	The error <code>limit</code> .
Attribute types and units	Table 8.
Extensions	<code>extensions</code> members, passed unchanged, and extension methods (Section 9.9).

12 Control and Supervision

12.1 The Controller of a Device

1. A device has at most one controller at a time [OAP-FS Section 7.5]. It stores its controller's key ID where it survives power loss.
2. A party takes control of a device with `control.take`, which returns `previous`, the key ID of the previous controller, or `null`. The device accepts it only from a party marked as a controller in its pairing list with the `owner` or `full-control` role [OAP-SA Section 11.6.3], which is the access rule `controller mark` of Section 9.8, and only when:
 - it has no controller;
 - the sender is already its controller; or
 - it reports `communication-lost` [OAP-FS Section 7.3], or, for a device without output or unit endpoints, has had no contact with its controller for 120 s.

Otherwise it refuses with `not-controller`.

3. Taking control does not count as contact, and does not resume control of any endpoint. The new controller resumes control of each endpoint with a command [OAP-FS Section 7.4].
4. A controller ends its control with `control.release`, for example when the owner removes the device from the system [OAP-SA Table 6, `remove-device`]. The device then has no controller, and each of its output and unit endpoints applies its fail-safe behavior at once [OAP-FS Section 7.3, steps 1 to 3], without reporting `communication-lost`.
5. Only a party on the local network can be a device's controller, because control runs on the controller, not in the cloud [P0001]. Apps and controllers MUST NOT mark as a device's controller a party that reaches it through the internet, such as a cloud service.
6. When a device accepts `control.take` from a party that was not its controller, each of its output and unit endpoints that is not yet in its fail-safe behavior MUST enter it at once, as Fail-Safe Behavior Section 7.5.3 requires, without reporting `communication-lost`. So every endpoint is in its fail-safe behavior before the new controller commands it, whether the device had lost contact, been released, or had its controller's entry removed.

12.2 Contact and Supervision

1. A valid message from the controller [OAP-FS Section 7.1.2] is a `control.supervise` notification, or a `device.command` or `device.stop` request, received on a session authenticated with the controller's key. Attribute writes, `control.take`, and all other messages do not count, as on the module bus [OAP-MB Table 9].
2. A controller MUST send each device it controls a valid message at least once in every period of one third of the shortest `failSafeTimeout` among the device's output and unit endpoints [OAP-FS Section 7.1.3], and at least once every 30 s. When it has no command to send, it sends `control.supervise`. It MUST send `control.supervise` at least once every 30 s even while it sends commands, so that a device keeps receiving the controller's time (Section 12.3).
3. The controller MUST create each `control.supervise` in the process that runs its control functions [OAP-FS Section 7.1.4]. Its communication task only sends what that process gives it, and MUST NOT create or repeat supervision messages on its own.
4. A controller MUST NOT send `control.supervise` or commands until its start-up is complete [OAP-FS Section 9.4]. Before then, it MAY open sessions, take control, subscribe, and read state.
5. A device measures the time since contact with a clock that does not change when the time of day is set [OAP-FS Section 7.1.5].
6. An adapter tracks contact for itself and all its bridged devices together (Section 16.3).
7. The RECOMMENDED `failSafeTimeout` for network devices is 120 s [OAP-FS Section 7.2.2].

```
{ "method": "control.supervise", "params": { "time": "2026-10-14T11:02:03.250+10:00" } }
```

12.3 Time

1. `control.supervise` carries `time`: the controller's current time, with its local UTC offset, while its clock is valid [OAP-FS Section 10.5]. Without a valid clock, the controller leaves `time` out.
2. A device that has no better source of time SHOULD set its clock from `time`, and use its UTC offset for local time, such as the local midnight at which `dosedToday` starts again. Time received from the controller is authenticated by the session, because it comes only from the controller's key [OAP-SA Section 10.3.3]. A clock set from `time` is valid as Fail-Safe Behavior Section 10.5.4 defines.
3. A device whose clock is not valid reports timestamps as `null`. The controller then uses the time it received a reading [OAP-FS Section 10.2.2].

12.4 Resuming Control

1. When a session with a device it controls ends, the controller SHOULD open a new one at once, and then keep trying at intervals that grow to no more than 10 s (provisional).
2. On each new session with a device it controls, the controller:

- sends `session.hello`, and checks `inControl` in the result;
 - takes control again if it is no longer in control and Section 12.1.2 allows it;
 - subscribes, with `eventsAfter` (Section 11.6);
 - reads the device description again if `firmwareVersion` changed; and
 - waits for a state report of each endpoint, and evaluates its control functions with current inputs, before it sends commands [OAP-FS Section 7.4.3].
3. A controller MUST NOT send again a command whose response it did not receive, until it has read the endpoint's current state and, for a command that delivers something, such as `dose` or `feed`, the events since it sent the command. A lost response therefore never makes equipment act twice.
 4. A controller MUST NOT send commands that it queued while it had no session [OAP-FS Section 7.4.3].

12.5 Loss of a Device

1. A controller MUST treat a device as lost [OAP-FS Section 7.6] when it has received nothing from the device for 30 s, or for three times the `maxInterval` in effect if that is longer (provisional). A session that ends and is opened again within that time does not lose the device.
2. A controller MUST also treat a bridged device as lost when its adapter reports it unreachable (Section 16.3).
3. A lost device's inputs are unavailable [OAP-FS Section 7.6.2]. The controller MUST treat the state of each of its outputs as unknown until that output's `failSafeTimeout` plus 2 s after the last valid message it sent the device, and as being in its fail-safe behavior after that [OAP-FS Section 7.6.3]. The outputs of a bridged device whose `failSafe` is `none` stay unknown until the controller hears from the device again (Section 16.2.4).

12.6 Summary

Table 18 — Fail-Safe Behavior on the network

[OAP-FS] concept	Device Protocol
Valid message from the controller (Section 7.1)	<code>control.supervise</code> , <code>device.command</code> , or <code>device.stop</code> , from the controller's key. Attribute writes do not count.
Supervision interval (Section 7.1.3)	One third of the shortest <code>failSafeTimeout</code> , and at least every 30 s.
Supervision from the control process (Section 7.1.4)	Section 12.2.3.
<code>failSafeTimeout</code> for network devices (Section 7.2)	120 s recommended.
<code>communication-lost</code> (Section 7.3)	In the device's own state report.
Recovery (Section 7.4)	An accepted command that changes the endpoint's output, after the steps of Section 12.4.
One controller at a time (Section 7.5)	<code>control.take</code> , <code>control.release</code> , and the controller mark in the pairing list; endpoints enter their fail-safe behavior when another controller takes control.
Loss of a device (Section 7.6)	Nothing received for 30 s, or three times <code>maxInterval</code> ; outputs unknown until <code>failSafeTimeout</code> plus 2 s after the last valid message.
Stopping is always allowed (Section 8.2)	No <code>local-control</code> or <code>busy</code> for stops, and no limit on requests; stops handled first; <code>device.stop</code> .
Device power-up (Section 9.1)	<code>busy</code> for other commands during <code>powerOnDelay</code> .
Controller start-up (Section 9.4)	No supervision or commands until start-up is complete.
Stale readings (Section 10.2)	<code>quality</code> and <code>updated</code> , reported at least every <code>maxInterval</code> , which is the binding's reporting interval.
Time (Section 10.5)	<code>time</code> in <code>control.supervise</code> , at least every 30 s; <code>null</code> timestamps.
Restarts (Section 11.1)	The session ends; <code>uptime</code> in the device's state.
Firmware updates (Section 11.2)	<code>firmware.update</code> and <code>firmware.revert</code> refused with <code>bad-state</code> while dosing, filling, or taking part in a water change (Section 13.1).
Local control (Section 12)	<code>controlSource local</code> ; the error <code>local-control</code> .
Configuration (Section 13)	<code>change-fail-safe-settings</code> ; <code>device.config</code> .
Authentication (Section 15.2)	Section 10.

13 Managing Units

13.1 Firmware Updates

1. A party with the `install-firmware` operation starts an update with `firmware.update`, with the parameters `manifest`, the release's signed update manifest [OAP-SA Section 15.6], and `path`, the name under which the sender offers the image (Section 13.2).
2. The device responds when it has started the update or refused it. It MUST refuse with `bad-state` while any of its endpoints is dosing, filling, or taking part in a water change, with the meanings that Fail-Safe Behavior Section 11.2.2 gives them, and with `busy` while another update is in progress.
3. The device then reads the image from the sender with `file.read`, on the same session, checks it as Security and Access Section 15.2 requires, installs it, and restarts. It sends `firmware.progress` to the sender whenever the update's `state` changes, and at least every 5 s while it downloads. `firmware.progress` has `state` (`downloading`, `verifying`, `installing`, `restarting`, or `failed`), and optionally `received` and `size`, in bytes, and `reason`, a message for people when the update failed.
4. If the session ends before the image is complete, the device abandons the update and keeps its firmware.
5. During the update, outputs behave as Fail-Safe Behavior Section 11.2 requires. Until its final restart, the device MUST accept commands that put an endpoint in its inactive state, and `device.stop`, and SHOULD refuse other commands with `busy`.
6. After the restart, the new firmware confirms itself as Security and Access Section 15.4.2 requires.
7. `firmware.revert` returns the device to its previous firmware [OAP-SA Section 15.4.4], and restarts it. A device MUST refuse it with `bad-state` when it has no previous firmware that its security version allows, and, because it restarts, while any of its endpoints is dosing, filling, or taking part in a water change, as for `firmware.update` (item 2). It refuses it with `busy` while an update is in progress.
8. A device of Device Classes revision 0.2 or later has its update channel and its automatic installation as the config attributes `updateChannel` and `automaticInstall` [OAP-DC Section 8.1], which a party sets with `device.write`, as Section 11.2.7 permits. `firmware.configure` sets them only on a device of Device Classes revision 0.1, as `channel` and `automaticInstall` of `org.openaquariumproject.firmware` [OAP-SA Section 15.9]; other devices need not serve it.

13.2 File Transfer

1. `file.read` has the parameters `path`, `offset`, and `length`, in bytes, and returns `data`, in base64url, and `eof`, which is `true` when the data reaches the end of the file.
2. A party serves only files that it offered to the party that asks, and refuses other paths with `bad-parameter`. It offers a file by naming it in a request: the image named in `firmware.update`, offered on the same session until the update ends; or a program file, which a controller offers to a device by writing a reference to it in an endpoint's `program`, as the Lighting Programs and Flow Programs specifications define [OAP-LP Section 16.2; OAP-FP Section 15.2], on every session with the device, for as long as that reference is the endpoint's latest `program`.
3. The party that reads chooses `length` so that the response fits in the largest message it accepts. The server MAY return less than `length`.

13.3 Network Settings

1. `network.read` returns the unit's network settings without passwords: `wifi` (`ssid` and `security`), `thread` (`networkName` and `extendedPanId`), `addresses` (a list of its IP addresses), and `connect` (Section 14.2), each where it applies.
2. `network.configure` changes them. Its optional parameters are `wifi` (`ssid`, `passphrase`, and `security`: `wpa2-personal`, `wpa3-personal`, or `wpa2-wpa3-personal`), `thread` (`dataset`: the Thread network's operational dataset, in hexadecimal), `connect`, and `confirmWithin` (s, from 60 to 3600, default 300).
3. The unit responds, then applies the settings. If, within `confirmWithin`, no session has been established again between it and the party that changed them, or its controller, it MUST return to its previous settings. An owner who mistypes a Wi-Fi password therefore never strands a device.
4. Passphrases and datasets are secrets [OAP-SA Section 8.4], and are never returned.

13.4 Restarting, Resetting, and Erasing

1. `unit.restart` restarts the unit after it responds; Fail-Safe Behavior Section 9.1 applies after the restart. A device MUST refuse it with `bad-state` while an endpoint is dosing, filling, or taking part in a water change, as for a firmware update.
2. `unit.factoryReset` factory resets the unit [OAP-SA Section 9.8] after it responds, and ends all its sessions with code 4003.
3. `unit.eraseData` erases history and other personal data without a factory reset [OAP-SA Section 16.6.2].

13.5 Audit Log and Consent

1. A device that keeps an audit log [OAP-SA Section 12.1.3] MUST serve `audit.read`. Its optional parameters are `after`, a `seq`, and `limit`, a number of records. It returns `records`, in the form of Security and Access Table 7, in order, and `more`, which is `true` when more records follow.
2. `consent.list` returns `purposes`: for each purpose for which the unit would send data off the owner's network [OAP-SA Section 16.3], its `id`, a `description`, the `destination`, and `given`, whether the owner has consented. A unit that sends nothing returns an empty list.
3. `consent.set`, with `purpose` and `given`, gives or withdraws consent for one purpose, with effect at once.

14 Network Arrangements

14.1 A Controller's Own Wi-Fi Network

A controller can host a Wi-Fi network for its devices, so that they do not depend on the home router [P0001].

1. A controller MAY host a device network. If it does:
 - the network MUST use WPA2-Personal or WPA3-Personal, with a passphrase chosen at random for each controller, which it gives to devices with `network.configure`;
 - the controller MUST give devices their addresses, and MUST be their only router;
 - the controller MUST NOT forward traffic between the device network and any other network, except traffic of a device that the owner has allowed to reach the internet, such as for consented update checks [OAP-SA Section 16], which it MAY pass through network address translation (provisional);
 - the controller MUST support at least 8 devices on it (provisional); and
 - the controller MAY offer emergency access on it, but only as Security and Access Section 14.6.1 allows: with the network's clients isolated from each other, so that a party that joins it reaches only the controller, and with emergency sessions accepted on it only while that specification's Section 14.3 allows them.
2. Devices on the device network are found with multicast DNS on that network only. Apps and services on the home network reach them through the controller, not directly.
3. A controller moves a device onto its device network by claiming it, wherever it is, and then sending `network.configure` (Appendix B.1). If the device cannot reach the controller on the new network, it returns to its previous network (Section 13.3.3).
4. A single-radio controller whose station connection to the home Wi-Fi changes channel takes its device network to the new channel too, and its devices must reconnect. A controller that hosts a device network SHOULD therefore connect to the home network by Ethernet where it can.
5. Devices on the device network depend on the controller's radio. Their fail-safe behavior covers its failure, and the device network is a radio link, so each safety function needs a path that does not use it, as Fail-Safe Behavior Section 13.4 requires.
6. When a controller that hosts a device network fails, its devices lose their network as well as their controller, and no other party can reach them there. An owner moves each of them to another network with new network settings, after a button press on the device where it allows that [OAP-SA Section 9.6.4], or by a factory reset and claiming it again [OAP-SA Section 9.8], and then adds it to the new controller (Security and Access Appendix B.4).

14.2 Devices That Connect to Their Controller

1. A device MAY open sessions itself: to its controller, which it finds by its key ID (Section 8.5), or to the parties listed in its `connect` settings (Section 13.3), each with a `url` (`wss://host:port/oap`) and the `keyId` it must present. This suits sleepy devices, and devices on networks from which the controller cannot reach them.
2. A device MUST NOT open a session to a party that is not in its pairing list.
3. Once open, such a session is like any other: both parties send `session.hello`, and the controller takes control, subscribes, and supervises as Section 12 requires.
4. Controllers MUST accept sessions from devices they have paired.
5. A device that keeps a session to its controller SHOULD open a new one at once when it ends, and then keep trying at intervals that grow to no more than 10 s, or its normal wake interval if it is sleepy.

14.3 Battery-Powered Devices on Thread

1. A sleepy device MUST NOT present output or unit endpoints (provisional), because a command that stops equipment must never wait for a radio to wake [OAP-FS Section 8.2].
2. A sleepy device keeps a session open to its controller (Section 14.2), so that it can report without opening a new connection, which costs several kilobytes over Thread.
3. It MAY return a `maxInterval` of up to 3600 s (Section 11.5.6), but MUST wake and report each change of `leak`, `wet`, and `closed` within 1 s of the change (provisional), and SHOULD wake and report each change of `faults` at once. Its other reports may be up to its `maxInterval` late.
4. A controller that uses a sleepy device's reading for control gives it, unless the owner configures a shorter one, a default maximum age of at least three times its `maxInterval` [OAP-FS Section 10.2.3; OAP-AU Section 9.8], MUST warn the owner of a shorter one, as Automation Section 9.8.9 requires, and treats the device as lost as Section 12.5 describes.
5. A request to a sleepy device waits until the device next wakes, and its response follows within 5 s of when the device receives it (Section 9.5.2).
6. Messages over Thread SHOULD be small, because each IPv6 packet larger than about 80 bytes is split into several radio frames. Sleepy devices SHOULD resume TLS sessions [OAP-SA Section 10.2.4] rather than make full handshakes.
7. A controller SHOULD ask a sleepy device whose reading a safety function uses [OAP-FS Section 13.4] for a `maxInterval` of no more than 900 s (provisional), so that it notices a silent failure of the device within 45 minutes (Section 12.5.1), and a sleepy device SHOULD accept a `maxInterval` of 900 s.
8. A controller SHOULD show the owner, for each sleepy device whose reading a safety function uses, how long a silent failure of it would go unnoticed: three times the `maxInterval` in effect.

A profile over UDP for constrained devices, with DTLS 1.3 [RFC 9147] or OSCORE, is an open issue (Appendix C, issue 17).

15 Local and Cloud Access

15.1 Access at the Same Time

1. A device MUST be able to hold at least 4 sessions at once: its controller's, and three more, such as the owner's app and a home automation service. A controller MUST be able to hold at least 8 sessions with apps and services, in addition to one with each device it controls (provisional). A unit MUST keep one of these free for its controller, or, on a controller, for an owner [OAP-SA Section 10.9.2].
2. Every session follows the same rules, whether it is local or remote: it is authenticated, and has a role. A device sends its state reports and events to every subscribed session.
3. Using one way of access MUST NOT turn off another. In particular, connecting a cloud service MUST NOT turn off local sessions, discovery, or local integrations [OAP-SA Section 13.4.3].

15.2 Services

1. A cloud service, a home automation server, or another integration takes part as a party with its own identity key: a user of kind `service` on a controller, or an entry in a device's pairing list, with the role an owner gave it [OAP-SA Section 13.4]. It is never a device's controller (Section 12.1.5).
2. A service reaches a controller through a relay (Section 15.4), or through a session that the controller opens to it. A device MAY open a session to a service listed in its `connect` settings (Section 14.2), but SHOULD instead be reached through its controller where it has one [OAP-SA Section 13.1.6].
3. A unit MUST NOT open a session to a service on the internet until an owner has given the consent that the service's purpose needs [OAP-SA Section 16.3].

15.3 Remote Access

Security and Access Section 13 sets the requirements for remote access, and describes its architectures. A VPN to the home network needs nothing from this Specification: the app connects as if it were at home. Owner-run relays and relay services use the relay protocol of Section 15.4, so that owners can switch between them and any app works with any relay. A controller tells apps which relays it listens on in the result of `session.hello` (Table 6).

15.4 Relay Protocol

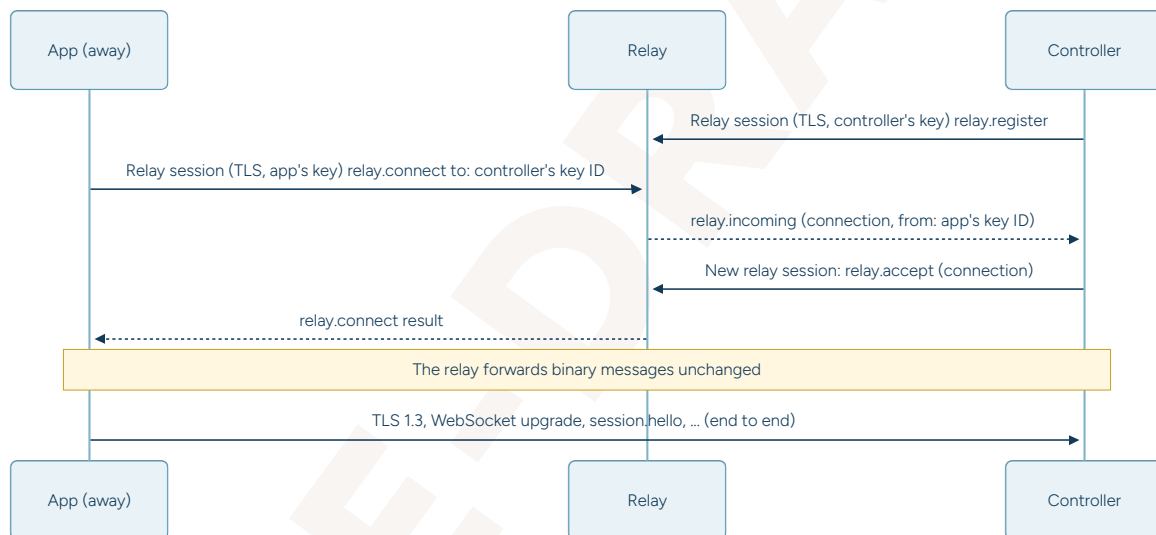


Figure 2 — A remote session through a relay

1. A relay is a WebSocket server on the internet, such as `wss://relay.example.net/oap-relay`. Relay sessions use TLS 1.3 and the WebSocket subprotocol `oap-relay.0.1`. The relay requests a client certificate, and controllers and apps present their identity certificates. Controllers and apps check the relay's certificate against the key ID configured with its URL if there is one, and otherwise as HTTPS clients do [RFC 9110 Section 4.3.4].
2. A relay session carries JSON-RPC messages, as Section 9.3 describes, until it begins to carry a connection. After that it carries only binary WebSocket messages, which the relay forwards unchanged to the other session of the connection. Relay sessions do not use `session.hello`.
3. **Listening.** A controller with remote access turned on opens a relay session and sends `relay.register`. The relay identifies the controller by its key ID. The controller keeps the session open, opens a new one when it ends, and sends a WebSocket Ping at least every 30 s (provisional) so that routers keep the connection open.
4. **Connecting.** An app opens a relay session and sends `relay.connect` with `to`, the controller's key ID. The relay sends `relay.incoming` to the controller, with `connection`, an identifier the relay chooses, and `from`, the app's key ID.

5. The controller either opens a new relay session and sends `relay.accept` with `connection`, or sends `relay.refuse` with `connection` on its listening session. It SHOULD refuse a key that is not the credential of a user whose grant allows remote access, unless an invitation that allows remote access is open [OAP-SA Section 13.1.4].
6. When the controller accepts, the relay responds to `relay.connect` with `{}`, and from then on forwards binary messages between the two sessions, in order, and closes each session when the other closes. When the controller refuses, or does not accept within 10 s, the relay responds with `not-permitted` or `unreachable`.
7. Inside the forwarded stream, the app and the controller make a TLS 1.3 connection, the WebSocket upgrade, and a session, exactly as on the local network (Sections 9.1 to 9.4), with the app as client. The relay holds no key to it [OAP-SA Section 13.1.3].
8. A relay MUST NOT need anything more than a controller's key ID to serve it. Which controllers and apps a relay serves, and how long, how many, and how fast their connections may be, are for its operator to decide.

16 Adapters for Commercial Devices

16.1 Adapters and Bridged Devices

1. An adapter is a device that presents other equipment, such as a commercial pump reached through its maker's local interface or cloud service, as bridged devices [P0001]. It may run on a controller, on an optional host, or on hardware of its own.
2. An adapter returns `bridge: true` in the result of `session.hello`, and serves `device.list`, which returns `devices`: for each bridged device, its `id` (Section 7.3), `manufacturer`, `model`, optional `name`, and `reachable`.
3. Requests about a bridged device name it in `device`. Requests without `device` are about the adapter itself, which may have endpoints of its own.
4. Each bridged device has a description of its own, with standard classes and the extension of Section 16.2.
5. When a bridged device is added or removed, or becomes reachable or unreachable, the adapter sends `device.listChanged` to every subscribed session.
6. An adapter has one controller, which controls all its bridged devices (Section 12.1).
7. An adapter built into a controller names its bridged devices with the controller's key ID (Section 7.3.3).

16.2 How a Bridged Device Is Reached

The description of each bridged device carries, in `device.extensions`, the extension `org.openaquariumproject.bridge` (provisional) with the members of Table 19.

Table 19 — Members of `org.openaquariumproject.bridge`

Member	Type	Presence	Description
<code>via</code>	<code>local</code> or <code>cloud</code>	REQUIRED	<code>local</code> : the adapter reaches the equipment over the local network or a direct link, with no service on the internet. <code>cloud</code> : only through a service on the internet.
<code>protocol</code>	string	REQUIRED	The maker's interface that the adapter uses, as an extension name, such as <code>com.example.wavepump-local</code> .
<code>failSafe</code>	<code>device</code> , <code>adapter</code> , or <code>none</code>	REQUIRED	Who applies the fail-safe behavior of the bridged device's output and unit endpoints. <code>device</code> : the equipment itself, when it stops hearing from the adapter. <code>adapter</code> : the adapter, by commanding the equipment, which works only while the adapter runs and can reach it. <code>none</code> : neither.
<code>updateInterval</code>	number (s)	OPTIONAL	How often the adapter refreshes the equipment's state.
<code>commandDelay</code>	number (s)	OPTIONAL	The usual time between a command and its effect.
<code>deviceTimeout</code>	number (s)	OPTIONAL	With <code>failSafe device</code> : how long the equipment waits, after it last hears from the adapter, before it applies its own fail-safe behavior, as the equipment's maker documents it.

1. An adapter MUST state `via`, `failSafe`, and `deviceTimeout`, where it gives one, truthfully. When it does not know who applies fail-safe behavior, it MUST state `none`.

2. Where the adapter carries out part of an endpoint's behavior itself, such as the timer of a `revertAfter`, that behavior depends on the adapter too, and `failSafe` MUST NOT be `device` unless the equipment does all of it.
3. An adapter built into a controller fails with the controller, so it MUST NOT state `adapter`.
4. An endpoint whose `failSafe` is `none` still has the output attributes [OAP-DC Section 8.3], but controllers treat it as having no fail-safe behavior.
5. An adapter MUST NOT give `deviceTimeout` unless `failSafe` is `device`, and SHOULD give it whenever the equipment's maker documents the equipment's timeout.

16.3 Fail-Safe Behavior Through an Adapter

1. An adapter tracks contact with its controller as a device does (Section 12.2). When it loses contact, it MUST make each bridged output and unit endpoint whose `failSafe` is `adapter` or `device` apply its fail-safe behavior no later than that endpoint's `failSafeTimeout` plus 2 s after the last contact [OAP-FS Section 7.2.3]:
 - with `failSafe adapter`, by commanding the equipment into its fail-safe behavior; and
 - with `failSafe device`, by stopping all communication with the equipment that the equipment counts as contact at least its `deviceTimeout` before then, where the adapter gives `deviceTimeout`, and otherwise by commanding the equipment into its fail-safe behavior.

An adapter that kept talking to equipment whose fail-safe behavior is its own would keep it from ever failing safe, for example a heater kept on after its controller has failed.

2. When the adapter cannot reach a bridged device, it reports it unreachable, with `reachable false` in the bridged device's own state report and in `device.list`, within 30 s or its `updateInterval`, whichever is longer.
3. Commands to an unreachable bridged device are refused with `unreachable`.

16.4 Keeping Dependent Equipment out of Safety Functions

The Project Lead decided where these rules draw the line on 16 October 2026 (Appendix C, issue 5).

1. A controller MUST NOT use a reading of a cloud-dependent device as an input of an interlock, of a mode's start condition, or of a control function [OAP-AU Sections 9 to 14]. It MAY show and record such readings.
2. A controller MUST NOT bind an output that drives stop-on-loss equipment to an endpoint of a bridged device that is cloud-dependent, or whose `failSafe` is not `device`.
3. A controller MUST show the owner `via` and `failSafe` for every bridged device, and `deviceTimeout` where the adapter gives it, and MUST warn the owner when they bind any endpoint of a cloud-dependent device.
4. A link to a bridged device is a radio link, whatever the equipment's own connection [OAP-FS Section 13.4.4], so a bridged device gives a safety function a path that uses no radio link only through protection inside the equipment [OAP-FS Section 13.4.2].
5. An adapter MUST NOT present a bridged device as anything other than bridged, and MUST NOT leave out its `org.openaquariumproject.bridge` extension.
6. An adapter MUST keep the credentials of the maker's services as secrets [OAP-SA Section 8.4]. It MUST NOT connect a cloud-dependent device until an owner has consented to what it sends to the maker's service [OAP-SA Section 16.3].
7. When an owner binds an output that drives run-on-loss equipment to an endpoint of a bridged device that is cloud-dependent, or whose `failSafe` is not `device`, a controller MUST warn the owner that a temporary stop of it, such as a pause for feeding or `stop-all`, may last until the adapter, or the maker's service, returns. It MUST NOT refuse the binding for that reason.

17 Conformance

17.1 Devices

A conforming device:

1. meets Sections 7 to 13, and Sections 14 and 15 where they apply to it;
2. serves `session.hello`, the pairing requests of Sections 10.2, 10.5, and 10.6, `control.take`, `control.release`, `device.describe`, `device.read`, `device.write`, `device.command`, `device.stop`, `device.identify`, `device.subscribe`, `device.unsubscribe`, `firmware.update`, `firmware.revert`, `network.read`, `unit.restart`, `unit.factoryReset`, `unit.eraseData`, `consent.list`, and `consent.set`; `network.configure` if it uses Wi-Fi or Thread; `audit.read` if it keeps an audit log; and `firmware.configure` if it implements Device Classes revision 0.1;
3. implements the Device Classes and Fail-Safe Behavior specifications for its endpoints, and meets Security and Access Section 19.1; and
4. passes the tests of Table 20 that apply to devices.

17.2 Controllers

A conforming controller:

1. meets Sections 7 to 15 as they apply to controllers, and Section 16.4;
2. serves `session.hello`, `session.signIn`, `pairing.start`, `pairing.share`, `pairing.confirm`, `pairing.invite`, and `file.read`;
3. as a client, claims, controls, supervises, and updates devices with the requests of Sections 10 to 13;
4. implements the relay protocol of Section 15.4 if it offers remote access through a relay; and
5. passes the tests of Table 20 that apply to controllers.

A controller MAY also present its own endpoints as a device does.

17.3 Apps and Services

A conforming app or service meets Sections 8, 9, 10.1 to 10.5, and 15, and Section 12.1.5 when it manages pairing lists.

17.4 Adapters

A conforming adapter meets the requirements for devices and Section 16.

17.5 Relays

A conforming relay meets Section 15.4.

17.6 Machine-Readable Definitions and Tools

`protocol.json` is normative. It defines the methods, error codes, and discovery keys, and Tables 3, 7, and 9 are generated from it by `tools/render_spec_tables.py`. The JSON Schema `schema/messages.schema.json` [JSON Schema] is normative for the structure of messages. Where the schema and this document differ, this document governs.

The validator, `tools/validate_device_protocol.py`, is informative. It checks recorded message exchanges against the schema and against rules a schema cannot express: that sessions begin with `session.hello`, that responses match requests, that methods go to parties that serve them, that the access rules and roles permit what succeeded, that only a device's controller commands it, that supervision and the controller's time keep to their intervals, that devices report firmware information as their revision of the Device Classes requires, that commands that end an operation carry no `revertAfter`, that only offered files are served, and that setup codes and discovery records are well formed. The `examples` directory contains message exchanges that pass it (Appendix B).

17.7 Test Procedures

These tests are in addition to those of the Fail-Safe Behavior and Security and Access specifications, which also apply. Each test names the requirements it checks. Every MUST and MUST NOT of Sections 7 to 16 is checked by at least one test.

Table 20 — Test procedures

Test	Requirements	Method	Pass criteria
Networks	7.1.2 to 7.1.4, 8.5.3	Give a device on Wi-Fi only IPv4 with DHCP, on a network with no internet access, and change its address. Then put it on another network segment, and enter its address and port by hand in a controller and an app.	The device works with no internet access, and the controller finds it again by its key ID at the new address. The controller and the app reach it at the address entered.
Discovery	8.1.1, 8.1.3 to 8.1.5, 8.2.1, 8.2.2, 8.3	Browse for <code>_oap._tcp</code> while a unit runs, then open and close a pairing window, then stop the unit accepting sessions. Repeat with a controller.	The unit is found on each interface on which it accepts sessions, with the TXT keys of Table 3 and no others, and a TXT record of no more than 200 bytes; its instance name, host name, and TXT record hold no owner names or serial numbers. The <code>_pairing</code> subtype, <code>pw</code> , and <code>d</code> appear and disappear within 1 s, and goodbye records are sent when it stops. The controller is also found under <code>_controller</code> .
Discovery on Thread	8.4	Put a device on a Thread network with a border router, browse from the controller's network, and open and close a pairing window. Repeat with a controller that is itself the border router.	The device is found, with the same records, and its registration changes within 1 s of the window opening and closing. The controller that is the border router finds it too.
Bridged identifiers	7.3.3	Read <code>device.list</code> from an adapter before and after it restarts.	Each identifier has 1 to 64 lowercase letters, digits, hyphens, and underscores, begins with a letter or digit, and is the same after the restart.
Mutual TLS	9.1.3, 9.5.6, 10.1.2	Connect without a client certificate; then with an untrusted key, and send each method. Look for a web interface on the port of the SRV record.	The first is refused. The second gets <code>not-trusted</code> for everything except <code>session.hello</code> , the pairing requests of Section 10.2, and, on a controller, <code>session.signIn</code> . The port serves no web interface.
WebSocket rules	9.2.2 to 9.2.5, 9.10.1	Upgrade without the subprotocol; with an <code>Origin</code> header; with <code>permessage-deflate</code> offered. Send a message split into several frames, a binary message, and a WebSocket Ping.	Status 400; status 403; no extension accepted. The split message is accepted; the session closes with 1003 after the binary message; a Pong answers the Ping.
Hello first	9.4.1, 9.4.4	Send <code>device.read</code> before <code>session.hello</code> . Then send <code>session.hello</code> from an untrusted key.	The error is <code>bad-state</code> . The result of the second reveals no more than <code>kind</code> , <code>maxMessageSize</code> , <code>claimed</code> , <code>pairingWindow</code> , and <code>peer.trusted</code> .
Messages	9.3.2 to 9.3.5, 9.3.8, 9.5.1 to 9.5.3, 9.6.3	Record every message a unit sends during the other tests. Send it a batch, a request whose <code>params</code> is not an object, and a request with an unknown member; then two attribute writes and a command to one endpoint in quick succession.	Every message is valid against the schema, uses only the error codes of Table 7, and reuses no outstanding request <code>id</code> , and every request gets exactly one response within 5 s. The batch gets one <code>invalid-request</code> , and the bad <code>params</code> <code>invalid-params</code> ; the unknown member is ignored; the writes and the command are carried out in the order sent.
Message size	9.3.6	Send a message larger than the receiver's <code>maxMessageSize</code> ; ask for a response larger than the sender's. Send a device a message of 8192 bytes, and a controller one of 131072 bytes.	The session closes with 1009; the error is <code>too-large</code> ; the 8192-byte and 131072-byte messages are accepted.
Revoked keys	10.1.3	Remove an entry from a device's pairing list while its key has two sessions open with the device.	Both sessions close within 5 s, with code 4001.
Pairing	10.2.1, 10.2.4, 10.2.6, 10.3.2, 10.5.1, 10.5.2, 10.6.2, 10.6.3	Send <code>pairing.start</code> with no window open, and again during another exchange. Give a Prover a wrong <code>confirmV</code> . Claim one device from a controller, and another from an app. Create two invitations on a device,	<code>no-pairing-window</code> ; <code>busy</code> . The Prover does not send <code>pairing.confirm</code> . The controller's entry is marked as a controller, and the app's is not. The invitations have different slots; the device accepts the <code>verifier</code> , returns no code twice, and stores only verifiers. <code>bad-state</code> ; <code>bad-parameter</code> .

Test	Requirements	Method	Pass criteria
		one with a <code>verifier</code> , and read the device's storage. Try to remove a device's last <code>owner</code> entry, and to mark an entry with the <code>view</code> role as a controller.	
Setup codes	10.4.1	Read the QR code of a unit, add a further colon and text to it, and pair with it.	The QR code has the form of Section 10.4; the app ignores the added text, and pairing succeeds with the unit whose discriminator the code names.
Sign-in	10.7.3	Sign in to a controller repeatedly with a wrong password.	Each attempt is refused with <code>sign-in-failed</code> , and the attempts are limited as Security and Access Section 10.5.4 requires.
Bluetooth setup	10.8	Claim a unit and give it Wi-Fi settings over Bluetooth, capturing the radio traffic. Send a message longer than 4096 bytes, and an encrypted message that fails to decrypt.	Only <code>session.hello</code> and the pairing exchange are unencrypted. The long message is refused, and the bad one ends the connection. Afterwards the unit joins the network and stops advertising.
Description	11.1.1 to 11.1.7	Read the description of a device, and read it again later while the device runs.	It is valid against the Device Classes schema, has no state attributes, lists <code>device-protocol</code> with the device's revision in <code>specRevisions</code> , is no larger than 64 KiB, and has endpoint identifiers that the module bus allows. A device of Device Classes revision 0.2 or later gives its firmware information as device attributes, without <code>org.openaquariumproject.firmware</code> . The description does not change.
Write all or none	9.5.6, 11.2.6, 11.2.7	Write two config attributes, one of them out of range, and then a valid write. Write <code>failSafeTimeout</code> , and then <code>updateChannel</code> , from a party whose role permits <code>change-device-settings</code> but neither other operation. Restart the device.	<code>bad-parameter</code> , and neither attribute changes. The valid write is reported with <code>device.config</code> , and kept after the restart. The other two writes are refused with <code>not-permitted</code> .
Command errors	11.3.3, 11.3.7	Send commands that should produce each error of Table 7 from 1 to 9, and a doser's <code>stop</code> with <code>revertAfter</code> .	Each gets its error, the last <code>bad-parameter</code> , and no command is carried out in part.
Commands from others	9.5.6, 9.9.3	From an owner's app that is not the device's controller, send a command, an extension method that changes an output, and then <code>identify</code> .	<code>not-controller</code> for the first two; <code>identify</code> is accepted.
State timing	11.5.5, 11.5.6	Subscribe with the defaults, switch an output, and change a reading quickly. Subscribe again with a <code>maxInterval</code> shorter than the device can meet.	The output's report comes within 100 ms; reading reports come no sooner than <code>minInterval</code> apart; every endpoint reports at least every <code>maxInterval</code> ; no two reports for one endpoint are less than 100 ms apart. The device returns the intervals it uses, with a <code>maxInterval</code> of no more than 60 s.
Subscriptions and maximum ages	11.5.7, 14.3.4	Let a controller take control of a device that returns a <code>maxInterval</code> of 60 s, and configure an input that reads it with a <code>maxAge</code> of 120 s.	The controller subscribes with a <code>maxInterval</code> of no more than 10 s, gives the input a default maximum age of at least 180 s, and warns the owner of the configured 120 s.
Supervision	12.2.2, 12.2.3	Stop the controller's supervision while keeping its session open; then send only attribute writes; then stop its control functions while its communication keeps working. Record the messages of a controller that sends a device a command every 5 s for 5 minutes.	In each of the first three cases, endpoints take their fail-safe behavior between <code>failSafeTimeout</code> and <code>failSafeTimeout + 2 s</code> , and <code>communication-lost</code> is reported. The busy controller sends a valid message in every third of the shortest <code>failSafeTimeout</code> , and <code>control.supervise</code> , with its time, at least every 30 s.
Controller start-up	12.2.4	Restart a controller, and record what it sends each device.	It sends no <code>control.supervise</code> and no command before its start-up is complete.

Test	Requirements	Method	Pass criteria
Resuming control	11.3.6	Restore supervision, send <code>identify</code> to each endpoint, and then a command that changes its output.	Endpoints stay in their fail-safe behavior after <code>identify</code> , until each is commanded, and then report <code>controlSource controller</code> .
Taking control	10.6.4, 12.1.6	With a second controller marked in the pairing list, send <code>control.take</code> while the first has contact; again after <code>communication-lost</code> ; then send a command from the first. Then, while the first controls an endpoint whose <code>failSafeTimeout</code> is 3600 s, remove its entry, and send <code>control.take</code> from the second at once.	Refused with <code>not-controller</code> ; accepted; the first controller's command is refused with <code>not-controller</code> . At the second takeover, every output and unit endpoint that is not in its fail-safe behavior enters it at once, without <code>communication-lost</code> .
Release	12.1.4	Send <code>control.release</code> .	Every output and unit endpoint takes its fail-safe behavior at once, without <code>communication-lost</code> .
No controller in the cloud	12.1.5	From an app and from a controller, try to mark as a device's controller a cloud service that reaches the device through the internet.	Neither app nor controller does it.
Lost device	12.5.1 to 12.5.3	Disconnect a device whose heater outlet has a <code>failSafeTimeout</code> of 120 s, with an interlock that lets a chiller run only while the heater is off. Then make an adapter report a bridged device unreachable.	The controller treats the device as lost within 30 s, or three times <code>maxInterval</code> , keeps the chiller stopped until 122 s after its last valid message to the device, and then treats the heater outlet as off. It treats the bridged device as lost at once.
Lost response	12.4.3, 12.4.4	Drop the session after a <code>dose</code> command, before its response. Then queue a command while there is no session.	The controller reads state and events before it decides whether to dose again, and never doses twice. It never sends the queued command.
Stopping	9.5.4, 9.5.5, 11.3.5, 11.4.2, 13.1.5	Send <code>device.stop</code> under local control, during <code>powerOnDelay</code> , and during a firmware update. Send a stop while the device is at its limit of requests, behind several requests it has not yet started, and while an endpoint fades.	Every output stops each time. The stop is never refused with <code>busy</code> or <code>local-control</code> , is carried out before the requests not yet started, and ends the fade.
Temporary stops	11.4.1	Send <code>device.stop</code> with <code>revertAfter</code> 60 to a device while one of its dosing heads doses and its pump runs its program, and wait 60 s.	Everything stops. After 60 s the pump returns to its program, and the dose does not resume, then or later.
Identifying a device	11.4.3	Send <code>device.identify</code> while outputs are on.	The device shows where it is, and no output changes.
Event replay	11.6.2, 11.6.3	End a session, cause an event, restart the device, reconnect, and subscribe with <code>eventsAfter</code> .	The kept events after <code>eventsAfter</code> are delivered first, in order, and then new ones; <code>seq</code> does not go back across the restart.
Firmware update	13.1.2, 13.1.5, 13.1.7, 13.4.1	Update a device; start an update, a <code>firmware.revert</code> , and a <code>unit.restart</code> while it doses; end the session partway through an update. Ask a device with no previous firmware that its security version allows to revert.	The update completes. The three requests during the dose get <code>bad-state</code> . The device keeps its firmware when the session ends. The revert gets <code>bad-state</code> .
Network settings	13.3.3	Send <code>network.configure</code> with a wrong Wi-Fi password.	The device returns to its previous network within <code>confirmWithin</code> , and the controller reaches it again.
Device audit log	13.5.1	Perform audited operations on a device that keeps an audit log, and read it with <code>audit.read</code> .	The records come back in order, in the form of Security and Access Table 7.
Device network	14.1.1	Put devices on the controller's device network, look for traffic between it and the home network, and add devices until there are 8. Offer emergency access on it, and	The network uses WPA2-Personal or WPA3-Personal with a passphrase unique to the controller, and the controller is the devices' only router; nothing is forwarded except what the owner allowed; 8 devices work. The emergency

Test	Requirements	Method	Pass criteria
		try an emergency session while the home network works, and after it has been down for 2 minutes.	session is refused while the home network works and accepted after, and a party that joins the network reaches only the controller.
Devices that connect	14.2.2, 14.2.4	Configure a device to open sessions to its controller, and to a party that is not in its pairing list.	The controller accepts the device's session; the device opens none to the other party.
Sleepy devices	14.3.1, 14.3.3	Read the description of a sleepy device on Thread whose <code>maxInterval</code> is 3600 s. Make its leak sensor wet and then dry, and change a level switch and a contact input.	It presents no output or unit endpoint. Each change of <code>leak</code> , <code>wet</code> , and <code>closed</code> reaches the controller within 1 s.
Sessions at once on a device	15.1.1, 15.1.3	Open sessions to a device from its controller, an owner's app, a home automation service, and one more party; then try another from a paired party; then end the service's access.	The first four work at once; the device refuses the fifth rather than the controller's; ending the service's access affects no other session.
Sessions at once on a controller	15.1.1, 15.1.3	Open sessions to a controller from 8 apps and services, one of them a cloud service through a relay, while it controls its devices; then try another, and end the cloud service's access.	All work at once, beside the sessions with its devices; an owner can still connect; ending the cloud service's access affects no other session, and turns off neither local sessions nor discovery.
Services on the internet	15.2.3	Configure a device to connect to a service on the internet, before and after an owner consents to the service's purpose.	The device opens no session to the service until the owner consents.
Relay	15.4.8	Connect to a controller through a relay that records what it forwards, giving the relay only the controller's key ID; then connect from an unknown key.	The relay serves the controller and holds only TLS records between app and controller; the unknown key is refused.
Adapter marking	16.2.1, 16.4.1 to 16.4.3, 16.4.5, 16.4.7	Add a cloud-only device through an adapter, bind its reading to an interlock and its outlet to a heater, and bind another of its endpoints to a pump. Then bind a circulation pump to a bridged device on the local network whose <code>failSafe</code> is <code>adapter</code> .	The description carries <code>org.openaquariumproject.bridge</code> with true <code>via</code> and <code>failSafe</code> ; the controller shows them, and <code>deviceTimeout</code> where given, warns the owner of each pump binding that a pause may last until the adapter or the maker's service returns, accepts both, and refuses the other two bindings.
Adapter loses its controller	16.2.2, 16.2.3, 16.2.5, 16.3.1	Stop the controller's supervision of an adapter whose bridged endpoints have <code>failSafe adapter</code> , <code>device</code> with a <code>deviceTimeout</code> , and <code>device</code> without one, while the adapter still reaches the equipment. Check an adapter built into a controller, and an endpoint whose <code>revertAfter</code> timer the adapter runs.	The equipment of each bridged endpoint applies its fail-safe behavior within its <code>failSafeTimeout</code> plus 2 s: the adapter commands the first and the third, and stops talking to the second at least its <code>deviceTimeout</code> before then. No <code>deviceTimeout</code> is given with another <code>failSafe</code> . The built-in adapter does not state <code>adapter</code> , and the endpoint whose timer the adapter runs does not state <code>device</code> .
Adapter secrets	16.4.6	Inspect an adapter that uses a maker's cloud service through every interface, before and after an owner consents to what it sends.	No interface reveals the service's credentials, and the adapter connects no cloud-dependent device before the consent.

18 Security Considerations

1. This Specification carries the Security and Access specification, and its threats, protections, and remaining risks apply [OAP-SA Sections 7 and 20].
2. **Discovery is not authenticated.** Anyone on the network can advertise any key ID or name, and so make parties try to connect to it, or hide a real unit behind a false address. Parties rely on the key each unit proves with TLS before they send anything but `session.hello` and pairing, so the worst outcome is a lost connection, which fail-safe behavior covers.

3. **Discovery is visible.** Anyone on the network can see each unit's key ID, model, and whether it is open for pairing, and could learn the key ID anyway by connecting to the unit [RFC 8882]. Units never advertise owner names or serial numbers.
4. **Parsing.** Messages come from parties that may be hostile. Receivers limit message size and nesting (Section 9.3), reject invalid UTF-8 and duplicate member names [RFC 7493], and never act on part of a message.
5. **Compression.** Compression is not used (Section 9.2.4), and TLS early data is not accepted [OAP-SA Section 10.2.3], so recorded or compressed traffic reveals nothing and cannot be replayed.
6. **Web pages.** Web pages cannot open sessions, because servers refuse requests with an `Origin` header and require a client certificate.
7. **Denial of service.** Session and request limits protect units, and a unit always keeps a session free for its controller or an owner. Limits on requests never hold back a stop (Section 9.5.5). An attacker who can block traffic can still force devices into their fail-safe behavior, which is safe by design.
8. **Relays.** Relays see when, how often, and from where owners connect, but not what they do [OAP-SA Section 20].
9. **Device networks.** Device networks keep devices off the home network, which reduces what an infected computer at home can reach, but make the controller the devices' only route.
10. **Bluetooth.** Before pairing, only the pairing exchange is unencrypted, and it reveals nothing that allows offline guessing of the setup code.
11. **Adapters.** Adapters hold the credentials of makers' services, and anyone who takes over an adapter can command its bridged equipment. Cloud-dependent equipment is also exposed to its maker's service, which is why it is kept out of safety functions (Section 16.4).
12. **Time.** Time from the controller is authenticated, but a controller that trusts a false time source passes it on [OAP-SA Section 20].
13. **Lost responses.** Lost responses could make equipment act twice if commands were repeated blindly; Section 12.4.3 prevents it.

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	
A device, a controller, and an app from different implementations have claimed, controlled, and updated each other.	Not started	
The implementations have passed the test procedures in Section 17.7, and those of the Fail-Safe Behavior and Security and Access specifications.	Not started	
Every normative reference is publicly available.	Partly	All are free to read, but the Bluetooth Core Specification restricts its use to members of the Bluetooth SIG (Appendix C, issue 9).
The protocol definitions, schema, and examples are published, and the examples pass the validator.	Done	
The Open Issues appendix is empty, or each open issue is explicitly deferred.	Not started	
No unresolved Exclusion Notices for this Specification are recorded in <code>Notices.md</code> .	Done	
The Maintainer has recorded consensus to approve.	Not started	

Appendix B — Message Exchanges (Informative)

The `examples` directory holds complete exchanges, which the validator checks:

- `adding-a-doser.json`: a controller claims a new network doser with its setup code, adds the owner's key, takes control, subscribes, doses, and supervises.
- `loss-of-contact.json`: a power strip loses contact with its controller during a Wi-Fi outage, takes its fail-safe behavior, and is resumed.
- `home-automation.json`: an owner invites a home automation service onto a device, which reads its state but cannot command it.
- `adapter.json`: an adapter presents a commercial light on the local network and a cloud-only wave pump.
- `firmware-update.json`: a controller updates a doser's firmware, after first being refused while it doses.
- `relay.json`: an owner away from home reaches the controller through a relay.

Cryptographic values in the examples are random placeholders of the right length, not a valid exchange.

B.1 Adding a Network Doser

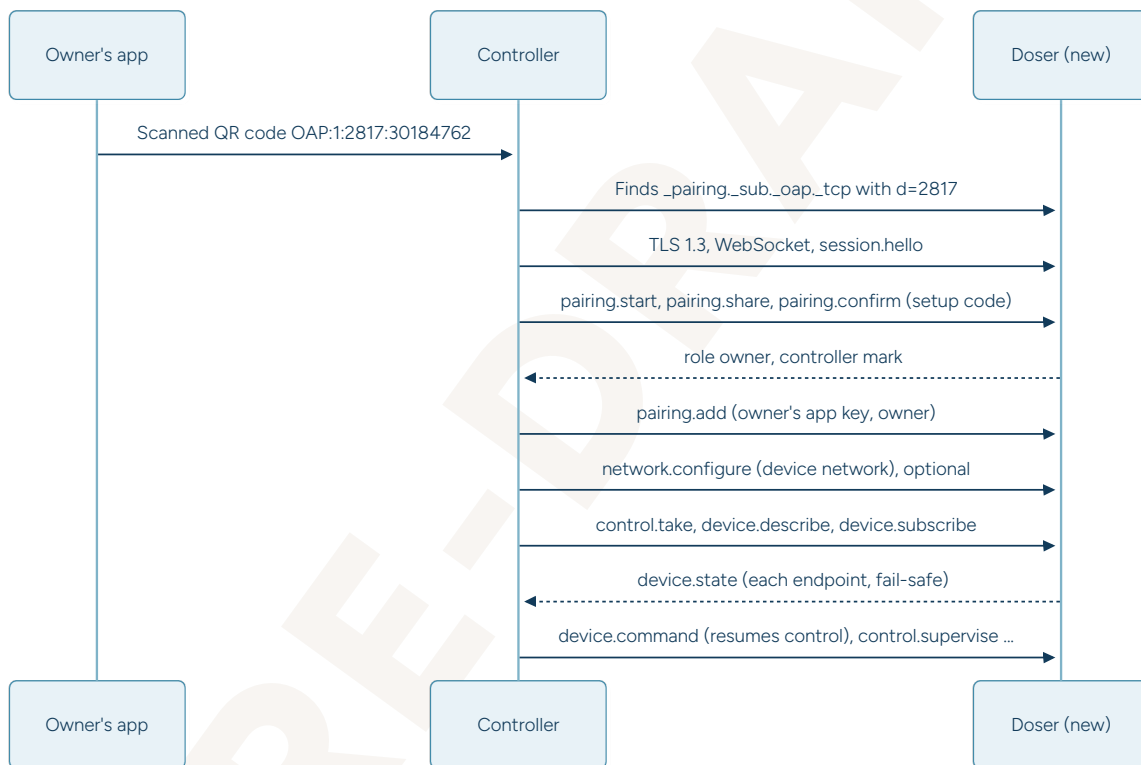


Figure 3 — Adding a network doser

B.2 Loss of Contact and Recovery

1. At 0 s, the home Wi-Fi fails. The controller's supervision stops reaching a power strip whose outlets have a `failSafeTimeout` of 120 s.
2. At 30 s, the controller has received nothing from the strip, and treats it as lost: the strip's temperature probe becomes unavailable, and a fault is raised. The state of each outlet is unknown, so an interlock that lets a chiller run only while the heater is off treats the heater as possibly on (Section 12.5.3).
3. 120 s after the last supervision message that got through, each outlet takes its fail-safe behavior: the heater outlet turns off, the return pump outlet stays on, and the strip reports `communication-lost`. 2 s later, the controller treats each outlet as being in its fail-safe behavior: the heater as off, and the return pump as on.

- At 180 s, Wi-Fi returns. The controller opens a session, finds that it is still the strip's controller because no other controller took over, subscribes with `eventsAfter`, and receives a state report of each outlet with `controlSource fail-safe`.
- It evaluates its control functions, and commands each outlet. Each reports `controlSource controller`, and the strip removes `communication-lost`.

B.3 A Cloud-Only Pump Behind an Adapter

The adapter presents the pump as a bridged device:

```
{ "id": 3, "result": {
  "oapDeviceClasses": "0.3",
  "device": { "manufacturer": "Example Reef", "model": "Wave 30", "firmwareVersion": "2.3.1",
    "specRevisions": { "device-classes": "0.3", "device-protocol": "0.2" }, "optional": [],
    "extensions": { "org.openaquariumproject.bridge": { "via": "cloud", "protocol": "com.example.wavepump-cloud",
      "failSafe": "none", "updateInterval": 30 } } },
  "endpoints": [ { "id": "pump", "class": "pump", "features": ["wave"], "optional": [], "attributes": {} } ] }
```

A controller shows the owner that the pump depends on the maker's cloud and has no fail-safe behavior. It can use the pump for flow and feeding pauses, but refuses to bind its readings to any control function (Section 16.4). A pause ends only when the adapter carries out its revert through the maker's cloud, so if the adapter or the cloud fails during a pause, the pump stays stopped until both return: it has no fail-safe behavior of its own to restart it, which is why a controller warns the owner who binds it (Section 16.4.7).

B.4 Remote Access Through a Relay

See Figure 2. The relay sees two TLS connections, one from the app and one from the controller, and between them only the bytes of a third TLS connection, which it cannot read.

Appendix C — Open Issues (Informative)

Issues 1 to 29 were open in revision 0.1, and the Project Lead decided issue 30 on 15 October 2026. Issues 31 to 33 arose in revision 0.2. The Project Lead decided issues 5 and 31 to 33 on 16 October 2026, accepting the recommendation in each. Each issue says how revision 0.2 resolves it, which specification is to apply it, or that it is still open. Issue numbers do not change, because other specifications refer to them.

Questions for the Project Lead

- A controller API.** *Open.* Apps need more from a controller than this Specification defines: history, the automation configuration, modes, overrides, alarms, users and sharing, remote access settings, and the devices of the whole system, including modules. The recommendation is a separate Controller API specification that uses this Specification's discovery, sessions, pairing, and relays, and adds methods of its own, for example in namespaces such as `system`, `history`, `automation`, and `users`. Device makers then implement only this Specification, and the controller API can follow the Automation, Data Model, and Alarms and Notifications specifications as they change. Until it exists, `pairing.invite` (Section 10.5) is the only method of a controller's user management that this Specification defines, and a Matter bridge on a host other than the controller has no standard way to ask the controller for overrides and modes (Matter Mapping Section 7.1).
- MQTT and Home Assistant discovery.** *Open.* The recommendation is a separate gateway specification, implemented by a controller or an optional host, rather than a binding that devices implement. An MQTT broker would sit between controller and device, so supervision and authentication would no longer run end to end (Section 6.2). Home Assistant's MQTT discovery format is outside the project's control, and has no device classes for many aquarium parameters, such as ORP, salinity, dissolved oxygen, and alkalinity [HA Sensor]. A native Home Assistant integration of this Specification, built on discovery of `_oap._tcp`, is the longer-term route.
- Transport and encoding.** *Open: the Project Lead is to confirm the choice.* This Specification uses WebSocket over TLS 1.3, with JSON-RPC 2.0 messages in JSON, chosen over MQTT [MQTT] and CoAP [RFC 7252]. MQTT puts a broker between the parties (issue 2). The CoAP library for the reference

controller's microcontroller supports only DTLS 1.2, which Security and Access does not allow (issue 16), and Home Assistant has no general CoAP integration.

4. **Registered names.** *Open.* The service type `_oap._tcp` (Section 8), the WebSocket subprotocols `oap.0.1` and `oap-relay.0.1`, and the Bluetooth UUIDs (Section 10.8) are provisional. Revision 0.2 keeps the subprotocol `oap.0.1`, because it keeps the format of messages, and units say which revision they implement in the TXT key `v` (Sections 8.2 and 9.2.1). Registering the service name with IANA [RFC 6335] publishes the project's name outside the repository. IANA's request form mentions no fee, and warns that registration may take up to one to two months. The UUIDs were chosen at random, which needs no registration.
5. **Dependent equipment.** *Decided by the Project Lead on 16 October 2026; resolved in revision 0.2.* Section 16.4 keeps cloud-dependent readings out of all control, and keeps stop-on-loss equipment off bridged devices whose fail-safe behavior is not their own. This trades features for safety, for example a cloud-only heater controller that some owners might want to use. Run-on-loss equipment has the opposite risk: a temporary stop of a bridged pump, such as a pause for feeding or `stop-all`, ends only when the adapter carries out the revert, through the maker's cloud for a cloud-dependent device, so a failure of either can leave the pump stopped until both return (Appendix B.3). The Project Lead decided to keep Section 16.4 as it is, so that the line falls where revision 0.1 drew it for stop-on-loss equipment and for readings, and to warn rather than forbid for run-on-loss equipment. Section 16.4.7 now requires a controller to warn the owner who binds run-on-loss equipment to a bridged device that is cloud-dependent, or whose `failSafe` is not `device`, that a pause may last until the adapter or the maker's service returns, and not to refuse the binding. Forbidding it would rule out the commercial wave pumps that owners most often pause for feeding, while the warning lets owners choose, and the controller already raises the loss of such a device (Section 12.5.2). Section 16.4 is no longer provisional.
6. **Sleepy devices with outputs.** *Open.* Section 14.3.1 does not allow them. A battery-powered valve or doser would need a guaranteed wake interval short enough for stopping.
7. **Device networks.** *Open.* Whether a controller may forward any device traffic to the internet (Section 14.1), and how many devices it must support. The reference controller's microcontroller accepts at most 15 stations in access-point mode. Revision 0.2 lets a controller offer emergency access on its device network only as Security and Access Section 14.6.1 allows (Section 14.1.1), which the Project Lead decided on 16 October 2026 (its Appendix D, issue 34).
8. **Thread.** *Open.* The Thread specification is licensed for internal use only, and its terms say that membership of the Thread Group is needed to ship Thread products [Thread]. This Specification uses only IP and SRP, but makers of Thread devices, including open hardware, face that requirement.
9. **Bluetooth.** *Open.* Setup over Bluetooth (Section 10.8) depends on the Bluetooth Core Specification, which anyone can read, but whose notice prohibits its use by anyone who is not a member of the Bluetooth SIG [BT Core]. Makers that ship Bluetooth products join the SIG anyway, but the reference does not meet the rule that normative references are available to anyone without membership. Setup over Ethernet needs no such reference; for Wi-Fi devices, the alternative would be setup through a temporary access point on the device, which this revision does not define. The Project Lead decided on 15 October 2026 that Bluetooth is for setup only, never for emergency access (Security and Access Appendix D, issue 5), which this Specification already followed.

For other specifications

10. **Device Classes: firmware information.** *Resolved in revision 0.2.* Device Classes revision 0.2 made the firmware information device attributes (its Section 8.1). Section 11.1.6 now requires them of devices of Device Classes revision 0.2 and later, and reads `org.openaquariumproject.firmware` only from devices of revision 0.1. One method now sets the update channel and automatic installation: `device.write`, which needs `install-firmware` for them (Sections 11.2.7 and 13.1.8); `firmware.configure` remains only for devices of revision 0.1.
11. **Device Classes: optional elements.** *Resolved by Device Classes revision 0.2.* A device of Device Classes revision 0.2 or later lists the optional commands and attributes it implements in its description (its Section 12), so controllers no longer learn them from `bad-command` and `bad-attribute` errors. The examples of this revision do so.

12. **Device Classes: bridged devices.** *Answered: they stay in this Specification.* The Project Lead decided to keep `via`, `failSafe`, and `reachable` in the Device Protocol, because only this binding has adapters (Device Classes Appendix D, issue 29). Device Classes revision 0.2 allows a description with no endpoints, for an adapter with no endpoints of its own.
13. **Device Classes: state reporting.** *Resolved.* Device Classes revision 0.2 leaves reporting rates to the bindings, and says what each must report (its Section 7.5). Section 11.5 defines them for the network, and the `maxInterval` a device returns is the reporting interval from which a controller sets its default maximum ages (Section 11.5.7).
14. **Automation: identifiers and reporting.** *Resolved.* Automation revision 0.2 uses the `device-protocol:` identifiers of Section 7.3 (its Section 8.5), and takes the `maxInterval` a device returns as the reporting interval of its binding (its Section 9.8.8), which answers its Appendix D, issues 10 and 21.
15. **Fail-Safe Behavior: standby controllers.** *Deferred, with Fail-Safe Behavior Appendix C, issue 2.* A pairing list can mark several controllers, and a standby can take over after `communication-lost` (Section 12.1). Revision 0.2 puts every output and unit endpoint in its fail-safe behavior when another controller takes control (Section 12.1.6), as Fail-Safe Behavior Section 7.5.3 requires. How a standby is kept ready remains Fail-Safe Behavior Appendix C, issue 2.
16. **Security and Access.** *Resolved.* This Specification answers Security and Access Appendix D, issues 8, 9, and 27: how pairing, invitations, and emergency codes are carried, the relay protocol, and a discriminator in the QR code. Invitation and emergency codes gain a two-digit slot before their eight secret digits (Section 10.3). DTLS 1.3, which Security and Access Section 10.1.3 allows for Thread, is not available in the TLS library that the reference controller uses, which issue 17 records.

Details of this Specification

17. **A constrained profile.** *Open.* Battery devices on Thread would gain from a profile over UDP, such as CoAP with OSCORE and EDHOC, or DTLS 1.3 [RFC 9147], with CBOR. It waits for library support on the reference hardware (issue 16).
18. **CBOR.** *Open.* Whether sessions should also allow CBOR, with the same structure as JSON [RFC 8949 Section 6].
19. **Numbers.** *Deferred until the bench tests of the breadboard controller and real Wi-Fi confirm them.* The 5 s response time, 100 ms and 10 s reporting, 30 s supervision and loss times, 10 s reconnection, message sizes, and session counts need confirming. So do the 1 s within which a sleepy device reports changes of `leak`, `wet`, and `closed` (Section 14.3.3), and the 900 s `maxInterval` of a sleepy device that a safety function uses (Section 14.3.7), which revision 0.2 adds.
20. **Stop commands behind other traffic.** *Open, in part.* Revision 0.2 makes devices carry out stops before other requests, and exempts stops from limits on requests (Sections 9.5.4 and 9.5.5). All messages of a session still share one TCP stream, so a stop command can wait behind a large message. Should urgent commands have a session of their own?
21. **Events.** *Open.* Whether 16 kept events are enough, and how devices keep `seq` from going back without wearing out flash.
22. **Browsers.** *Open.* Web apps cannot present identity keys, so they cannot open sessions. A browser profile would need another way to authenticate (Security and Access Appendix D, issue 4).
23. **Other networks.** *Open.* Discovery across routed networks, with unicast DNS-SD or a discovery proxy [RFC 8766].
24. **Privacy of discovery.** *Open.* The key ID in the TXT record is a stable identifier [RFC 8882]. Units could publish a short hint instead, at the cost of more connections.
25. **QR code.** *Open.* Whether the QR code should also carry the manufacturer and model, and whether typed codes need a check digit.
26. **Device history.** *Open.* Security and Access lets a device keep history (`view-history`), but this revision has no method to read it.
27. **Reference implementation.** *Open.* A reference implementation and conformance test suite, like the `module bus simulator`, and SPAKE2+ test vectors (Security and Access Appendix D, issue 19).
28. **Program files.** *Resolved in revision 0.2.* The Lighting Programs and Flow Programs specifications write a program too long for one message to a light or pump as a reference to a file that the controller serves

with `file.read` [OAP-LP Section 16.2; OAP-FP Section 15.2]. Section 13.2.2 now says that writing the reference offers the file to the device, which answers their Appendix D, issue 4, for this binding.

29. **Time accuracy.** *Open, in part.* The Flow Programs specification asks network pumps that run their own programs in a group to keep their clocks within 20 ms of their controller's [OAP-FP Section 11.4]. Revision 0.2 sends the controller's time at least every 30 s, whatever else the controller sends (Section 12.2.2), so that a device can correct its clock as often as that specification asks. A device still sets its clock from the `time of control.supervise` (Section 12.3) without knowing how long the message took. This Specification should define a time exchange that measures the round trip, and state the accuracy a device can expect; until then, that specification suggests that a device estimate the delay as half the round trip of a WebSocket ping. Raised by the Flow Programs specification (Appendix D, issue 5).

Decided by the Project Lead

30. **Prompt reports from sleepy devices.** *Resolved in revision 0.2.* The Project Lead decided on 15 October 2026 that a sleepy device reports changes of `leak`, `wet`, and `closed` within 1 s (provisional). Section 14.3.3 makes it a requirement, with a test (Table 20), and Section 2.4 says how sleepy devices differ from other devices.

The following issues arose in revision 0.2:

31. **A whole-module stop with an end.** *Decided by the Project Lead on 16 October 2026; resolved in revision 0.2: the Module Bus needs no change.* `COMMAND_EMERGENCY_STOP` stops a module as `device.stop` stops a network device, but has no `revertAfter`, and a command without one cancels a pending revert [OAP-DC Section 7.4.7]. The Module Bus specification could have carried the time left in the command's parameter, so that a module could take a whole-module stop with an end. The Project Lead decided to keep `COMMAND_EMERGENCY_STOP` as it is: `stop-all` sends each endpoint's stop with `revertAfter`, at the immediate priority, and sends `COMMAND_EMERGENCY_STOP` only as a fallback, to a module that has not accepted the stops of its endpoints (Security and Access Section 11.8.3; Section 11.4.1). The stops of a module's endpoints take a few frames at the highest priority that commands use, so one message more would save little, and a parameter would change the firmware of every module.
32. **The timeouts of bridged equipment.** *Decided by the Project Lead on 16 October 2026; resolved in revision 0.2.* An adapter makes equipment whose fail-safe behavior is its own fail safe in time by stopping its communication with the equipment early enough, or else by commanding the fail-safe state (Section 16.3.1). To stop early enough, the adapter needs the equipment's own timeout, which makers rarely document. The Project Lead decided that `org.openaquariumproject.bridge` gives it, as the OPTIONAL member `deviceTimeout`, only with `failSafe` device, and whenever the equipment's maker documents it (Table 19; Section 16.2.5). An adapter relies on going silent only when it gives `deviceTimeout`, and otherwise commands the fail-safe state (Section 16.3.1), and controllers show the timeout to owners (Section 16.4.3).
33. **Sleepy devices in safety functions.** *Decided by the Project Lead on 16 October 2026; resolved in revision 0.2.* A sleepy device reports a leak within 1 s, but if it fails silently, its controller notices only after three times its `maxInterval`, up to three hours (Section 12.5.1). Fail-Safe Behavior already counts such a device as a radio link, so a controller warns while a safety function depends on it alone [OAP-FS Section 13.4]. The question was whether a sleepy device whose reading a safety function uses, such as a leak sensor, should also be limited to a shorter `maxInterval`, at some cost in battery life. The Project Lead decided against a hard limit: a controller should ask such a device for a `maxInterval` of no more than 900 s (provisional), so that a silent failure is noticed within 45 minutes, the device should accept it, and the controller should show the owner how long a silent failure of each such device would go unnoticed (Sections 14.3.7 and 14.3.8). The 900 s waits for measurements of battery life on Thread (issue 19).