Date: Mar 27, 2022
Version 1.1.4
ECHONET Consortium

ECHONET Lite Web API Guidelines
API specifications section
Ver.1.1.4

```
Revision record
```

Date	Version	Description
October 03, 2018	Ver.1.00	Established, published for general public
March 27, 2020	Ver.1.1.0 draft	Change in versioning policy: 1.**->1.*.*
		Changed the name of this document to "API Specifications Section".
		Added Structure of the Guidelines (Chapter 3), Applied Service Functions (Chapter 7).
		Deleted previous descriptions of 3.7 "Streamlining target use case".
		Added "Example of realizing INF using Webhook" to 5.10.
		Reorganized the contents of use case in Chapter 4.
		Added "PATCH" method to 6.5.
		Added explanation related to EPC/EDT/Action to 6.4.
		Added a list of basic model API list to Table 5-1.
		Added a API list of device object operations to Table 6-3.
		Revised wording of "user" and "client" throughout.
August 27, 2020	Ver.1.1.0	Added explanation related to Action Object to 5.7.
		Changed examples of SetGet in 6.5 to Set examples.
		Revised wording of "resource" and "service" throughout.
		Added bulks and histories to 5.5 as service examples.
October 19, 2020	Ver.1.1.1	Corrected contents in group description in 7.2 (to correct clerical errors).
June 25, 2021	Ver.1.1.2	Replaced "querySchema" in 5.7 and 7.5.2 with "urlParameters".
July 30, 2021	Ver.1.1.3	Corrected "urlParameters" clerical errors in Table 5-4.
		Added description of abort in 7.1.
		Deleted unnecessary descriptions of group description in 7.2.
		Corrected "Type" of "responses[].status" in 7.2.

Date	Version	Description
		Deleted unnecessary descriptions of history description in 7.3.
		Corrected "multipleOf" and "id/deviceId" clerical errors.
May 27, 2022	Ver.1.1.4	Renewed content of authentication/authorization in 5.11.
		Deleted "POST /devices/<device id>/echoCommands" in 6.5. Changed specifications to the nodes-supported version and added as 7.7.
		Abolished null in each description of bulk/group/history in 7.1, 7.2, and 7.3.
		Corrected clerical errors in progress key values in figure and example in 7.1.
		Added server pre-registration function in 7.2.
		Added a guideline for combined uses of defined devices in 7.6.

```
CONTENTS
```

- 1. Introduction
  - 1.1. Terms and definitions
  - 1.2. Reference standard
- 2. Target scope of ECHONET Lite Web API
- 3. Structure of the Guidelines
- 4. ECHONET Lite Web API use cases
  - 4.1. Obtaining status, control, and notification
  - 4.2. Obtaining device list/management information
  - 4.3. Bulk operation of devices
  - 4.4. Virtual devices
  - 4.5. Server logs
  - 4.6. Authentication/authorization
- 5. Guidelines for Web API models
  - 5.1. Basic policy
  - 5.2. Application name
  - 5.3. API versions
  - 5.4. Obtaining API version list
  - 5.5. Obtaining target service type list
  - 5.6. Obtaining device list
  - 5.7. Obtaining device information (device description)
  - 5.8. Property value operation of device objects (e.g. SET/GET)
  - 5.9. Handling client caches
  - 5.10. Property value notification of device objects (INF)
  - 5.11. Authentication/authorization (introducing cases)
- 6. Guidelines for mapping ECHONET Lite specifications
  - 6.1. Mapping ECHONET Lite frames
  - 6.2. Mapping DEOJ
  - 6.3. Mapping ESV
  - 6.4. Mapping EPC and EDT
  - 6.5. Mapping method and operation of ECHONET Lite device objects
  - 6.6. Action
  - 6.7. Processing errors
  - 6.8. Device information
- 7. Applied service functions
  - 7.1. Batch direction for multiple commands (bulks)
  - 7.2. Grouping devices (groups)
  - 7.3. Historical data (histories)
  - 7.4. Guidelines for additional expansion of the device list
  - 7.5. Guidelines for expansion of device information (device description)
    - 7.5.1. Adding new devices

        - 7.5.2. Expansion for existing device information
    - 7.6. Guidelines for combined uses of defined devices
        - 7.6.1. Guidelines for deviceType names
        - 7.6.2. Guidelines for property names
        - 7.6.3. Guidelines for definition of Device Description
    - 7.7. echoCommand
- 8. Conclusion
- 9. Acknowledgments

FIGURES

Figure 1-1 Assumed models of these guidelines

Figure 2-1 Basic system configuration diagram in the scope of this guideline

Figure 2-2 Target scope of these guidelines

Figure 3-1 Updating policy for API specification section and Device specification section

Figure 4-1 Use case: Monitoring, control and notification

Figure 4-2 Use case: Obtaining device list/management information

Figure 4-3 Use case: Bulk operation

Figure 4-4 Use case: virtual device

Figure 4-5 Use case: Server logs

Figure 4-6 Use case: Authentication/authorization

Figure 5-1 Example of realizing INF using MQTT

Figure 7-1 Holistic overview of bulk execution

Figure 7-2 Concurrent mode (asynchronous) and sequential mode (synchronous)

Figure 7-3 Concurrent mode: executing until d)

Figure 7-4 Sequential mode: executing until b)

Figure 7-5 Sequential mode: c) stopped after execution

Figure 7-6 Overview of group execution

Figure 7-7 Overview of histories execution

Figure 7-8 Relationship of historical data set and subset

TABLES

Table 5-1 API related to the basic model specified in this document

Table 5-2 Detailed response if obtaining device list

Table 5-3 Detailed response if obtaining device information

Table 5-4 Details of property objects

Table 5-5 Details of Action Objects

Table 6-1 Response of ECHONET Lite frames

Table 6-2 Response with ESV

Table 6-3 API related to operation of device objects

Table 6-4 HTTP status code

Table 6-5 Use cases for HTTP status codes

Table 6-6 Error response of service level

Table 6-7 Variation of error type

Table 7-1 API for batch direction of multiple commands

Table 7-2 API related to device grouping

Table 7-3 APIs for historical data

1. Introduction

This document is a guideline summarizing points of view and reference cases to be considered in order to realize service provision via Web API using ECHONET Lite standard. This guideline organizes policies that should be referred to in case that configuring systems using Web API and implementing applications, to offer ECHONET Lite models to clients via server environments and further use them to promote more use of ECHONET Lite, including development of affiliated services and applied applications. Figure 1-1 shows an overall model envisioned in these guidelines, in which various service providers (clients) can access the ECHONET Lite Web API-supported cloud (server) in a unified style, enabling operation and monitoring of IoT devices placed in users' houses/premises via the cloud. Further spread and expansion of ECHONET is expected through publication of this document.

The following Chapter 2 specifies the target scope of these guidelines, Chapter 3 specifies the structure of the guideline document and Chapter 4 clarifies use cases that may be studied. Chapter 5 describes resource design, interface design, and authentication/authorization methods as a basic policy to promote Web API, Chapter 6 describes rules and error expressions related to application (convert) of ECHONET Lite specifications to Web API. Chapter 7 discusses the guidelines for providing a more sophisticated API as an applied model.

This document is created as a guideline to indicate design policy ("API specifications section"), while a separate document provides specific data types (equivalent to property) of device objects to be converted to Web API and various service definitions as a "Device specification section".

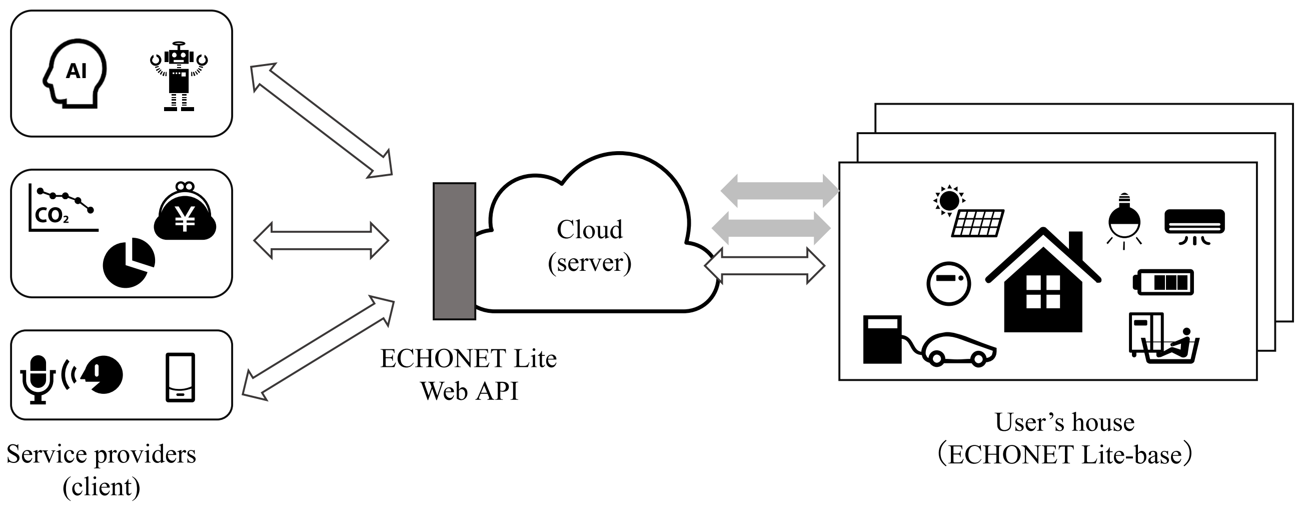


Figure 1-1 Assumed models of these guidelines

1.1. Terms and definitions

term	description
`Web API`	An interface between systems using the HTTP(S) protocol where data is exchanged using the request/response method. In this document, this term particularly refers to those to be provided by system at the called side (server side).
`RESTful`	A design principal to specify resources to be operated using URI and HTTP methods

term	description
`JavaScript Object Notation (JSON)`	A lightweight text-based and language-independent data exchange format
`OAuth2.0`	An open standard to perform authorization
`OpenID Connect`	One of the expanded specifications of OAuth 2.0, which defines the authentication/authorization function for linking IDs.

1.2. Reference standard

Standards referenced in this document are as stated below. Matters not specifically explained in this document are as described in each document.

- [EL] The ECHONET Lite Specification Version 1.01 or later
- [ELOBJ] ECHONET Specification APPENDIX: Detailed Requirements for ECHONET Device Objects Release J or later
- [HTTP] Hypertext Transfer Protocol:
    - HTTP Semantics (RFC 9110), https://www.rfc-editor.org/info/rfc9110
    - HTTP Caching (RFC 9111), https://www.rfc-editor.org/info/rfc9111
    - HTTP/1.1 (RFC 9112), https://www.rfc-editor.org/info/rfc9112
- [JSON] The JavaScript Object Notation (JSON) Data Interchange Format (RFC 8259), https://www.rfc-editor.org/info/rfc8259
- [OAUTH] OAuth:
    - OAuth 2.0 for Native Apps (RFC 8252), https://www.rfc-editor.org/info/rfc8252
    - The OAuth 2.0 Authorization Framework: Bearer Token Usage (RFC6750), https://www.rfc-editor.org/info/rfc6750
- [OIDC] OpenID Connect Core 1.0 incorporating errata set 1,https://openid.net/specs/openid-connect-core-1_0.html

2. Target scope of ECHONET Lite Web API

In this chapter, Figure 2-1 describes the system configuration assumed as a target scope.

As a rule, the model assumed should have a controller supporting ECHONET Lite in a house (one or more units can be connected), and multiple ECHONET Lite devices are connected under control of the controller (left in the figure). Two or more combinations of controllers and devices may exist. Normally, assumed cases are connected to server X (cloud) possessed by a vendor through a controller. However, cases that can be connected to server X without going through a controller can be included in the assumed cases (center in the figure). Further, even devices not supporting ECHONET Lite can be included in the scope, if they can be converted to the Web API models described in this document by mapping them to server X (right in the figure). Note that this guideline does not handle how to map these controllers and devices to server X, since it depends on implementation.

Also note that this document presupposes that server X knows client A, who is the provider of the service app beforehand, and gives appropriate authentication/authorization (or is capable of doing so). In this document, specific authentication/authorization procedures are deemed as matters to be stipulated by implementation type for each server, so only some cases are introduced here.

Negotiations between clients such as client A and client B, as well as among servers such as server X and other servers are allowed; however, this guideline does not cover such activities. In this document, a policy is established to only stipulate Web API that are provided by the server side between client and server, to simplify the discussion.

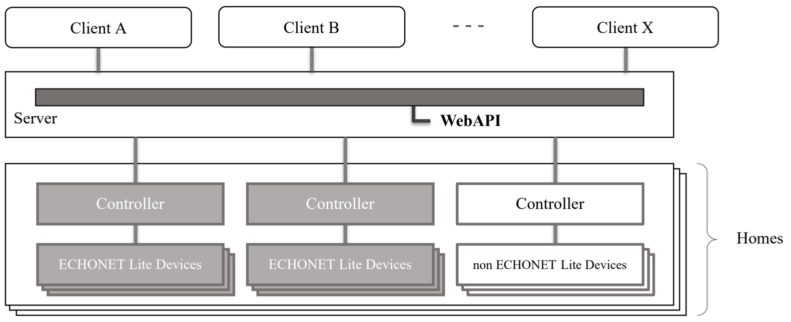


Figure 2-1 Basic system configuration diagram in the scope of this guideline

Once again, the scope of the API specified by these guidelines is shown in Figure 2-2. The ECHONET Lite Web API covers only the Web API presented from server to client. It does not cover the communication part from server to home (that is vendor-specific).

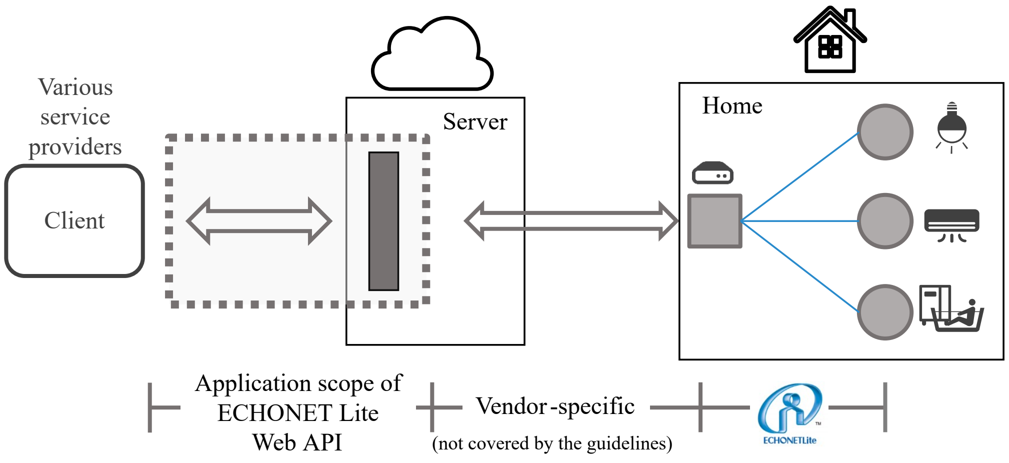

Various service providers
Client
Server
Home
Application scope of ECHONET Lite Web API
Vendor-specific
(not covered by the guidelines)
ECHONETLite

Figure 2-2 Target scope of these guidelines

3. Structure of the Guidelines

The ECHONET Lite Web API Guidelines consist of an "API specifications section" (this document) and a "device specification section" (separate document).

"API specifications section" provides use cases covered by the guidelines, Web API model policies, and mapping policies from the ECHONET Lite specifications to Web API.

The "device specification section" mainly provides examples of device descriptions for each device (schema specifications for implemented data types, and the like: discussed later), as well as data types to be used and naming guidelines.

Under the update and maintenance policies for each specification section is as follows: the API specification section will be updated in accordance with functional enhancements and specification additions for applied services and the like, while the device specification section will be updated in accordance with additions to target devices and properties.

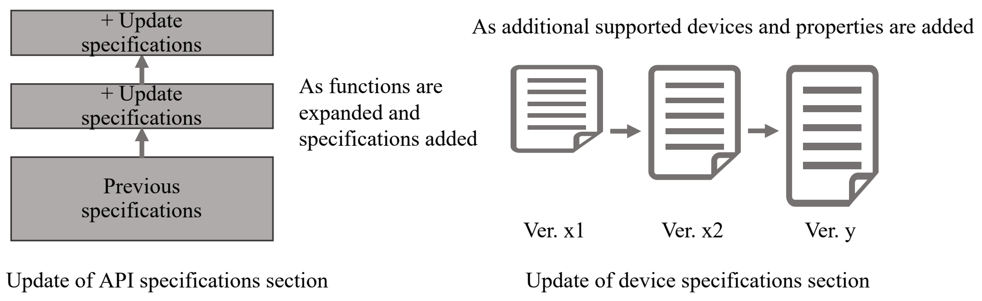

+ Update specifications
+ Update specifications
Previous specifications
As functions are expanded and specifications added
As additional supported devices and properties are added
Ver. x1
Ver. x2
Ver. y
Update of API specifications section
Update of device specifications section

Figure 3-1 Updating policy for API specification section and Device specification section

These guidelines use semantic versioning. The format must be version X.Y.Z (major version, minor version, and patch version), where the major version is increased if there is a loss of compatibility with API changes, and the minor version is increased if functions are added or changed while maintaining backward compatibility. The patch version is increased if bugs are fixed with backward compatibility.

4. ECHONET Lite Web API use cases

This chapter summarizes use cases covered by these guidelines.

4.1. Obtaining status, control, and notification

Use case that maps GET/SET/INF, a basic operation (ESV) of ECHONET Lite (Figure 4-1). For GET, in consideration of delays in response to in-home devices, cases that return data cached within the server to the client should be included. SET basically assumes operations to in-home devices. Like INF, cases that sends status change announcement and the like generated by devices to clients via the server as soon as possible are assumed.

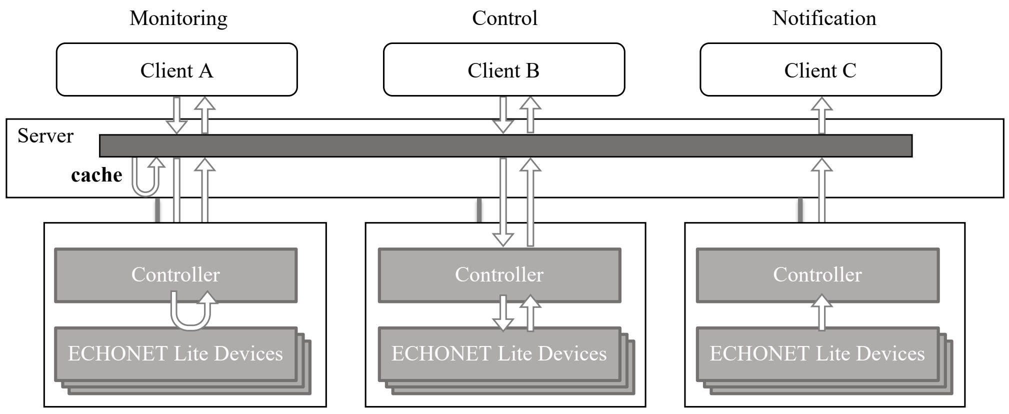


Figure 4-1 Use case: Monitoring, control and notification

4.2. Obtaining device list/management information

Use case that obtains device list (left of Figure 4-2). Potential cases should include those obtaining a list of all devices in the device user’s house or a list of devices designated by equipment type (e.g. air conditioner only). In addition to the above, potential cases should include those that handle a whole list or a part extracted from a list of devices possessed by all device users contained in the server.

ECHONET Lite controllers are equipped with a controller class as a device object , and they can retain device management information such as a list of devices that the controller manages as optional functions. Assumed cases include those that map the device management information to the server side to enable the client to obtain it. Assumed device management information includes cases where device management information cached to a server is updated with notification, only if changes (e.g. adding/deleting device) occur after caching on the server (right of Figure 4-2).

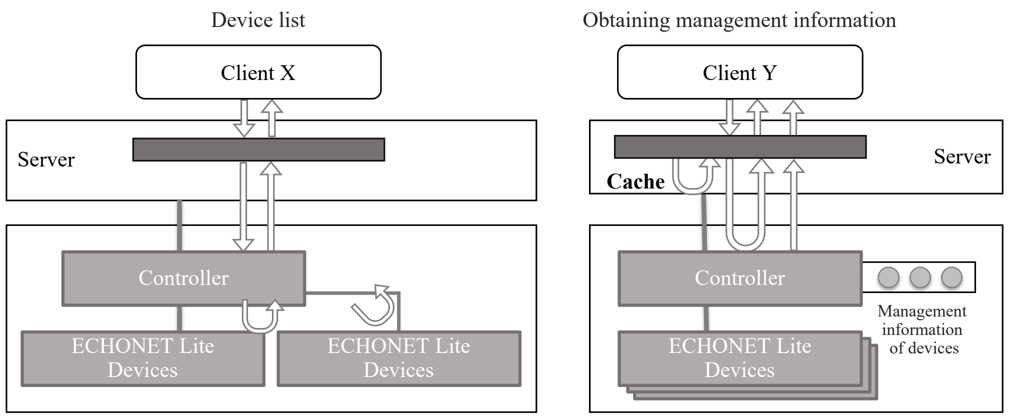


Figure 4-2 Use case: Obtaining device list/management information

4.3. Bulk operation of devices

With ECHONET Lite, bulk operations to all devices under control of a controller or bulk operation by designating a device (e.g. all air conditioners) are supported. Use cases that enable client to obtain these. Also, other cases that simultaneously operate devices in two or more device users' houses can be included.

In case that operating multiple devices simultaneously (within a home or across multiple device users), operations may be performed on the same properties for some cases, while in other cases operations are performed on different properties for each device. There may also be cases where the operating instructions (GET/SET) differ from device to device. Otherwise, multiple operations may be performed on a single device. Either of the cases above can be considered cases where multiple operations can be performed on one or more devices with a single instruction.

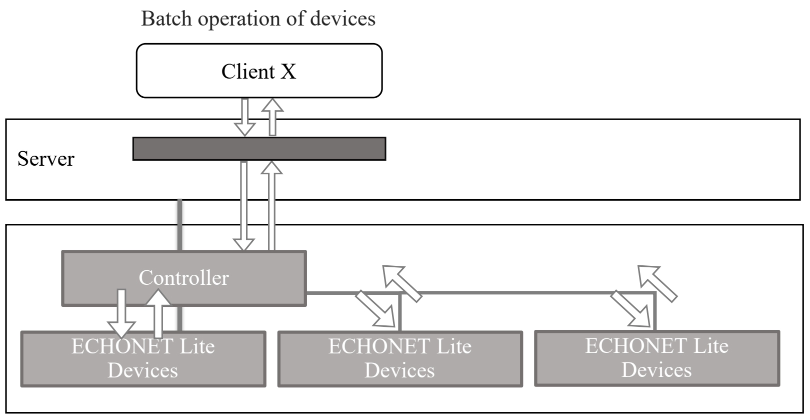


Figure 4-3 Use case: Bulk operation

4.4. Virtual devices

Use cases that can virtually extend device functions through various information processing functions if mapping in-home devices on a server. Here is a list of three assumed examples as a variation to make devices look as if they are virtual devices (Figure 4-4).

For example, in case that mapping an air conditioner supporting certain device object specifications in Appendix Release X, it is possible to make the air conditioner look to a client like one supporting Release Y, if no change in relation to the properties related to the contents corresponding with the specifications from the time of Release X to the latest Release Y occur. The benefit is that it enables air conditioners that only support previous models to support apps provided by clients which only support new models through appropriate conversion on a server, if it can be handled the same way as air conditioners supporting new models (see left side of the figure).

Or, it is possible to combine an in-home air conditioner and some kind of sensors to make them to be considered as a virtual air conditioner with new sensors so that they can be provided to client's app. It is allowed to handle air conditioners and sensors as individual devices. However, combining them as one may expand the scope of the app (see central part of the figure).

Even if an air conditioner itself does not have a particular function, it may be disclosed as an extended air conditioner to a client by adding functions on the server. An air conditioner with weather forecasting functions would work well (see right side of the figure).

As a variation of cases handling virtual devices or handling devices not supporting ECHONET Lite may be possible.

Also, collecting energy-related data retained by multiple energy-related devices (e.g. power-generating capacity, power storage capacity, and load information) to provide them to the client as virtual (EMS) information devices by integrating them within the server may be possible.

Treatment as a single virtual device may be possible if grouping multiple devices under the same purpose (for example, specifying an operation API that enables acquisition of energy information and energy control such as charging/discharging).

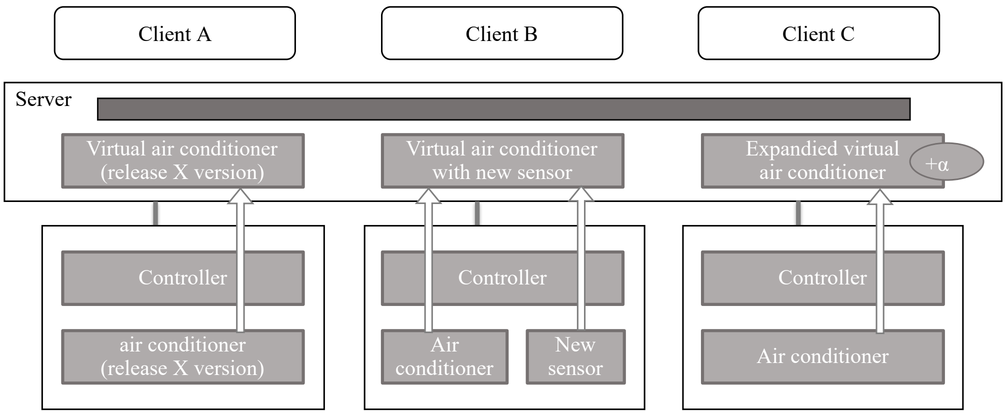

Client A
Client B
Client C
Server
Virtual air conditioner (release X version)
Virtual air conditioner with new sensor
Expandied virtual air conditioner
+α
Controller
Controller
Controller
air conditioner (release X version)
Air conditioner
New sensor
Air conditioner

Figure 4-4 Use case: virtual device

4.5. Server logs

Services accumulating data that can be obtained from in-home devices and sensors on the server as accumulated logs to use them for living activity analysis may also be possible.

This is a style to provide information summarized or processed by a device to a client, rather than that of directly-mapped in-home ECHONET Lite devices (Figure 4-5).

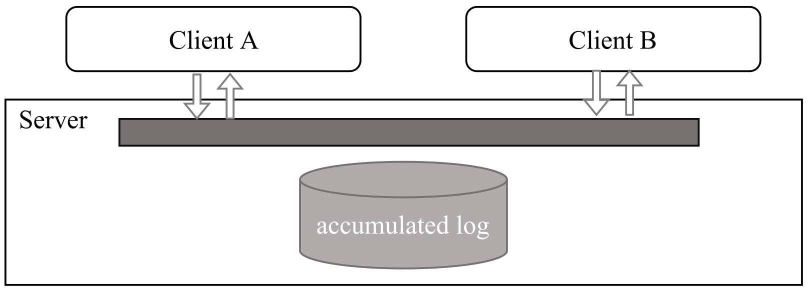


Figure 4-5 Use case: Server logs

4.6. Authentication/authorization

For cases handling customer information such as the one stipulated in this document, it is common for a client to be required to go through some kind of authentication or authorization if receiving services from servers, to begin with. Specifically, the server should prepare a resource group that admits access by each client, after establishing a session using protocols such as OAuth2.0 [OAUTH], to actually provide services.

Also, on the server, a mechanism that enables addition/deletion of customer information, and addition/deletion of customer's controllers/devices is required. Providing these client management and device management functions to the outside through Web API may be possible. However, a style performing necessary operations between the server and client beforehand may also be possible (Figure 4-6).

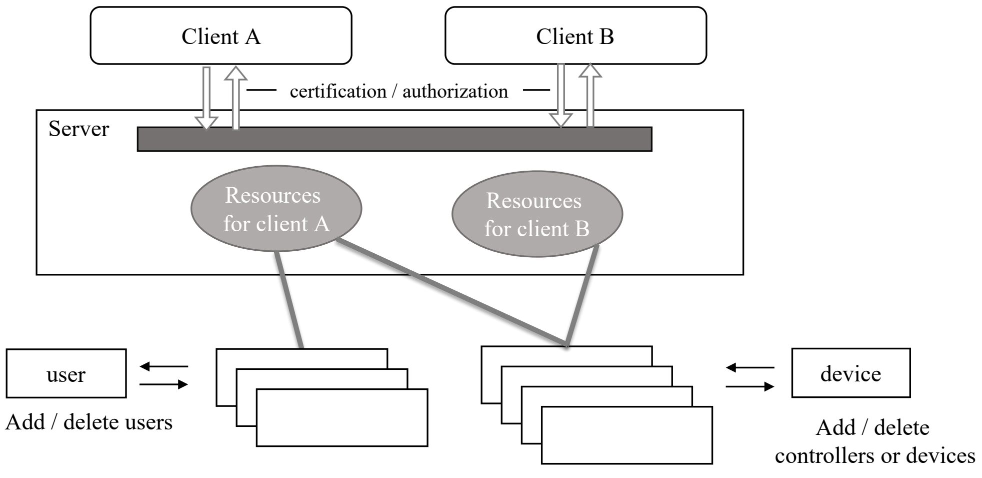


Figure 4-6 Use case: Authentication/authorization

5. Guidelines for Web API models

5.1. Basic policy

As a rule, this guideline adopts REST (REpresentational State Transfer) as a Web API model. The REST enables a simple control for URI based on the HTTP [HTTP] method. These days, the REST serves as a substantial standard of Web API. This guideline is designed as a whole to be a Rest(ful) API by adopting the currently dominant data format, JSON [JSON].

The API is designed with four principles of REST in mind: (1) statelessness by HTTP, (2) visibility of address by URI, (3) integrated interface through HTTP methods, and (4) connectivity among resources by JSON.

Identification method of resources is based on the following:

```
Format   Scheme://{host name}/{pass to resources}
Example  https://www.example.com/elapi/v1/devices/12345678
```

This example specifies a device data resource with ID 12345678 on host www.example.com using the HTTPS scheme (port number omitted). Like the above, the REST API that designates resources by hierarchization using a path is a basic model. In addition to the above, a query API that designates resources by query parameter can be complementarily selected. The following sections describe how to designate specific resources.

Many of the actions on the resource are assumed to be implemented by HTTP methods on the resource's URI or query parameters of the URI, with reference to the create, reference, update and delete operations expressed in CRUD (Create, Read, Update and Delete).

The followings are stipulated as basic policies other than the above.

- The HTTP version is HTTP/1.1[HTTP].
- The JSON format is specified in RFC 8259, specifying "`application/json`" as the Content-Type and using UTF-8 character encoding. On the client side, specify "`application/json`" as the media type in the Accept request header.
- Use HTTPS for the scheme.
- The date and time format is based on RFC 3339 (ISO 8601) and follows the notation below, taking into account the time zone.
    - 2018-01-02T12:34:56+09:00
    - However, where individual items such as the year, month, day, hour, minute and second need to be specified, they will follow the individual descriptions.
- The naming convention for property names and other attribute names specified in the resources is to use lowercase for single words, but lower camel case for compound words. For the sake of illustration, this guideline uses the '< >' bracket to express arbitrary values or descriptions.
- The device description format of ECHONET Lite device objects (see below) is partly described with reference to the WoT model being developed by the W3C.

The following sections describe examples for designating/operating the following services.

- Application name
- Obtaining API version and API version list
- Obtaining target service type list
- Obtaining device list
- Obtaining device information (device description)
- Property value operation of device objects (e.g. SET/GET)

- Handling caches
- Property value notification of device objects (INF)
- Authentication/authorization

In the examples below, the (request/response) header part of the contents of the request line, request header and message body in an HTTP(S) request and the status line, response header and message body in an HTTP(S) response are omitted unless it is necessary to specify otherwise.

Table 5-1 indicates an API list related to the basic model. The following explains the APIs individually.

Table 5-1 API related to the basic model specified in this document

http method	**path**	**description**
GET	/elapi	Obtaining API version list
GET	/elapi/v1	Obtaining target service type list
GET	/elapi/v1/<service specification optional>/devices	Obtaining device list
GET	/elapi/v1/devices/<device id>	Obtaining device information (device description) (service types are omitted)

5.2. Application name

The name of the application using the ECHONET Lite Web API specified in this document is "elapi". Embedding the application name "/elapi" as a path in the top hierarchy indicates that it is a component of the ECHONET Lite Web API.

If a vendor wishes to use a different application name than "elapi", the vendor’s preferred name may be used. In the examples given in the following parts of this document, the application name should be referred to as "elapi". However, if using the vendor’s preferred name, replace them as appropriate.

Reasons that this document selected methods to contain particular passes

> As a method to designate API, the following examples of descriptions may be suitable.
> (1). If contained in a particular path: https://www.service.xx/elapi/xxx
> (2). If distinguishing by host name: https://elapi.service.xx/xxx
> Although the form of implementation of the web application and the scale/load distribution of the service should also be taken into account, this document uses method (1), which allows for a specific path, because it is possible to change the target web application server using a reverse proxy or other means. As mentioned above, format (2) is also acceptable when using vendor-specified application names.

5.3. API versions

The API version specified in this policy is maintained in the form of a revision number, currently "`v1`". If detailed specification additions or changes are to be made in the future, if they are specified within the scope of backward compatibility, they are applied without increasing the version. If incompatible specification changes become necessary in the future, it is the policy to establish "`v2`" to distinguish between specifications (the major version is assumed to have been updated from 1 to 2). The version specification adopts the method of embedding version information in the URI.

As with the "application name" above, it is not necessary to follow this provision when implementing vendor-specific versioning. The vendor must follow the policy specified by the vendor. The examples in this document refer to the API version as "`v1`", but it should be read in the format preferred by the vendor.

Reasons to select version management by designating URI

> There are roughly three methods for designating versions.
> (1). To be included in URI: https://www.service.xx/elapi/v1
> (2). To be included in query parameters: https://www.service.xx/elapi/xx/80234512?v=1.5
> (3). To be included in the header HTTP: `Accept: application/vnd.echonet.v2+json`
> Method (2) above has the advantage of allowing omission of version information. However, if the default version is the latest, problems are likely to occur if changing API. Also, method (3) above needs to respond by giving "`Vary:Accept`" if it is possible to respond, a header together with "`Content-Type: application/vnd.echonet.v2+json`", if requesting the server as a vendor-specific media type. However, there are some other cases that do not accept (unique media type), unless it falls in line with "`Content-Type application/json`". This document adopts (1) on the grounds that it is easily identifiable and widely used by many Web API-providing services.

5.4. Obtaining API version list

Designating "`/elapi`" to the top URI to request with the GET method makes it possible to obtain a version list that the server responds to.

■ Request

```
GET /elapi HTTP/1.1
```

■ Response

```
{
    "versions": [
        {
            "id": "v1",
            "status": "CURRENT",
            "updated": "2018-01-01T12:34:56+09:00"
        },
        {
            "id": "v2",
            "status": "EXPERIMENTAL",
            "updated": "2018-01-02T01:02:03+09:00"
        }
    ]
}
```

The above examples show that they support two versions of "`v1`" and "`v2`". At least one version must be supported. The version that corresponds with this document is "`v1`", and the one to be updated in the future is "`v2`". The following conditions can be designated to "`status`" according to development level.

- `CURRENT`: latest stable version
- `SUPPORTED`: stable version (not the latest version)
- `DEPRECATED`: already abolished versions
- `EXPERIMENTAL`: versions under development/experiment

With "`updated`", date/time update can be designated, enabling the server-side system to describe a timing to make some kind of changes. Designation of "`id`" is mandatory, while "`status`" and "`updated`" are optional.

5.5. Obtaining target service type list

Designating "`/elapi/<version id>`" and demanding with the GET method enables the server to obtain a service type list immediately under "`/elapi/<version id>`".

■ Request

```
GET /elapi/v1 HTTP/1.1
```

■ Response

```
{
    "v1": [
        {
            "name": "devices",
            "descriptions": {
                "ja": "devicesの説明文",
                "en": "device resource"
            },
            "total": 10
        },
        {
            "name": "controllers",
            "descriptions": {
                "ja": "controllersの説明文",
                "en": "controller resource"
            },
            "total": 1
        },
        {
            "name": "sites",
            "descriptions": {
                "ja": "sitesの説明文",
                "en": "sites resource"
            },
            "total": 5
        },
        {
            "name": "users",
            "descriptions": {
                "ja": "usersの説明文",
                "en": "user resource"
            },
            "total": 100
        },
        {
            "name": "groups",
            "descriptions": {
                "ja": "groupsの説明文",
                "en": "group resource"
            },
            "total": 3
        },
        {
            "name": "bulks",
            "descriptions": {
                "ja": "bulksの説明文",
                "en": "bulks resource"
            },
            "total": 10
        },
        {
            "name": "histories",
            "descriptions": {
                "ja": "historiesの説明文",
                "en": "histories resource"
            },
            "total": 20
        }
    ]
}
```

The example above shows that seven service types are supported: devices, controllers, sites, users, groups, bulks and histories. This document provides examples of some of these service types, but similar service types may be specified by the server at its discretion. At least one service type should always be offered. Service types that are not offered to clients should not be described. The description of each service type is given in the "`descriptions`" (mandatory) and the total number of each object is given in the "`total`" (optional). The "`description`" must be an arbitrary string and the "`total`" must be an integer. In principle, the type of service provided to each client can be changed (switched) according to the requirements and authorizations of the client (recipient) and the content of the service provided.

The following part of this guideline stipulates device, bulk, group, and history cases. Other service types can be arbitrarily stipulated and are outside the scope of this document.

5.6. Obtaining device list

Designating "`/elapi/v1/<service designation type (can be omitted)>/devices`" to GET can obtain a device list.

■ Request

```
GET /elapi/v1/devices HTTP/1.1
```

■ Response

```
{
    "devices": [
        {
            "id": "0xFE000006123456789ABCDEF123456789AB",
            "deviceType": "generalLighting",
            "protocol": {
                "type": "ECHONET_Lite v1.13",
                "version": "Rel.J"
            },
            "manufacturer": {
                "code": "<manufacturer code>",
                "descriptions": {
                    "ja": "<manufacturer name(日本語)>",
                    "en": "<manufacturer name(English)>"
                }
            }
        },
        {
            "id": "",
        }
    ]
}
```

Table 5-2 Detailed response if obtaining device list

Property	Type	Required	Description	Example
`id`	string	Yes	Device-specific ID	"0xFE000006...AB"
`deviceType`	string	Yes	Device type	"generalLighting"
`protocol`	object	Yes	Used protocol	–
`protocol.type`	string	Yes	ECHONET Lite version number	"ECHONET_Lite v1.13"
`protocol.version`	string	Yes	Appendix release number	"Rel.J"
`manufacturer`	object	Yes	Manufacturer information	–
`manufacturer.code`	string	Yes	Manufacturer code	"0x******"
`manufacturer.descriptions`	object	Yes	Manufacturer name	–
`manufacturer.descriptions.ja`	string	Yes	Name (Japanese)	"Aベンダ"
`manufacturer.descriptions.en`	string	Yes	Name (English)	"Vendor Name A"

The "`id`" is the unique ID of the device and must be a unique value that is not duplicated in the system (under "`/devices`"). Examples of possible values include unique information held by the device itself (e.g. MAC address, unique ID of the device object, etc.), values assigned by the controller and values uniquely assigned in the server. In the example above, the values are described as hexadecimal strings with a '0x' prefix, but arbitrary strings without '0x' are acceptable. The "`deviceType`" must be a name corresponding to the class name of the device object. The class name to be mapped to the ECHONET Lite device object is described in the "Device Specification Section" (separate document). In the "`protocol`", the ECHONET Lite version number in "`type`" and the Appendix release number (EPC = 0x82) of the device object in "`version`" should be described as an ASCII character string. In the case of a device to be mapped to an ECHONET Lite device object, the manufacturer code (EPC = 0x8A) property value of the device object superclass should be entered in "`code`" and the name in Japanese ("`ja`") and English ("`en`") in "`description`". The property value of the property code (EPC = 0x8A) should be entered in code. The handling of the Required column in Table 5-2 can be flexibly changed as described in Section 7.4 if the device is independently extended.

The list of devices can be very large. If the server wishes to limit the number of devices to be returned to the client at one time, it can return the following information: the existence of subsequent devices ("`hasMore`"), the number to be returned at one time ("`limit`") and the offset position from the start ("`offset`").

```
{
    "devices": [
        :
    ],
    "hasMore": true,
    "limit": 25,
    "offset": 0
}
```

It can also support a query format where the offset position and number of returns are specified by the client.

```
GET /elapi/v1/devices?offset=0&limit=25 HTTP/1.1 |
```

5.7. Obtaining device information (device description)

By specifying "`/elapi/v1/<service specification (can be omitted)>/devices/<device id>`" and performing a GET, the specification corresponding to the device can be obtained as a device description in JSON format. The device description is a definition of the functions ("`properties`", "`actions`", "`events`") implemented by the device.

For devices mapped to ECHONET Lite device objects, the device description for each device (currently described for major devices) is described in the "Device specification section" (separate document).

■ Request

```
GET /elapi/v1/devices/<device id> HTTP/1.1
```

■ Response

```
{
    "deviceType":<device type>,
    "eoj":<eoj in Hex string>,
    "descriptions": {
        "ja": <description of property in Japanese>,
        "en": <description of property in English>
    },
    "properties": { <property1>: <property object>, <property2>: <property object> ...
```

```
        },
        "actions":  { <action1>: <action object>, <action2>: <action object>...
        },
        "events": { <event1>: <event object>, <event2>: <event object>...
        }
    }
```

Table 5-3 Detailed response if obtaining device information

Property	Type	Required	Description	Example
`deviceType`	string	Yes	This shows device type. This is equivalent to ECHONET Lite device object name (EOJ). For the values, refer to "5. Device descriptions of each device" in the "Device specification section" (separate document).	"generalLighting"
`eoj`	string	No	ECHONET Lite EOJ class code in hex notation as string (not used as ID)	"0x0130"
`descriptions`	object	Yes	Device object name defined by ECHONET Lite	—
`descriptions.ja`	string	Yes	Device object name of ECHONET Lite in Japanese	"一般照明"
`descriptions.en`	string	Yes	Device object name of ECHONET Lite in English	"General Lighting"
`properties`	object	Yes	List of "property objects". Property 1 and Property 2 are property resource names (described in the "Device specification section" (separate document)).	
`actions`	object	No	List of "action objects".Action 1 and Action 2 are action resource names (described in the "Device specification section" (separate document)).	
`events`	object	No	List of "event objects"."event1" and "event2" are event resource names.	

Property object

The property object describes the property definition of a device, corresponding to an ECHONET property that supports GET or SET in ECHONET Lite. The GET method is used to obtaining a property in this Web API and the PUT method is used to set it. The structure of the property object is shown below.

```
    {
        "epc":<epc in Hex string>,
        "epcAtomic":<epc in Hex string>,
        "descriptions": {
            "ja": <description of property in Japanese>,
            "en": <description of property in English>
        },
        "writable":<writable flag>,
        "observable":<observable flag>,
        "urlParameters": <schema of parameters>,
        "schema":<schema of data>,
        "note": {
            "ja":<note in Japanese>,
            "en":<note in English>
        }
    }
```

Table 5-4 Details of property objects

Property	Type	Required	Description	Example
`epc`	string	No	Hex notation of the corresponding ECHONET Lite EPC in string.	"0x80"
`epcAtomic`	string	No	An EPC that requires "Atomic operation" as ECHONET Lite.	"0xCD"
`descriptions`	object	Yes	An ECHONET property name defined by ECHONET Lite	—
`descriptions.ja`	string	Yes	Property name of ECHONET Lite in Japanese	"動作状態"
`descriptions.en`	string	Yes	Property name of ECHONET Lite in English	"Operation Status"
`writable`	boolean	Yes	This indicates whether writing is possible or not. Corresponds to SET of ECHONET Lite.	true, false
`observable`	boolean	Yes	This indicates whether status changes can be notified or not. Corresponds to INF of ECHONET Lite.	true, false

Property	Type	Required	Description	Example
`urlParameters`	object	No	Use query when it is necessary to pass parameters in GET; describe parameter information in JSON Schema format.	{ "day":{ "descriptions":{ "ja": "xx", "en": "yy" }, "schema": { "type": "number", "minimum": 0, "maximum": 99 }, "required": false } }
`schema`	object	Yes	Describes property data information in JSON Schema format.	{ "type":"string", "format":"time" }
`note`	object	No	Additional information related to properties	—
`note.ja`	string	No	Write additional information in Japanese	"秒の指定は無視される"
`note.en`	string	No	Write additional information in English	"number of seconds is ignored"

Action object

Action object describes a function provided by a device that is difficult to express in a property. As an example, SET-only property operations, changes to internal state of devices without defining a property, atomic changes in multiple properties, and time-consuming changes in device state (e.g., fading light over time). It is also possible to define "action" as a pure function irrelevant to the internal state of the device (returns the arithmetic results as output only for the input arguments). Although "action" does not exist in the ECHONET Lite device object definitions, it is possible to define it with "action" instead of property; for example, define a function to reset the "cumulative amount of charging electric energy" to 0 by setting the "cumulative amount of charging electric energy resetting" property for the electric vehicle chargers. Use POST method to execute "action" with this Web API. The following shows action object configurations.

```
{
    "epc": <epc in Hex string>,
    "descriptions": {
        "ja": <description of action in Japanese>,
        "en": <description of action in English>
    },
    "input": {
        "type": "object",
        "properties": {
              <input arg1>: <schema of arg1>,
              <input arg2>: <schema of arg2>,
              …
        },
        "required": <array of required args>
    },
    "schema": <schema of data>,
    "note": {
        "ja": <description of action in Japanese>,
        "en": <description of action in English>
    }
}
```

Table 5-5 Details of Action Objects

Property	Type	Required	Description	Example
`epc`	string	No	The corresponding ECHONET Lite EPC is expressed as a string in hex. Omitted if no corresponding EPC exists.	"0xB3"
`descriptions`	object	No	Name of "action".	—
`descriptions.ja`	string	No	"action" name in Japanese	"一括停止設定"
`descriptions.en`	string	No	"action" name in English	"All stop setting"
`input`	object	No	Input parameter of "action"	—
`input.type`	object	No	Type of input parameters. Its type is object fixed.	"object"
`input.properties`	object	No	List of "property objects" of input parameters	{"mode": "sleep","speed": "fast"}
`input.required`	array	No	List of mandatory "property objects"	["mode"]
`schema`	object	No	Indicates output of "action". Leave empty if no output is required	{}
`note`	object	No	Additional information related to "action"	—
`note.ja`	string	No	Write additional information in Japanese	"ECHONET LiteではSet only property"
`note.en`	string	No	Write additional information in English	"Access rule of the corresponding ECHONET Lite property is Set only"

Event object

No specific examples are given for event objects in this edition of the guidelines. Additional descriptions are expected in the next edition of the guidelines or later. 5.10 provides examples of other way to realize notification (using long polling or webhook).

5.8. Property value operation of device objects (e.g. SET/GET)

ECHONET stipulates property operations as ESV (ECHONET Lite service). Mapping ECHONET property values on the server and disclosing them as a resource to operate while linking ESV with the HTTP method (CRUD operation) enables "restful" API operation. The following chapter discusses specific mapping and operations at the device object property level.

5.9. Handling client caches

In case that operating in-home devices from a client via server (performing status obtainment and control), some processes may require a longer time to respond. Although an HTTP cache (RFC 9111) is not mandatory, client reuse of cached resources is effective in cases where values are being obtained from a client to a server. The examples are shown as a reference.

The following shows cases where effectively using HTTP caches for handling static data including manufacturer code (basically inalterable for long-term or fixed information) and data not having been updated for a certain period after the previous access (dynamic data). HTTP cache models include the "validation model" and "expiration model".

Validation Model

A model obtains only if data is updated.

- Server: uses the last update date or entity tag as needed.

```
Last-Modified: True, 02 Jan 2018 00:00:00 GMT
ETag: "fa39b31e285573ee37af0d492aca581"
```

- Client: if using a conditioned request with the last update date

```
GET /elapi/v1/devices/12345 HTTP/1.1
If-Modified-Since: True, 02 Jan 2018 00:00:00 GMT
```

- Client: if using a conditioned request with entity tag

```
GET /elapi/v1/devices/12345 HTTP/1.1
If-None-Match: "fa39b31e285573ee37af0d492aca581"
```

- Server: sends status code 304 (and empty body), if not changed, while sending 200 (and the detailed change), if changed

Expiration Model

A model obtains only if cache is expired

- Server: Specify the expiration date and time or the number of seconds from the current time

```
Expires: Wed, 03 Jan 2018 00:00:00 GMT
```

```
Cache-Control: max-age=3600
```

The former indicates that clients can cache and use the data returned in the response until the specified deadline. The latter designates elapsed seconds from the date header, since it is a non-real time case. If both are used, Cache-Control takes precedence. The date and time format in the HTTP header (the three types described in HTTP data in RFC 9112 (HTTP1.1)) must be supported.

- Client: Consider when to issue the next request with reference to the value of the response header above.

The above assume the effective use of caches. On the other hand, there are cases where no cache at all is used (is NOT allowed to be used).

Cases where you don't want to cache (explicitly)

The Cache Control header can be used to request access from the client each time (instructing the client not to cache or reuse).

- Server: clearly indicate that a validation is required (asking server if it is valid or not even now, and not allowing reuse unless it is confirmed).

```
Cache-Control: no-cache
```

- Server: clearly indicate that caching is not allowed.

```
Cache-Control: no-store
```

5.10. Property value notification of device objects (INF)

The Web APIs covered in this guide basically assume normal, synchronous HTTP REST APIs that respond instantly to client requests. Therefore, some ingenuity is required to use INF, which transmits information asynchronously to the timing of the device side.

Handling INF

As the simplest case, a method that may be feasible is if receiving INF from the device at the server side, update the cache using the values received, then detect the updates by regularly accessing them from a client (perform polling). For this communication the "validation model" explained in 5.9 can be used. This method is very simple, but comes with some problems. The first one is a constant delay in information transmission timing due to the limited timing of requests from the client. This occurs because the server cannot propagate its value to the client at the moment it receives the INF asynchronously. The second is that the server must save a history for all properties potentially receiving INF, regardless of the interest of the client in the value.

Therefore, to appropriately process the INF, it is ideal to support a push notification-type communication method that sends messages from the server side to the client. There are some known methods to send push notifications from the server.

- W3C-related issues:
    - Push API (Web Push. Single-browser notification. Can be pushed from the server. Supports Android, but not iOS)
    - Service Workers
    - Web Notification (single-browser notification. Supports Android, but not iOS)
- Smartphone app-related issues (native Push):
    - APNS (Apple Push Notification Service)
    - GCM (Google Cloud Messaging), FCM (Firebase Cloud Messaging)
- Others:
    - WebSocket
    - MQTT
    - Webhook

There is another method called long polling, that helps sustain a connection for a relatively long time while using a method provided by REST. This section introduces realization cases using long polling, WebSocket, MQTT, and Webhook.

Example of realizing INF using long polling

Originally, access using HTTP REST is expected to immediately send a response once it receives a request. "Long polling" is a technique for sustaining a connection between server and client by delaying a response for the period delayed. Given this, the server can immediately send a message if any information is generated. This is a highly versatile method that can be implemented by using normal HTTP access methods. However, doing this tends to have more simultaneous connections with a server. Therefore, it is necessary to take measures and handle reconnection after disconnection.

Based on the method of obtaining property values using the GET method, which will be explained in detail in Chapter 6, it is assumed that the timing of receiving the INF is obtained by a long polling. To distinguish this request from a normal GET, the client requests a long polling to the server side by using other methods for INF, e.g. POST to get it. The server side should be implemented so that no response is returned until the value is updated.

■ Request

```
POST /elapi/v1/devices/<device id>/properties/<Property resource name> HTTP/1.1
```

■ Response (not to respond until updated. The contents are the same as GET cases)

```
{
    <property resource name>:<data>
}
```

or

■ In case where the response is an object

```
{
    <property resource name>: {
        <element name>: <data> ,
        <element name>: <data> ,
        ...
    }
}
```

Furthermore, it is desirable to allow the server to receive INF information from the device on reconnection, even if the server receives INF information from the device before reconnecting after the connection is broken for some reason. This can be achieved by adding an "`If-Modified-Since`" header to the request header, so that the server responds immediately if there have been any changes since that time. This part is similar to the cached implementation described at the beginning of this section in that the value must be recorded on the server side.

However, it is still superior to the simple cached implementation from two perspectives: (1) if assuming INF obtainment using long polling, the time from the disconnection to reconnection would be insignificant, so there is less need to secure a longer time to save history; and (2) there is no need to save property values not accessed with POST.

Example of realizing INF using WebSocket

If using a connection-based method including WebSocket, it is easier and more convenient to use Pub/Sub as connection model. Pub/Sub uses a character string called "topic" for communication. By using resource names that have been explained as the topic, extensions to INF can be implemented without significantly changing the basic concept of this Web API.

In cases where the Pub/Sub model is supported at the protocol level (e.g. MQTT and WAMP), a server called broker or router is often used that specialize in message delivery alone, but in Pub/Sub implemented on WebSockets, which is described in this section, the server also acts as a broker, and communication takes place between the server and the client, taking care of the continuity of REST.

Example of requests and responses (adopted "`echonet`" character string as a sub-protocol)

■ Request

```
GET /websocket  HTTP/1.1
Host: example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: echonet
Sec-WebSocket-Version: 13
```

■ Response

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: echonet
```

■(Client) subscription

```
{
    "method": "subscribe",
    "path": "/elapi/v1/devices/<device id>/properties/operationStatus"
}
```

■(Server) subscription Ack

```
{
    "method": "subscribeAck",
    "path": "/elapi/v1/devices/<device id>/properties/operationStatus"
}
```

■(Server) publication

```
{
    "method": "publish",
    "path": "/elapi/v1/devices/<device id>/properties/operationStatus",
    "value": false
}
```

■(Client) unsubscription

```
{
    "method": "unsubscribe",
    "path": "/elapi/v1/devices/<device id>/properties/operationStatus"
}
```

■(Server) unsubscription Ack

```
{
    "method": "unsubscribeAck",
    "path": "/elapi/v1/devices/<device id>/properties/operationStatus"
}
```

Example of realizing INF using MQTT

MQTT is a lightweight protocol that implements the Pub/Sub model on top of TCP and has several features that are suitable for IoT. We will not discuss the details here, but there is an important difference from the methods explained earlier in this document. That is, the MQTT network configuration must have a broker in charge of message delivery other than the client that generates/consumes messages. In this regard, having a separate broker is necessary to use the MQTT to receive INF (to be explained in this section). Although both the server (explained earlier in this document) and the client using the API are MQTT clients in terms of the MQTT broker, the server is a MQTT client (publisher) that only performs "publishing", while the client is a MQTT client (subscriber) that only performs "subscribing".

The MQTT is a binary protocol directly implemented on the TCP not using HTTP. Therefore, it is not, strictly speaking, a Web API. If using MQTT from the Web client (e.g. browser), it is necessary to use a broker that can perform "MQTT over WebSocket" such as "AWS IoT" and "Mosquitto".

The concept is very simple: the Pub/Sub used in the WebSocket explained in the preceding section is simply implemented. However, its sequence varies according to QoS, the parameters that control the percentage of guaranteed arrival of messages, and the number of times messages are resent according to the parameter. The closest case to the request/response example represented in the INF realization example by WebSocket is a case with QoS=1. The following shows the sequence if realizing a similar thing with MQTT (as a binary protocol, the MQTT does not have a technique to simply describe messages as a request line).

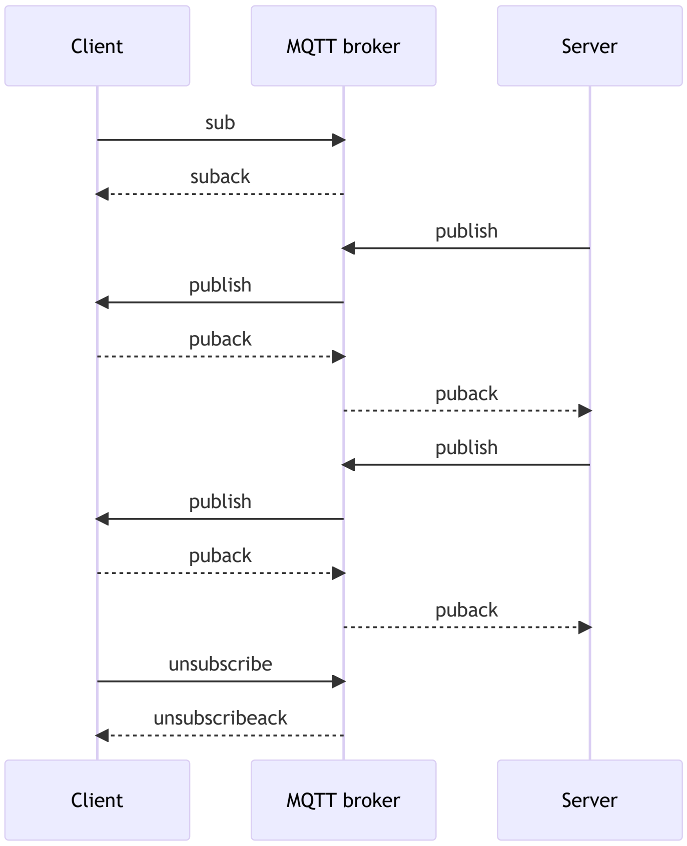


Figure 5-1 Example of realizing INF using MQTT

As shown above, it is almost the same as WebSocket, unless the broker is used as an intermediary. The topic for publish/subscribe should also be the path part shown in the WebSocket request and response example (the resource string itself used in REST), and can be implemented with a value as payload if appropriate.

Example of realizing INF using Webhook

Webhook is a mechanism in which the server notifies the URL specified in advance by the client using the POST method. First, check to see if the server supports Webhook. The response is information about the notification methods supported by the server. The following examples are the cases where the server supports Webhook, but the notification settings remain in the default state.

Request: check to see if the server supports Webhook

```
GET /elapi/v1/notifications HTTP/1.1
```

Response: the server supports Webhook, but "subscription" has not been configured

```
{
    "webhook": {
        "subscriptions": []
    }
}
```

To set up notifications, specify "`subscribe`" by "`method`", the property to be notified by "`path`", and the URL to be notified by "`callBackUrl`". If authentication of the notification recipient is required, this information should also be described. The following examples are for the cases where setting "`apiKey`" of the recipient. It is possible to issue other requests to set different "`callBackUrl`", "`apiKey`", and the like for each "`path`". If the property specified by "`path`" has already been subscribed, "`callBackUrl`" and "`apiKey`" should be overridden.

Request (setting property to be subscribed)

```
POST /elapi/v1/notifications HTTP/1.1

{
    "webhook": {
        "method": "subscribe",
        "path": "https://www.example.com/elapi/v1/devices/0123/properties/operationStatus",
        "callBackUrl": "https://sh-center.org/postreceive",
        "apiKey": {
            "key": "X-Webhook-key",
            "value": "0123ABC"
        }
    }
}
```

To cancel a notification by specifying a property, designate "unsubscribe" by "method".

Request (unsubscribing)

```
POST /elapi/v1/notifications HTTP/1.1
```

```
{
    "webhook": {
        "method": "unsubscribe",
        "path": "https://www.example.com/elapi/v1/devices/0123/properties/operationStatus"
    }
}
```

The following shows an example of obtaining notification details if Webhook notification is configured. The properties "`operationStatus`" and "`operationMode`" are notified to https://sh-center.org/postreceive, and it is revealed that the property "`faultStatus`" is a setting for sending a notification to https://sh-center.org/faultStatus.

Request (confirmation of the properties requested notification)

```
GET /elapi/v1/notifications HTTP/1.1
```

Response

```
{
    "webhook": {
        "subscriptions": [
            {
                "path":
"https://www.example.com/elapi/v1/devices/0123/properties/operationStatus",
                "callBackUrl": "https://sh-center.org/postreceive",
                "apiKey": {
                    "key": "X-Webhook-key",
                    "value": "0123ABC"
                }
            },
            {
                "path":
"https://www.example.com/elapi/v1/devices/0123/properties/operationMode",
                "callBackUrl": "https://sh-center.org/postreceive",
                "apiKey": {
                    "key": "X-Webhook-key",
                    "value": "0123ABC"
                }
            },
            {
                "path":
"https://www.example.com/elapi/v1/devices/0123/properties/faultStatus",
                "callBackUrl": "https://sh-center.org/faultStatus",
                "apiKey": {
                    "key": "X-Webhook-key",
                    "value": "456XYZ"
                }
            }
        ]
    }
}
```

The expected data to be notified are as follows:

```
POST https://sh-center.org/postreceive HTTP/1.1
X-Webhook-key: 0123ABC

{
    "path": "/elapi/v1/devices/0123/properties/operationStatus",
    "body": {
        "operationStatus": true
    }
}
```

To subscribe or unsubscribe multiple properties at once, use the bulks described in Chapter 7.

Next, another realization example is presented.
Assuming that registration/deregistration of URLs to be notified are done by some means (e.g., the procedure described in the realization example above), the method of notifying INF by Webhook is described below.

The HTTP method, for example, uses the POST method. It also notifies that the request is a Webhook and the target resource name. One way to notify resource names, for example, is to enter the resource name in the header field or body of the request. If a resource name is set to the header field, the header name is, for example, "`X-Elapi-notification`". The resource name described in the header field or body should be identifiable by the client, e.g. by using the resource name specified by the client when registering the target property/URL. In the following cases, as an example of describing a resource name in the body, a resource URL is described for the resource key. However, keys such as device ID and property may also be established.

In case that a client receives INF from multiple servers, there may be cases where the resource names (device IDs) of multiple servers overlap. Therefore, the client should: (1) specify different URLs for each server if registering target property and URLs, or (2) be able to uniquely identify resources by writing an identifier that identifies the server in the header or body of the HTTP request of the INF. The identifier that identifies the server is, for example, the host name or the contents of the HOST header in the target property/URL registration. In the following example, it is assumed that the client specifies a different URL for each server, and no identifier identifying the server is entered in the header and body of the INF HTTP request.

Example of INF by Webhook (if resource names are written on the header field)

■ Request

```
POST <arbitrary URL registered by client> HTTP/1.1

X-Elapi-notification: /elapi/v1/devices/<device id>/properties/<Property resource name>
Content-Type: application/json

{
    <property resource name>: <data>
}
```

■ Response

```
Similar to the request
```

Example of INF by Webhook (if resource names are entered on the body)

■ Request

```
POST <arbitrary URL registered by client> HTTP/1.1
Content-Type: application/json

{
    "events": [
        {
            "resource": <resource URL>,
            "value": <data>
        }
    ]
}
```

■ Response

```
Similar to the request
```

Example of general lighting (if resource names are written in the header field)

■ Request

```
POST <arbitrary URL registered by client> HTTP/1.1
X-Elapi-notification: /elapi/v1/devices/<device id>/properties/operationStatus
Content-Type: application/json

{
    "operationStatus": true
}
```

■ Response

```
Similar to the request
```

Example of general lighting (if resource names are entered on the body)

■ Request

```
POST <arbitrary URL registered by client> HTTP/1.1
Content-Type: application/json

{
    "events": [
        {
            "resource": "/elapi/v1/devices/<device id>/properties/operationStatus",
            "value": true
        }
    ]
}
```

■ Response

```
    Similar to the request
```

5.11. Authentication/authorization (introducing cases)

The number of cases requiring authentication would be significant, since they include cases with disconnected communications if excessively accessing due to identifying API users and counting the number of calls.

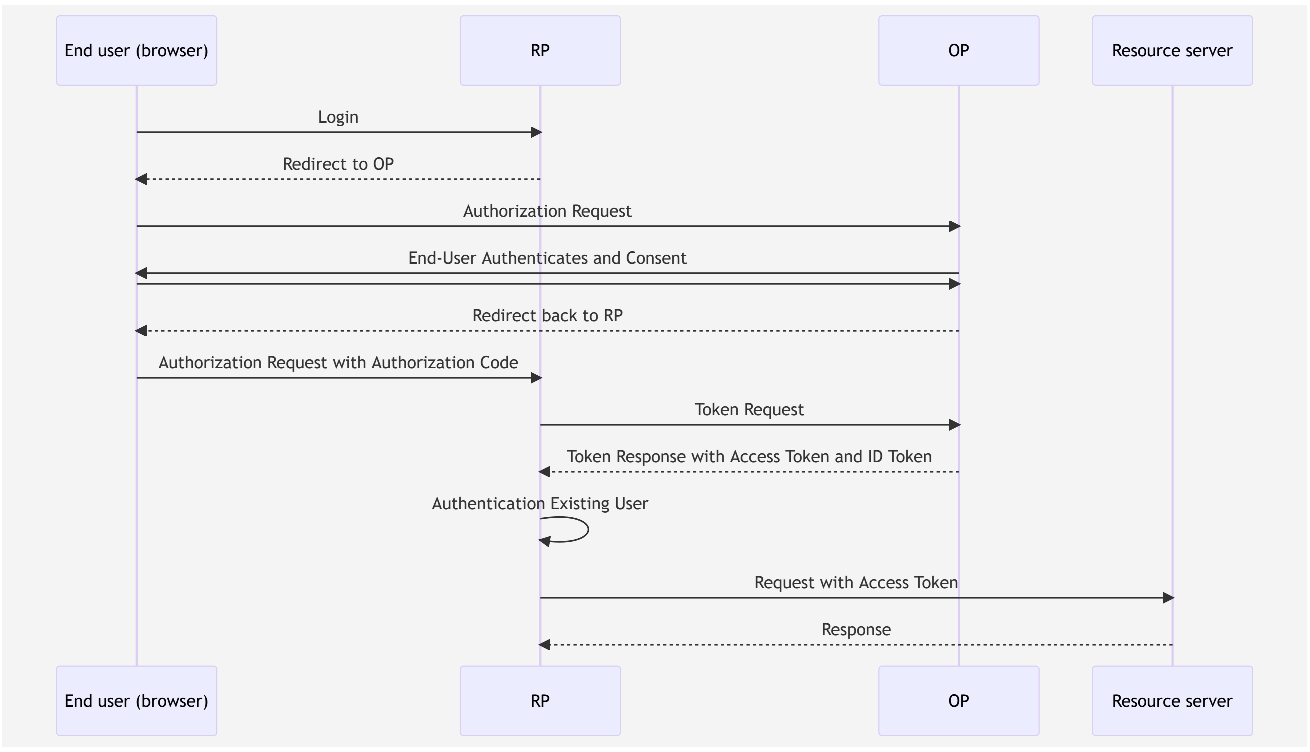
The Web API guidelines introduce a mechanism of authentication/authorization as a reference case. The Web API authentication/authorization methods include token authentication and Oauth2.0 authorization. However, the methods to be adopted vary according to use cases.

The following shows two examples as typical cases using OpenID Connect and OAuth2.0: (1) if using this Web API from a smartphone or other device (Authorization Code); and (2) if using between servers (Client Credentials).

Example of realizing Authorization Code

First, an example of an Authorization Code is presented.

In the figure, RP is the Relying Party and OP is the OpenID Provider (also called as Identity Provider (IdP)). The resource server is a server that implements this Web API.


End user (browser)
RP
OP
Resource server
Login
Redirect to OP
Authorization Request
End-User Authenticates and Consent
Redirect back to RP
Authorization Request with Authorization Code
Token Request
Token Response with Access Token and ID Token
Authentication Existing User
Request with Access Token
Response
End user (browser)
RP
OP
Resource server

If the end user (browser) attempts to log into the RP, a redirect message from the RP to the OP is returned.

Example of responses from RP:

```
HTTP/1.1 302 Found
Location: https://server.example.com/authorize?
    response_type=code
    &scope=openid%20profile%20email
    &client_id=s6BhdRkqt3
    &state=af0ifjsldkj
    &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcallback
```

Mandatory except "`state`". The "`scope`" must include the openid, and if Authorization Code Flow is used, the response_type is "`code`". Then, the end user (browser) sends an authorization request to the OP as described below.

Example of the authorization requests from end users:

```
GET /authorize?
     response_type=code
     &scope=openid%20profile%20email
     &client_id=s6BhdRkqt3
     &state=af0ifjsldkj
     &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcallback HTTP/1.1
Host: server.example.com
```

The OP verifies the authorization requests, and if eligible, attempts to authenticate the end user. Usually, a user interface for authentication is displayed and authentication is performed by login (user name, password, and the like), and consent is obtained from the end user for sending information to the RP after authentication.

After obtaining consent, the OP returns a response including a redirect to the end user (browser).

Example of responses from OP:

```
HTTP/1.1 302 Found
Location: https://client.example.org/callback?
    code=SplxlOBeZQQYbYS6WxSbIA
    &state=af0ifjsldkj
```

The RP called by the end user (browser) via redirection issues a token request to the OP.

Example of requests from RP:

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcallback
```

The OP verifies the contents of the authorization code (code) and returns an access token, ID token, and the like as shown below as long as it is a valid credential.

Example of responses from OP:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
   "access_token": "SlAV32hkKG",
   "token_type": "Bearer",
   "refresh_token": "8xLOxBtZp8",
   "expires_in": 3600,
   "id_token": "eyJhbGciO……"
}
```

The RP calls the Web API of the resource server, accompanied an access token in the header.

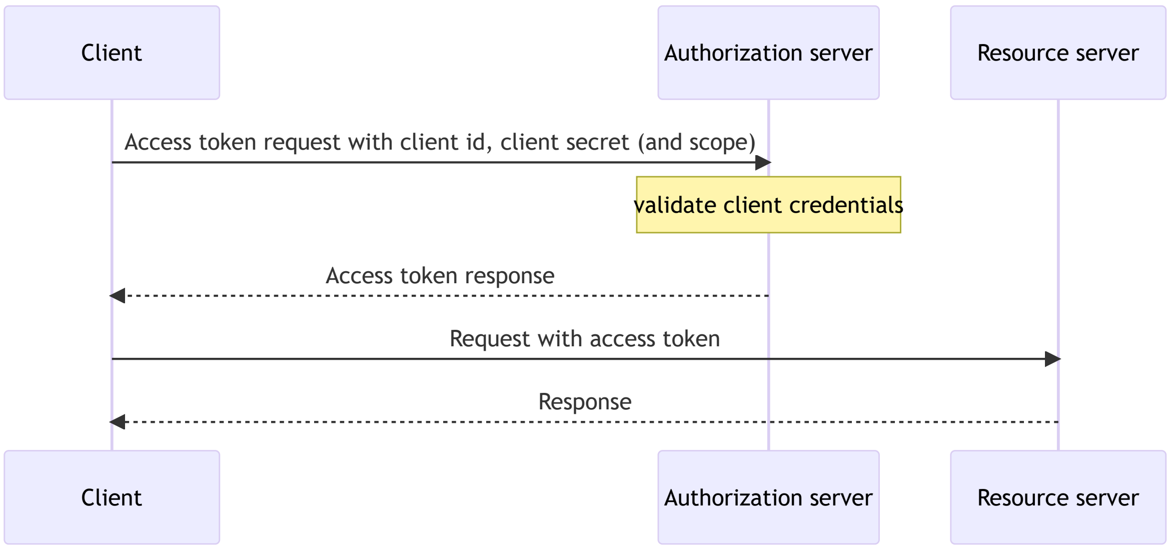
Example of requests of RP:

```
GET /elapi/v1/devices HTTP/1.1
Host: api.xxx.com
Authorization: Bearer  SlAV32hkKG
```

Example of realizing Client Credentials

Next, the Client Credentials case is presented.

In this case, the client usually targets a daemon service or website.


Client
Authorization server
Resource server
Access token request with client id, client secret (and scope)
validate client credentials
Access token response
Request with access token
Response
Client
Authorization server
Resource server

As an authorization server endpoint, for example, "`/oauth2/token`" is called using the POST method protected by TLS. The request header contains attributes with the values of client_id and client_secret encoded in "base64" (HTTP Basic authentication: RFC 7617).

Example of requests from the client:

```
POST /oauth2/token HTTP/1.1
Host: www.auth-server.com
Accept: application/json
Cache-Control: no-cache
Authorization: Basic eW91clfaWQ6eW91cl9jbG......
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials&scope=xxxx
```

"`eW91clfaWQ6eW91cl9jbG......`" (second half is omitted) is a base64(<client_id>:<client_secret>) value. In the request body, "`client_credentials`" is specified in "`grant_type`". Scope "`xxxx`" is the access level required to obtain an access token which can be omitted.

The authorization server verifies client_id and client_secret, and returns an access token ("`eyJraWQiOiJ......`": omitted the latter half), as long as it is a valid credential.

Example of authorization server responses:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
```

```
{
    "access_token": "eyJraWQiOiJ……",
    "token_type": "Bearer",
    "expires_in": 3600
}
```

The client calls Web API of the resource server, accompanied by an access token in the header.

Example of requests from the client:

```
GET /elapi/v1/devices HTTP/1.1
Host: api.xxx.com
Authorization: Bearer eyJraWQiOiJ……
```

The resource server returns the requested data after succeeding verification of the access token. The access token can be verified by querying "introspection" endpoint of the authorization server or by using a public key cached by the authorization server beforehand.

6. Guidelines for mapping ECHONET Lite specifications

This chapter explains the guidelines for mapping ECHONET Lite frames and specifications of device objects.

6.1. Mapping ECHONET Lite frames

ECHONET Lite is a lower layer independent communication protocol. In general, UDP (mandatory) and TCP (optional) are used for the transport layer if using IPv4 or IPv6, and ECHONET Lite frames (Table 6-1) are carried in the payload for communication.

Table 6-1 Response of ECHONET Lite frames

Field	Description	Response with Web API
`EHD`	ECHONET Lite header	Not supported
`TID`	Transaction ID	Not supported
`SEOJ`	Source ECHONET object	Not supported
`DEOJ`	Destination ECHONET object	Identify with device object (instance) ID
`ESV`	ECHONET Lite service	To be discussed later
`OPC`	Target property counters (OPC)	Not supported
`EPC`	ECHONET property	To be discussed later
`PDC`	Property data counter	Not supported
`EDT`	ECHONET property value data	To be discussed later

Table 6-1 describes the guidelines to be followed in this Web API for each of the fields that make up ECHONET Lite frames. Conversion to the Web API is mainly achieved by mapping "`DEOJ`" (recipient device object), "`ESV`" (service type), "`EPC`" (property), and "`EDT`" (property value data) to targets. "`EHD`", "`TID`", and "`PDC`" are matters that are only relevant between in-home controllers and devices, not to be disclosed to the outside from the server. "`TID`" is used to manage communications between the controller and the device, and could be treated as a part of session management between the client and the server. However, there may be cases where the server side performs various forms of communication, including retransmissions, to the controller and devices in response to requests from the client, and as this depends on the server implementation, "`TID`" is outside the scope of mapping to this Web API.

6.2. Mapping DEOJ

Although there are two types of ECHONET Lite objects (node profile objects and device objects), this Web API guideline only handles the device objects. This is because client-side applications that operate through Web API normally and mainly operate device objects. There are some cases that require operations of node profile objects such as checking communication units; however, they are outside the scope of this version’s guidelines. Controllers are treated as a type of devices.

The following shows some examples of descriptions for designating multiple devices to be used for obtaining lists. For the method used to designate multiple devices to be applied to the batch control, refer to "`bulks`" and "`groups`" in Chapter 7.

Designating all controllers:

In case that designating all controllers, following the method used if designating all specific models described later as query parameters, the type is equal to controller (`type=controller`). Or, if a server supports a service type called a "`controller`", "`/elapi/v1/<service designation (can be omitted)>/controllers`" can be designated.

Designating all devices:

If designating all devices (including all controllers). "`/elapi/v1/<service designation (can be omitted)>/devices`" can be designated.

Designating all specific models:

If designating all specific models, it is acceptable to designate the model type such as "`/elapi/v1/<service designation (can be omitted)>/devices?type=homeAirConditioner`".

Handling where the instance code is 0x00:

ECHONET Lite can designate all instances of the device objects by defining the instance code as 0x00 in relation to the specific device objects on a specific node. However, there is no mapping to the Web API.

6.3. Mapping ESV

The ESV under ECHONET Lite is defined as shown in Table 6-2.

Table 6-2 Response with ESV

ESV	Services descriptions	Symbol	Response with Web API
`0x60`	Property value write request (response not required)	SetI	PUT method (request)

ESV	Services descriptions	Symbol	Response with Web API
`0x61`	Property values write request (response required)	SetC	PUT method (request)
`0x62`	Property values read request	Get	GET method (request)
`0x63`	Property value notification request	INF_REQ	To be discussed later
`0x6E`	Property value write/read request	SetGet	PUT method (request)
`0x71`	Property values write response	Set_Res	PUT method (response)
`0x72`	Property values read response	Get_Res	GET method (response)
`0x73`	Property values notification	INF	To be discussed later
`0x74`	Property value notification (response required)	INFC	To be discussed later
`0x7A`	Property value notification response	INFC_Res	To be discussed later
`0x7E`	Property value write/read response	SetGet_Res	PUT method (response)
`0x50`	Property value write response failed	SetI_SNA	PUT method (response)
`0x51`	Property value write response failed	SetC_SNA	PUT method (response)
`0x52`	Property value read response failed	Get_SNA	GET method (response)
`0x53`	Property value notification response failed	INF_SNA	To be discussed later
`0x5E`	Property value write/read response failed	SetGet_SNA	PUT method (response)

Types of ESV can be classified roughly into request, response, and response failed. In case that mapping Set/Get operations to the HTTP method, they can be corresponded to request and response/response failed (response successful/failed).

The Set-related ESV corresponds to update and control values, so it should be mapped to the PUT method. The three types (SetI, SetC, and SetGet) are indistinguishable and their property values should be returned in the response using the PUT method request and response. In addition, how to map the Ack return and timeout processes related to the response latency of devices specified in the AIF certification specification is outside the scope of this version of the Guideline.

The Get-related ESV should be mapped to the GET method as it is used to retrieve and read values. The handling of data cached on the server is described later.

With normal Web API, INF-related ESV is a pull-based so that it is difficult to receive real-time notifications. However, it is realized by polling mimicry using the GET method or effectively using a push-based communication model such as WebSocket, MQTT, Web Notification API, or Webhook.

6.4. Mapping EPC and EDT

EPC and EDT are defined by being linked with device object specifications. EPC is defined as resources in URI, and EDT is communicated in request and response bodies. Specifically, properties that can be obtained are mapped to "`/elapi/v1/devices/<device id>/properties/<property resource name>`" based on the device descriptions described in 5.7. Property names are converted to property resource names, and those details are specified in the "Device specification section" (separate document). The properties to be set are handled by the property resource names under "`properties`", if they can be obtained. If they cannot be obtained, they should be handled by separately defining operation names under "`actions`".

The following sections explain this in more detail.

In addition, the prefix sections ("`/elapi/v1`") of the URI indicating the endpoints may be omitted hereafter, to simplify the notation.

6.5. Mapping method and operation of ECHONET Lite device objects

Table 6-3 shows the basic API related to operation of device objects.

Table 6-3 API related to operation of device objects

http method	path	description
GET	/devices/<device id>/properties	Batch acquiring of property values to be obtained
GET	/devices/<device id>/properties/<property resource name>?<query>	Acquiring designated property values
PUT	/devices/<device id>/properties/<property resource name>	Setting designated property values
PATCH	/devices/<device id>/properties	Setting multiple property values to be set
POST	/devices/<device id>/echoCommands	Setting value of designated properties

ECHONET Lite defines two types of objects (Node Profile Objects and Device Objects), but this Web API guideline handles only device objects as described above. The device object super class basically maps individual properties. However, information that becomes apparent due to JSON conversion (e.g. property map) is omitted. For the mapping of super class properties, see "Device specification section" (separate document).

The following indicates the basic policy for the various operation APIs for device objects.

GET /devices/<device id>/properties

By specifying "`/elapi/v1/<service specification (optional)>/devices/<device id>/properties`" and calling it with the GET method, values can be returned for all properties that are subject to GET.

■ Request

```
GET /elapi/v1/devices/<device id>/properties HTTP/1.1
```

■ Response

```
{
    <property resource name>: <property value>,
    <property resource name>: <property value>
    ...
}
```

Example of general lighting

■ Request

```
GET /elapi/v1/devices/<device id>/properties HTTP/1.1
```

■ Response

```
{
    "operationStatus": true,
    "faultStatus": false,
    "brightness": 50,
    "operationMode": "color",
    "rgb": {"r": 20, "g": 255, "b": 0 }
}
```

GET /devices/<device id>/properties/<property resource name>?<query>

By specifying "`/elapi/v1/<service specification (optional)>/devices/<device id>/properties/<property resource name> ? <query>`" and calling it with the GET method, the specified property value can be returned. The "`query`" is normally not mandatory. However, it is used for designating SET data if SET (EL) and GET (EL) "`atomic operations`" are required for specific properties of some devices (such as smart electric meters). The "`keys`" are defined for each property.

■ Request

```
GET /elapi/v1/devices/<device id>/properties/<property resource name>?<query> HTTP/1.1
```

■ Response

```
{
    <property resource name>: <data>
}
```

or

■ In case where the response is an object

```
{
    <property resource name>: {
        <element name>: <data> ,
        <element name>: <data> ,
        ...
    }
}
```

Example of general lighting

■ Request

```
GET /elapi/v1/devices/<device id>/properties/operationMode HTTP/1.1
```

■ Response

```
{
    "operationMode": "color"
}
```

■ Request

```
GET /elapi/v1/devices/<device id>/properties/rgb HTTP/1.1
```

■ In case where the response is an object

```
{
    "rgb": {
        "r": 20,
        "g": 255,
        "b": 0
    }
}
```

■ If a request has a query

```
GET /elapi/v1/devices/<device id>/properties/normalDirectionIntegralElectricEnergyLog1?
day=0 HTTP/1.1
```

■ Response

```
{
    "normalDirectionIntegralElectricEnergyLog1": {
        "day": 0,
        "energy": [
            20,
            34,
            59,
            109
        ]
    }
}
```

PUT /devices/<device id>/properties/<property resource name>

By specifying "`/elapi/v1/<service specification (optional)>/devices/<device id>/properties/<property resource name>`" and calling it with the PUT method, the specified property value can be set. The server obtains the value further obtained through GET operation after ECHONET Lite SET operation for the property corresponding to the property resource of the target device via the controller. Then, the server returns it to the client.

■ Request

```
PUT /elapi/v1/devices/<device id>/properties/<property resource name> HTTP/1.1

{
    <property resource name>: <data>
}
```

■ If a request is an object

```
{
    <property resource name>: {
        <element name>: <data>,
        <element name>: <data>,
        ...
    }
}
```

■ Response

```
Similar to the request
```

Example of general lighting

■ Request

```
PUT /elapi/v1/devices/<device id>/properties/operationMode HTTP/1.1

{
    "operationMode": "color"
}
```

■ Response

```
{
    "operationMode": "color"
}
```

■ If a request is an object

```
PUT /elapi/v1/devices/<device id>/properties/rgb HTTP/1.1

{
    "rgb": {
        "r": 20,
        "g": 255,
        "b": 0
    }
}
```

■ Response

```
{
    "rgb": {
        "r": 20,
        "g": 255,
        "b": 0
    }
}
```

The procedures stated above assume execution of operations based on the device descriptions targeting devices described in the "Device specification section" (separate document). For more details, see the "Device specification section" (separate document).

On the other hand, supplementary functions are introduced (optional, later described in 7.4) to enable EPC operation that is outside the scope of this guideline in cases where no desired property definition exists in the device description stipulated in the "Device specification section" (separate document), and if using vendor-specific manufacturer-dependent codes.

PATCH /devices/<device id>/properties

In case that the server corresponds with RFC 5789 (PATCH Method for HTTP), specified multiple property values can be set by performing "`PATCH`" by specifying "`/elapi/v1/<service specification (can be omitted)>/devices/<device id>/properties`". Other properties not specified remain unchanged.

In case that the server can judge that the request contains a property cannot be set to request (e.g. the property value to be set is out of range or the property resource name does not exist), it returns the 400-499 error as the HTTP response status code. In this case, no request is sent from the server to the controller.

If the fact that the request contains a property that cannot be set to request becomes apparent during processing of controller or device (e.g., the property cannot be updated due to an internal condition on the device side), the server returns the 500-599 error as the HTTP response status code.

Error responses can be returned in the response body (JSON format) as needed (optional).

For properties that could not be set, the response is returned as a pair of error type and message (arbitrary string) along with the property value specified in PATCH.

■ Request

```
PATCH /elapi/v1/devices/<device id>/properties HTTP/1.1

{
    <property resource name>: <property value>,
    <property resource name>: <property value>
    ...
}
```

■ Response

```
{
    <property resource name>: <propertyValue>,  // Properties successfully changed after
PATCH
    <property resource name>: <propertyValue>,
    ... ,
    "errors": [ <error object>, <error object>...  // Properties unchanged after PATCH
(option)
    ]
}
```

■ error object

```
{
    <property resource name>: <property value>,  // Property requested in PATCH
    "type": <error type>,
    "message": <error message>
}
```

Example of general lighting

■ Request (if the value is out of range or contains invalid properties)

```
PATCH /elapi/v1/devices/<device id>/properties HTTP/1.1

{
    "operationMode": "color", // Normal value
    "rgb": {
        "red": 20,
        "green": 300, // Out of range value
        "blue": 0
    }
}
```

■ Response

```
HTTP/1.1 400 Bad Request

{
    "operationMode": "color", // Normal value (Error of 400-499, so the request will not be
sent)
    "errors": [ // Abnormal value (returns the same property value as if PATCH was
requested)
        {
            "rgb": {
                "red": 20,
                "green": 300,
                "blue": 0
            },
            "type": "rangeError",
            "message": "’green’ value is out of range."
        }
    ]
}
```

■ Request (if some properties cannot be processed and a "response failed" response is returned from the device)

```
PATCH /elapi/v1/devices/<device id>/properties HTTP/1.1
{
    "operationMode": "color",
    "rgb": {
        "red": 20,
        "green": 128,
        "blue": 0
    }
}
```

■ Response

```
HTTP/1.1 500 Internal Server Error
{
    "operationMode": "color", // setting succeeded
    "errors": [ // setting failed
        {
            "rgb": { // property values if response is a matter of implementation (this
example has the same property value as if requesting PATCH)
                "red": 20,
                "green": 128,
                "blue": 0
            },
            "type": "deviceError",
            "message": "SetC_SNA"
        }
    ]
}
```

6.6. Action

To call an "`action`", perform the POST method on the URI containing the target action resource name. Actions are used in situations such as device and service operations that are difficult to define as operations performed by defining property resources.

POST /<device id>/actions/<action resource name>

■ Request

```
POST /elapi/v1/devices/<device id>/actions/<action resource name> HTTP/1.1

{
    <input arg1>: <data1>,
    <input arg2>: <data2>,
    ...
}
```

■ Response

```
HTTP/1.1 200 OK

<output data>
```

6.7. Processing errors

The HTTP stipulates status codes for responses as shown in Table 6-4. Considering above, the server must appropriately return status codes and the like.

Table 6-4 HTTP status code

Status Code	Name	Meanings
100-199	Informational	Processing is ongoing
200-299	Successful	Successfully completed
300-399	Redirection	Redirect request
400-499	Client Error	Error caused by client
500-599	Server Error	Server-induced errors (including communication with in-home controllers and devices)

If more detailed error information can be returned, it is allowed to return appropriate status codes, while referring to Table 6-5 as a reference (optional).

Table 6-5 Use cases for HTTP status codes

Status Code	**Name**	**Meanings (example)**
200	OK	The request has succeeded
201	Created	The request has been fulfilled and resulted in a new resource being created
204	No Content	The server has fulfilled the request but does not need to return an entity-body
301	Moved Permanently	Resources are permanently moved
304	Not Modified	Resource not updated
400	Bad Request	Incorrect request or wrong data format
401	Unauthorized	Authorization is required.
404	Not Found	Resources (corresponding path or property) not found
405	Method Not Allowed	Method designated as a resource not allowed
406	Not Acceptable	Does not match "Accept header"
409	Conflict	Resources conflict (e.g. conflicting ID)
415	Unsupported Media Type	Data format is correct, but the server is not responding
500	Internal Server Error	Error(s) occurred at the server side
503	Service Unavailable	The server is temporary halted

In this guideline, ECHONET Lite Web API calls (if necessary) and error responses arising from services can be returned in a response body (JSON format), and if the above-mentioned status codes are 4XX/5XX-related ones (optional).

As shown below, a pair of error type and message (any string) is returned.

```
{
    "type":<error type>,
    "message":<error message>
}
```

Table 6-6 Error response of service level

Property	Type	Required	Description	Example
`type`	string	Yes	Indicates error types There are two cases for error types: a server judges an incident as an error (rangeError/referenceError/typeError/timeoutError) or a device judges an incident as an error (deviceError)	"rangeError"
`message`	string	Yes	Arbitrary string data to describe error details	

Assumed types are as follows:

Table 6-7 Variation of error type

ErrorType	Description	Example
Caused by client	-	-
`rangeError`	If value to be set is outside the scope of the specifications	`number`, `integer`: if values are not between the min. and max. `enum`: if no value exists
`referenceError`	If target device ID or property resource name does not exist	-
`typeError`	If the type of the value to be set does not match the type described in the device description.	-
Caused by server	-	-
`timeoutError`	No response returned from device within a certain time	Communication-related errors
`deviceError`	If the data received from device are values that correspond with errors. If received SNA from a device	If received SetGet_SNA or Set_SNA

Ex.

```
{
    "type": "rangeError",
    "message": "data is out of range"
}
{
    "type": "rangeError",
    "message": "no value matches to the data"
}

{
    "type": "referenceError",
    "message": "device name is wrong"
}
{
    "type": "referenceError",
    "message": "property resource name is wrong"
}
{
    "type": "typeError",
    "message": "data should be boolean"
}

{
    "type": "timeoutError",
    "message": "timeout !"
}

{
    "type": "deviceError",
    "message": "undefined"
}
{
    "type": "deviceError",
    "message": "GET_SNA"
}
```

6.8. Device information

Described in "Device specification section" (separate document).

7. Applied service functions

The specifications introduced so far have mainly focused on deployment models for mapping the basic functions that ECHONET Lite protocol has to Web services. This chapter shows some examples of how to realize more convenient applied services by combining basic functions, adding other information, and processing data. Note that the description of prefix "`/elapi/v1`" in the paths representing resources are omitted hereafter.

7.1. Batch direction for multiple commands (bulks)

Servers generate and register command sets (bulk) listing multiple commands targeting arbitrary resources based on requests from a client, thereby providing a function to direct and execute multiple commands at once (bulk execution) to the client.

Figure 7-1 illustrates a series of actions related to bulk execution.

- To perform bulk execution, (1) the client first requests the server to register a command set (bulk) listing multiple target commands (consists of "`method`", "`path`", and ["`body`"] ("`body`" can be omitted)). The server registers the bulk, and (2) returns "bulk ID", which serves as an identifier for the bulk to the client.
- Next, (3) the client specifies the bulk ID to direct the "`execute`" action and the valid period of bulk execution starts at the server, and (4) the server returns the execution ID to the client.
- Thereafter, the server sends each command that constitutes the bulk to in-home/on-premise controllers (including devices), ((5) and (7)) to obtain execution results.
- Then, (9) the client specifies the execution ID to attempt to obtain a response from the server to execute the bulk (the server responds at (10)). The client performs this operation multiple times, until the bulk execution is completed, as needed.
- Next, (11) after the last command execution is completed, bulk execution at the server is completed. Hereafter, (13) in case that the client attempts to obtain a response to the bulk execution, the "`processStatus`" in the obtained

response is a value indicating the end (14). However, if the client attempts to obtain the same response beyond the valid period of the bulk execution stipulated by the server, an error is returned from the server.

- The valid period of the bulk execution can be freely defined by the server. However, it is generally desirable to set a sufficient period to complete the bulk execution and to allow the client to obtain results using the "`getResults`" action.
- Note that the bulk ID can be reused so that if the user wants to execute the bulk again, the procedure from (3) onward can be repeated (by obtaining a new execution ID).
- Eventually, if the registered bulk is no longer needed, the client deletes the bulk registered by server specifying the bulk ID ((16) (17)).

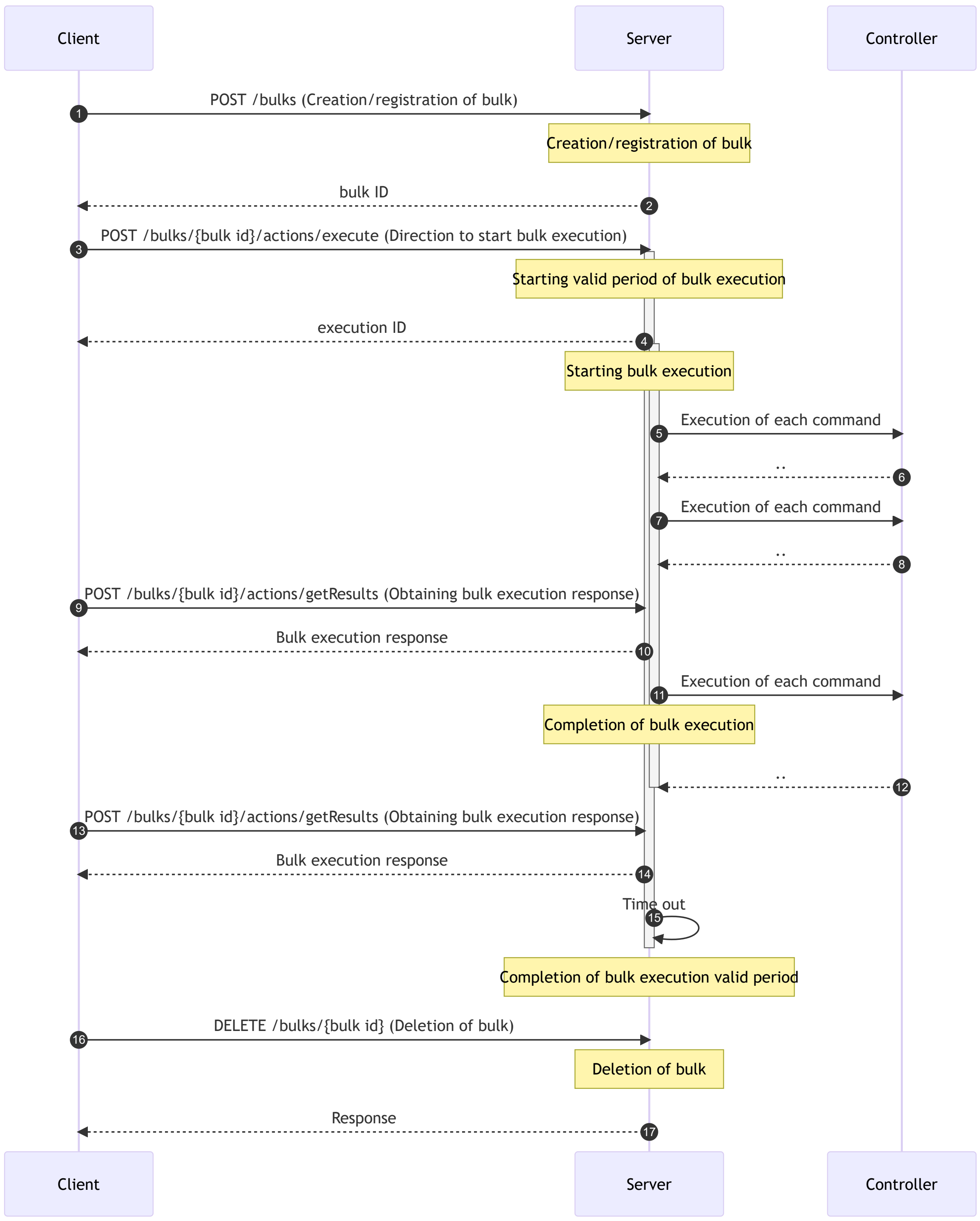

Client
Server
Controller
POST /bulks (Creation/registration of bulk)
Creation/registration of bulk
bulk ID
POST /bulks/{bulk id}/actions/execute (Direction to start bulk execution)
Starting valid period of bulk execution
execution ID
Starting bulk execution
Execution of each command
..
Execution of each command
..
POST /bulks/{bulk id}/actions/getResults (Obtaining bulk execution response)
Bulk execution response
Execution of each command
Completion of bulk execution
..
POST /bulks/{bulk id}/actions/getResults (Obtaining bulk execution response)
Bulk execution response
Time out
Completion of bulk execution valid period
DELETE /bulks/{bulk id} (Deletion of bulk)
Deletion of bulk
Response
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

Figure 7-1 Overview of bulk execution

In case that the server executes a command to in-home/on-premise controllers (including devices), the bulk process mode ("`processMode`") can designate either of two types (Figure 7-2). If concurrent mode is specified at the time of registering the bulk in (1), the commands should be executed in the order specified in the array in the "`requests`" key (without waiting for a response). If "`sequential`" mode is specified, the commands should be executed in the order specified in the array in the "`requests`" key (while waiting for a normal response).

If "`processMode`" is not specified, it is executed in concurrent mode.

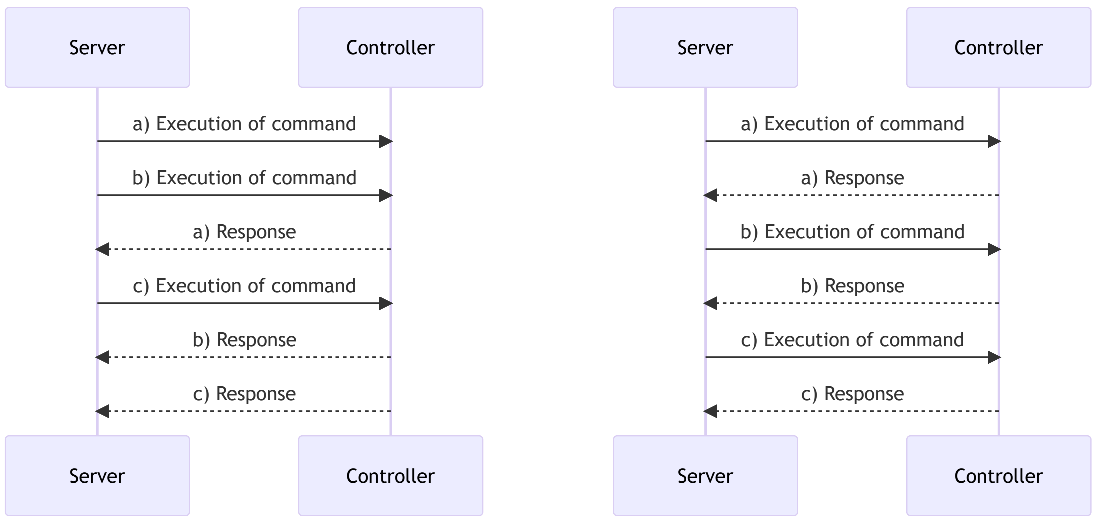


Figure 7-2 Concurrent mode (asynchronous) and sequential mode (synchronous)

The response details of the bulk execution in (10) allows for checking the status of the bulk execution from both individual and overall perspectives.

The individual status can be checked for each command listed in the array in the "`responses`" key (in the order of the array specified by the "`requests`" key), while the execution status of each command can be checked with the "`progress`" key.

The values can be any one of the following: "`unexecuted`" (not performed; default value), "`executing`" (in progress), "`completed`" (successfully completed), "`failed`" (unsuccessfully completed), "`timeout`" (time is expired), or "`aborted`" (suspended).

If "`processMode`" is "`concurrent`", all commands are executed unless the client explicitly instructs an "`abort`" action or the processing time on the server side does not exceed the global timeout value (defined by the server).

Figure 7-3 provides an example of process in the "`concurrent`" mode. The figure illustrates that each command in the bulk containing command a) through command e) is being executed sequentially; command a) is completed, b) is failed, c) is timed out during execution, d) is executing, and e) is in an unexecuted state (the point in time when the client called is d); a position while executing).

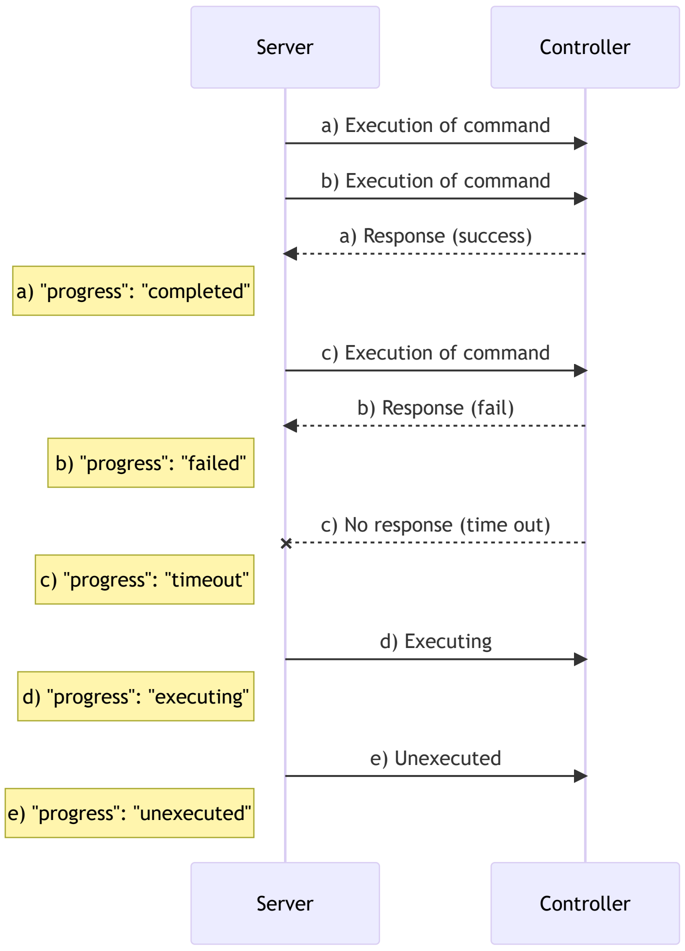


Figure 7-3 Concurrent mode: executing until d)

If "`processMode`" is "`sequential`", all commands are executed unless the server execution of each command fails (failed or timeout), the client explicitly instructs an "`abort`" action or the processing time on the server side does not exceed the global timeout value (defined by the server). In other words, all commands are executed as long as a normal response (`succeeded`) continues.

Figure 7-4 shows an example of process in "`sequential`" mode. The figure illustrates that each bulk command containing command a) through command e) is being executed sequentially; command a) and b) are completed and c) and subsequent commands are in an unexecuted state (the point in time when the client called is b); a position immediately after).

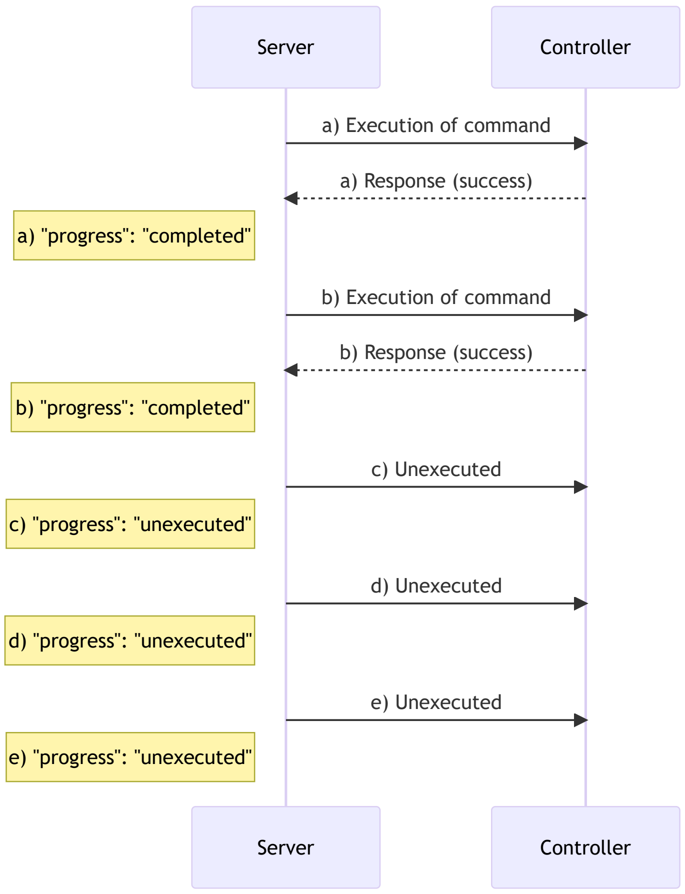


Figure 7-4 Sequential mode: executing until b)

Subsequently, Figure 7-5 indicates that d) and e) have shifted from the unexecuted to the aborted state, since c) has failed. Because c) has failed, no further commands are executed.

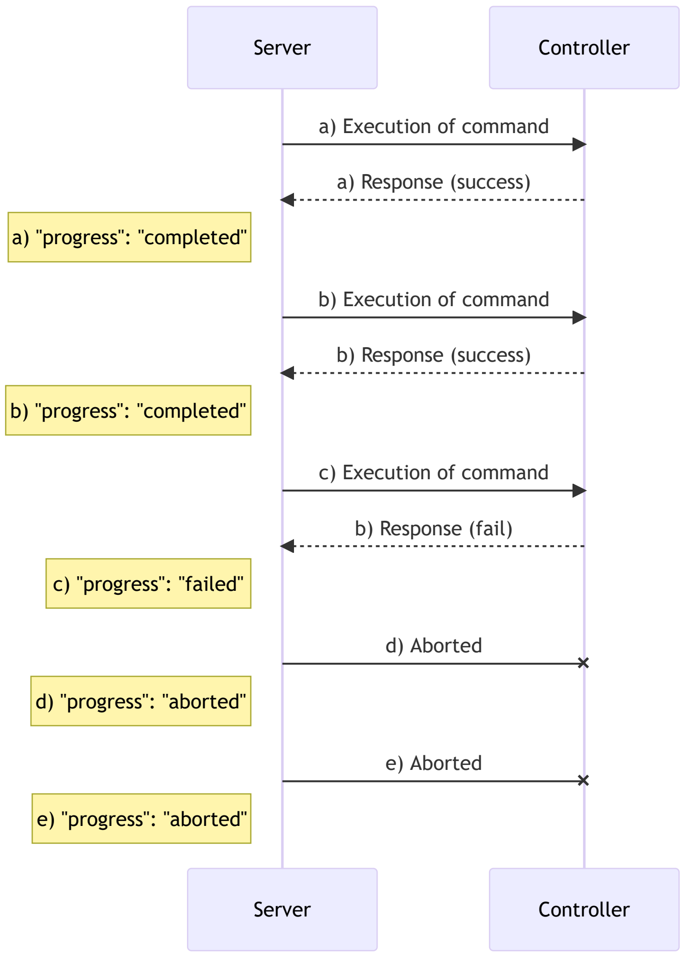


Figure 7-5 Sequential mode: c) stopped after execution

When an abort is sent from the client in any case of "`processMode`", the remaining execution process corresponding to the specified execution ID on the server is aborted, and commands whose "`progress`" is "`executing`" or "`unexecuted`" are moved to "`aborted`".

Since the valid period of the bulk execution lasts forever as long as it is not automatically deleted after a certain period of time on the server (see explanation above), the client can obtain the status if aborted, using the getResults action accompanied by the execution ID.

It is possible to understand the overall operation status by checking the "`progress`" of all commands listed in the responses described in (10). However, if there are a large number of target commands, the status checking process to be performed by the client becomes too complicated. A "`processStatus`" is provided so that a client can easily understand the overall progress status related to bulk execution.

A "`processStatus`" can be any one of the following: "`inProgress`" (execution in progress), "`succeeded`" (all executions are successfully completed), "`failed`" (executions are completed with one or more failures or timeouts), and "`aborted`" (at least one execution is suspended; it is prioritized over "`failed`").

The values of "`processStatus`" according to the overall progress are as follows:

- If all commands are unexecuted or one or more commands are in progress: "`inProgress`"

- In any one of the following cases: if the client aborts by issuing an "`abort`" action during the bulk execution, if it is aborted beyond the timeout value of the entire server system, or if "`failed`" or "`timeout`" occurs in "processMode" with "sequential" mode: "`aborted`"
- If all command executions are completed:
  - If all executions succeeded → "`succeeded`"
  - If one or more executions failed or timed out → "`failed`"

It is assumed that the timeout values are set by the server in a timely manner. The execution timeout value "t_i" (where i is from 1 to total) for each command and the execution timeout value T (bulk execution validity period) for all commands need not necessarily be greater than the sum of the respective execution timeout values ($\sum_{i=1}^{n} t_i$), but should be set to a period with a sufficient margin, as described above.

Table 7-1 lists the APIs related to bulk. The following explains the APIs individually.

Table 7-1 API for batch direction of multiple commands

http method	path	description
POST	/bulks	Creates a bulk
GET	/bulks	Returns a list of bulk IDs
GET	/bulks/<bulk id>	Returns a bulk description
GET	/bulks/<bulk id>/properties	Returns all property resource values of the bulk
GET	/bulks/<bulk id>/properties/<property resource name>	Returns a property resource values of the bulk
PUT	/bulks/<bulk id>/properties/<property resource name>	Sets a property resource values of the bulk
POST	/bulks/<bulk id>/actions/execute	Starts a bulk execution
POST	/bulks/<bulk id>/actions/abort	Aborts a bulk execution
POST	/bulks/<bulk id>/actions/getResults	Returns bulk execution results
DELETE	/bulks/<bulk id>	Deletes a bulk

POST /bulks

Creates and registers a bulk, generates an identifier (bulk ID), and returns it to the client.

The client specifies the common URI part of the target resource (command) with "`base`" (optional), specifies the bulk processing mode ("`processMode`"), enumerates commands (in the form of an array of "`requests`"), and makes a request to the server.

"`processMode`" can be omitted, in which "`concurrent`" is used. Each command consists of sets which are listed in an array in the "`requests`" key of the following: "`method`" (HTTP request method) , "`path`" (relative resource location from "`base`", absolute resource position if "`base`" is not specified), ["`body`" (HTTP request body to be argument, can be omitted)]. If the creation and registration of a bulk on the server is successfully completed, bulk ID is returned from the server to the client as the HTTP status code 201 and HTTP body. In case that the maximum number of bulks that can be registered by the server is exceeded, `{ "type": "rangeError", "message": "You can't create bulks over the registration limit" }`is returned as HTTP status code 400 and HTTP body.

■ Definition of request

```
{
    "descriptions": {
        "ja": <description in Japanese>,
        "en": <description in English>
    },
    "base": <base uri>,
    "processMode": "concurrent" | "sequential",
    "requests": [
        {
            "method": <http method>,
            "path": <path>,
            "body": <value>(optional)
        },
        ...
    ]
}
```

■ Definition of response

```
{
    "id": <bulk id>
}
```

■ Example of request

```
{
    "descriptions": {
        "ja": "帰宅",
        "en": "I'm home"
    },
    "base": "https://xxx.xxx/elapi/v1/",
    "processMode": "concurrent",
    "requests": [
        {
            "method": "GET",
            "path": "devices/0123/properties/operationStatus"
        },
        {
            "method": "PUT",
            "path": "devices/0124/properties/targetTemperature",
            "body": {
                "targetTemperature": 24
            }
        }
    ]
}
```

■ Example of response

```
{
    "id": "00000011"
}
```

Request:

Property	Type	Required	Description
`descriptions`	object	Yes	Description related to the bulk to be registered
`descriptions.ja`	string	Yes	Explanation of the content of the bulk to be registered
`descriptions.en`	string	Yes	The content of a bulk to register in English
`base`	string	No	Common base URI to the resource to be operated (can be omitted)
`processMode`	string	No	"`concurrent`" or "`sequential`". If omitted, the description will be "`concurrent`" mode. The former performs parallel execution on the command set to be registered in requests, while the latter performs sequential execution. If performing parallel execution, each command is executed sequentially in the order it is described. However, each response is executed without waiting. If performing sequential execution, the system waits for a normal response to a command and then executes the next command.
`requests`	array	Yes	List each command as an array
`requests[].method`	string	Yes	HTTP request method
`requests[].path`	string	Yes	Path to the resource to be operated. If "`base`" is specified, the resource is specified with a relative path, and if it is omitted, the resource is specified with an absolute path (URI).
`requests[].body`	object	No	Request body required during operation

Response:

Property	Type	Required	Description
`id`	string	Yes	bulk ID

GET /bulks

Returns a list of bulk IDs. The bulk IDs registered to the server are returned in an array format. If there is no registration, an empty array will be returned.

■ Definition of response

```
{
    "registrationLimit": <maximum number of registered bulks>,
    "bulks": [
        {
            "id": <bulk id>,
            "descriptions": {
                "ja": <description in Japanese>,
                "en": <description in English>
            }
        },
        ...
    ]
}
```

■ Example of response

```
{
    "registrationLimit": 100,
    "bulks": [
        {
            "id": "00000011",
            "descriptions": {
                "ja": "帰宅",
                "en": "I'm home"
            }
        },
        {
            "id": "00000012",
            "descriptions": {
                "ja": "外出",
                "en": "I'm out"
            }
        }
    ]
}
```

Response:

Property	Type	Required	Description
`registrationLimit`	number	No	Maximum number of bulks that can be registered
`bulks`	array	Yes	Enumerates registered bulk IDs etc in an array. Empty (`"bulks": []`) if there is no registration.
`bulks[].id`	string	No	Bulk ID
`bulks[].descriptions`	object	No	Description of registered bulk
`bulks[].descriptions.ja`	string	No	Explanation of the content of the registered bulk

Property	Type	Required	Description
`bulks[].descriptions.en`	string	No	The content of the registered bulk in English

GET /bulks/<bulk id>

Returns a bulk description specified with bulk ID.

■ Example of response

```
{
    "properties": {
        "descriptions": {
            "descriptions": {
                "ja": "bulkの説明",
                "en": "explanation of bulk."
            },
            "writable": true,
            "observable": false,
            "schema": {
                "type": "object",
                "properties": {
                    "ja": {
                        "type": "string"
                    },
                    "en": {
                        "type": "string"
                    }
                }
            }
        },
        "base": {
            "descriptions": {
                "ja": "ベースのURI",
                "en": "base URI"
            },
            "writable": false,
            "observable": false,
            "schema": {
                "type": "string"
            }
        },
        "processMode": {
            "descriptions": {
                "ja": "処理モード",
                "en": "processing mode."
            },
            "writable": false,
            "observable": false,
            "schema": {
                "type": "string",
                "enum": [
                    "concurrent",
                    "sequential"
                ]
            }
        },
        "requests": {
            "descriptions": {
                "ja": "bulkの中の要求命令セット",
                "en": "set of request commands in a bulk."
            },
            "writable": false,
            "observable": false,
            "schema": {
                "type": "array",
                "items": {
                    "type": "object",
                    "properties": {
                        "method": {
                            "type": "string",
                            "enum": [
                                "GET",
                                "PUT",
                                "POST",
                                "PATCH",
                                "DELETE"
                            ]
                        },
                        "path": {
                            "type": "string"
                        },
                        "body": {
                            "oneOf": [
                                {
                                    "type": "string"
                                },
                                {
                                    "type": "number"
                                },
                                {
                                    "type": "object"
                                },
                                {
                                    "type": "boolean"
                                },
                                {
                                    "type": "array",
                                    "items": {
                                        "oneOf": [
                                            {
                                                "type": "string"
                                            },
                                            {
                                                "type": "number"
                                            },
                                            {
                                                "type": "object"
                                            },
                                            {
                                                "type": "boolean"
                                            }
                                        ]
                                    }
                                }
                            ]
                        }
                    }
                }
            }
        }
    },
    "actions": {
        "execute": {
            "descriptions": {
                "ja": "bulkを実行",
                "en": "execute a bulk."
            },
            "schema": {
                "type": "object",
                "properties": {
                    "executionId": {
                        "type": "string"
                    }
                }
            }
        },
        "abort": {
            "descriptions": {
                "ja": "bulk実行中断",
                "en": "abort an execution of a bulk."
            },
            "input": {
                "type": "object",
                "properties": {
                    "executionId": {
                        "type": "string"
                    }
                }
            },
            "schema": {
                "type": "object",
                "properties": {
                    "executionId": {
                        "type": "string"
                    }
                }
            }
        },
        "getResults": {
            "descriptions": {
                "ja": "実行結果の取得",
                "en": "get results."
            },
            "input": {
                "type": "object",
                "properties": {
                    "executionId": {
                        "type": "string"
                    }
                }
            },
            "schema": {
                "type": "object",
                "properties": {
                    "base": {
                        "type": "string"
                    },
                    "total": {
                        "type": "number",
                        "minimum": 0
                    },
                    "processStatus": {
                        "type": "string",
                        "enum": [
                            "succeeded",
                            "failed",
                            "inProgress",
                            "aborted"
                        ]
                    },
                    "responses": {
                        "type": "array",
                        "items": {
                            "type": "object",
                            "properties": {
                                "method": {
                                    "type": "string",
                                    "enum": [
                                        "GET",
                                        "PUT",
                                        "POST",
                                        "PATCH",
                                        "DELETE"
                                    ]
                                },
                                "path": {
                                    "type": "string"
                                },
                                "body": {
                                    "oneOf": [
                                        {
                                            "type": "string"
                                        },
                                        {
                                            "type": "number"
                                        },
                                        {
                                            "type": "object"
                                        },
                                        {
                                            "type": "boolean"
                                        },
                                        {
                                            "type": "array",
                                            "items": {
                                                "oneOf": [
                                                    {
                                                        "type": "string"
                                                    },
                                                    {
                                                        "type": "number"
                                                    },
                                                    {
                                                        "type": "object"
                                                    },
                                                    {
                                                        "type": "boolean"
                                                    }
                                                ]
                                            }
                                        }
                                    ]
                                },
                                "progress": {
                                    "type": "string",
                                    "enum": [
                                        "unexecuted",
                                        "executing",
                                        "completed",
                                        "failed",
                                        "timeout",
                                        "aborted"
                                    ]
                                },
                                "statusCode": {
                                    "type": "number"
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Response:

Property	Type	Required	Description
`properties`	object	Yes	—
`properties.descriptions`	object	Yes	Description of registered bulk
`properties.descriptions.ja`	string	Yes	Explanation of the content of the registered bulk
`properties.descriptions.en`	string	Yes	The content of the registered bulk in English
`properties.base`	string	No	Common base URI to the resource to be operated (if not specified at the time of bulk generation, it should be omitted)

Property	Type	Required	Description
`properties.processMode`	string	Yes	"`concurrent`" or "`sequential`". If not specified at the time of bulk generation, it should be "`concurrent`" mode. The former performs parallel execution on the command set to be registered in requests, while the latter performs sequential execution. If performing parallel execution, each command is executed sequentially in the order it is described. However, each response is executed without waiting. If performing sequential execution, the system waits for a normal response to a command and then executes the next command.
`properties.requests`	array	Yes	List each command as an array
`properties.requests[].method`	string	Yes	HTTP request method
`properties.requests[].path`	string	Yes	Path to the resource to be operated. Paths under "`base`". Relative path if "`base`" is present, and if it is omitted, absolute path (URI)
`properties.requests[].body`	object	No	Request body required during operation. This should be omitted if "`requests[].method`" is GET
`actions`	object	Yes	Description of action
`actions.execute`	object	Yes	Command to start bulk execution
`actions.abort`	object	Yes	Command to abort bulk execution
`actions.getResults`	object	Yes	Obtaining bulk execution results

GET /bulks/<bulk id>/properties

Returns all property resource values of bulk specified by bulk ID.

■ Definition of response

```
{
    "descriptions": {
        "ja": <description in Japanese>,
        "en": <description in English>
    },
    "base": <base uri>,
    "processMode": "concurrent"|"sequential",
    "requests": [
        {
            "method": <http method>,
            "path": <path>,
            "body": <value>(optional)
        },
        ...
    ]
}
```

■ Example of response

```
{
    "descriptions": {
        "ja": "帰宅",
        "en": "I'm home"
    },
    "base": "https://xxx.xxx/elapi/v1/",
    "processMode": "concurrent",
    "requests": [
        {
            "method": "GET",
            "path": "devices/0123/properties/operationStatus"
        },
        {
            "method": "PUT",
            "path": "devices/0124/properties/targetTemperature",
            "body": {
                "targetTemperature": 24
            }
        }
    ]
}
```

Response:

Property	Type	Required	Description
`descriptions`	object	Yes	Description of registered bulk
`descriptions.ja`	string	Yes	Explanation of the content of the registered bulk
`descriptions.en`	string	Yes	The content of the registered bulk in English
`base`	string	No	Common base URI to the resource to be operated (if not specified at the time of bulk generation, it should be omitted)
`processMode`	string	Yes	"`concurrent`" or "`sequential`". If not specified at the time of bulk generation, it should be "`concurrent`" mode. The former performs parallel execution on the command set to be registered in requests, while the latter performs sequential execution. In case that performing parallel execution, each command is executed sequentially in the order it is described. However, each response is executed without waiting. In case that performing sequential execution, the system waits for a normal response to a command and then executes the next command.
`requests`	array	Yes	List each command as an array

Property	Type	Required	Description
`requests[].method`	string	Yes	HTTP request method
`requests[].path`	string	Yes	Path to the resource to be operated. Paths under "`base`". Relative path if "`base`" is present, and if it is omitted, absolute path (URI)
`requests[].body`	object	No	Request body required during operation. This should be omitted if not required, e.g. "`requests[].method`" is GET

GET /bulks/<bulk id>/properties/<property resource name>

Returns a bulk property resource value of bulk specified by bulk ID

■ Example of request

```
GET /bulks/00000012/properties/processMode
```

■ Definition of response

```
{
    <property resource name>: <property value>
}
```

■ Example of response

```
{
    "processMode": "concurrent"
}
```

PUT /bulks/<bulk id>/properties/<property resource name>

Sets a bulk property resource value of bulk specified by bulk ID. The following example shows that the descriptions section is rewritable.

■ Definitions of request and response

```
{
    <property resource name>: <property value>
}
```

■ Examples of request and response

```
{
    "descriptions": {{"ja": "就寝"}, {"en": "I'm going to sleep"}}
}
```

POST /bulks/<bulk id>/actions/execute

Starts bulk execution. No body designation is required at the time of request. Bulk execution adopts the model in which the execution results are obtained asynchronously, assuming it takes time for all requests to complete execution. Specifically, the client directs the start of the bulk execution with "`POST actions/execute`" to obtain the execution ID, and then calls "`POST actions/getResults`" by specifying this ID to obtain the results.

Since execution ID is a temporary ID, no means of retrieval or deletion is provided. The server can be implemented to delete the ID after a certain period of time (the valid period of the execution ID depends on the server implementation). The server accepts the next bulk execution if "`processStatus`" is complete, but returns an error if not.

■ Definition of response

```
{
    "executionId": <execution id>
}
```

■ Example of response

```
{
    "executionId": "0023"
}
```

Response:

Property	Type	Required	Description
`executionId`	string	Yes	Execution ID assigned when executed

POST /bulks/<bulk id>/actions/abort

Aborts a bulk execution. The body of the request specifies the execution ID to be suspended. If the bulk execution is successfully aborted, the same execution ID is returned.

■ Definition of request

```
{
    "executionId": <execution id>
}
```

■ Example of request

```
{
    "executionId": "0023"
}
```

■ Definition of response

```
{
    "executionId": <execution id>
}
```

■ Example of response

```
{
    "executionId": "0023"
}
```

Response:

Property	Type	Required	Description
`executionId`	string	Yes	Execution ID assigned when executed

POST /bulks/<bulk id>/actions/getResults

Returns the response of the bulk execution corresponding to the specified execution ID.

■ Definition of request

```
{
    "executionId": <execution id>
}
```

■ Example of request

```
{
    "executionId": "0023"
}
```

■ Definition of response:

```
{
    "base": <base uri>,
    "total": <total number of the requests>,
    "processStatus": <processing status of whole commands>,
    "responses": [
        {
            "method": <http method>,
            "path": <path>,
            "body": <body data>,
            "progress": <progress state>,
            "status": <status code>
        },
        ...
    ]
}
```

■ Example of response:

```
{
    "base": "https://xxx.xxx/elapi/v1/",
    "total": 3,
    "processStatus": "inProgress",
    "responses": [
        {
            "method": "GET",
            "path": "devices/0123/properties/operationStatus",
            "body": {
                "operationStatus": true
            },
            "progress": "completed",
            "status": 200
        },
        {
            "method": "PUT",
            "path": "devices/0124/properties/targetTemperature",
            "body": {
                "type": "rangeError",
                "message": "..."
            },
            "progress": "failed",
            "status": 400
        },
        {
            "method": "GET",
            "path": "devices/0124/properties/roomTemperature",
            "progress": "unexecuted"
        }
    ]
}
```

Response:

Property	**Type**	**Required**	**Description**
`base`	string	No	Common base URI to the resource to be operated (if "`base`" is not specified at the time of bulk creation, it is omitted)
`total`	number	Yes	Total number of execution commands

Property	Type	Required	Description
`processStatus`	string	Yes	Overall progress: "`inProgress`" (in progress), "`succeeded`" (all executions are successfully completed), "`failed`" (executions are completed with one or more failures or timeouts), and "`aborted`" (at least one execution is suspended; it is prioritized over "`failed`").
`responses`	array	Yes	List responses to each command as an array
`responses[].method`	string	Yes	HTTP request method
`responses[].path`	string	Yes	Path to the resource to be operated. Paths under "`base`". Relative path if "`base`" is present, and if it is omitted, absolute path (URI)
`responses[].body`	object	No	Response (response body) after successful execution of operation. If an operation execution error is issued: the error details are described in the object format "`error`" and "`message`" (optional). If the operation is not executed or is being executed, the
same contents as the request body is returned. (if "`requests[].method`" is "`GET`" and the operation is not executed or is being executed, "`responses[].body`" is omitted.)			
`responses[].progress`	string	Yes	Execution status of operation: The value can be any one of the following: "`unexecuted`" (operation is not executed yet), "`executing`" (in progress), "`completed`" (successfully completed after executing operation), "`failed`" (unsuccessfully completed), "`timeout`" (when timeout occurred), or "`aborted`" (execution aborted).
`responses[].status`	string	No	Response after executing the operation (status code). Omitted if the operation has not been executed or is being executed.

DELETE /bulks/<bulk id>

Deletes a registered bulk. The bulk ID specifies the bulk ID to be deleted. The response body is not included and only the HTTP status code 204 (No Content) is returned. If a bulk ID that has already been deleted or does not exist is specified, 404 (Not Found) will be returned.

7.2. Grouping devices (groups)

The server provides the client with a function that enables the client to classify and organize resources and to execute individual and common commands for a target group of devices (group operation function) by registering multiple devices as a group based on a request from the client.

The server can create groups without a request from the client. For example, a server could register "group" in advance as necessary in "the case where a product consists of multiple device objects" or "the case where multiple products are grouped by the HEMS controller in the house". As a means of distinguishing groups registered by the server from groups created by the client's request, the "`preConfigured`" property is defined..

Figure 7-6 illustrates a series of actions related to group operation.

- For group operation, (1) the client first requests the server to register a device list that lists multiple devices to be grouped (group). The server registers the group, and (2) returns "group ID" that serves as an identifier of the group to the client.
- Next, the client can select one of (3), (5), or (11) to perform the desired group operation.
- In (3), the client can obtain all property resource values of the grouped devices specifying the returned group ID; in (5), the client can obtain values of the specified property resources; and in (11), the client can set values of the specified property resources.
- Eventually, if the registered group is no longer needed, the client deletes the group registered on the server by specifying the group ID ((17) (18)).

By default, when operating property resource operations such as (3), (5), or (11), the operation will be executed if the target devices have the properties and the method is executable. If the target devices do not have the specified property or the method is not supported, an error message is returned with HTTP status code 404 (Not Found) or 405 (Method Not Allowed). Even if the property operation is executed, if the response time from the actual device exceeds the timeout value specified by server, an error message (`"type": "timeoutError"`) is returned with HTTP status code 500 (Internal Server Error).

Depending on the characteristics of the group to be generated, the client may not want to require the return of 400-499 or 500-599 codes or error messages for error cases described above. For example, if a group combining one or more storage battery classes with one or more household solar power generation classes is created to treat the group of devices as a virtual composed device ("Virtual Composed Device"), an operation of a property common to the two classes is expected to be executed on all devices and an operation of a property implemented by only one class is expected not to be executed on the class (devices) that does not have the property. As an optional feature, a mechanism to enable such processing is provided.

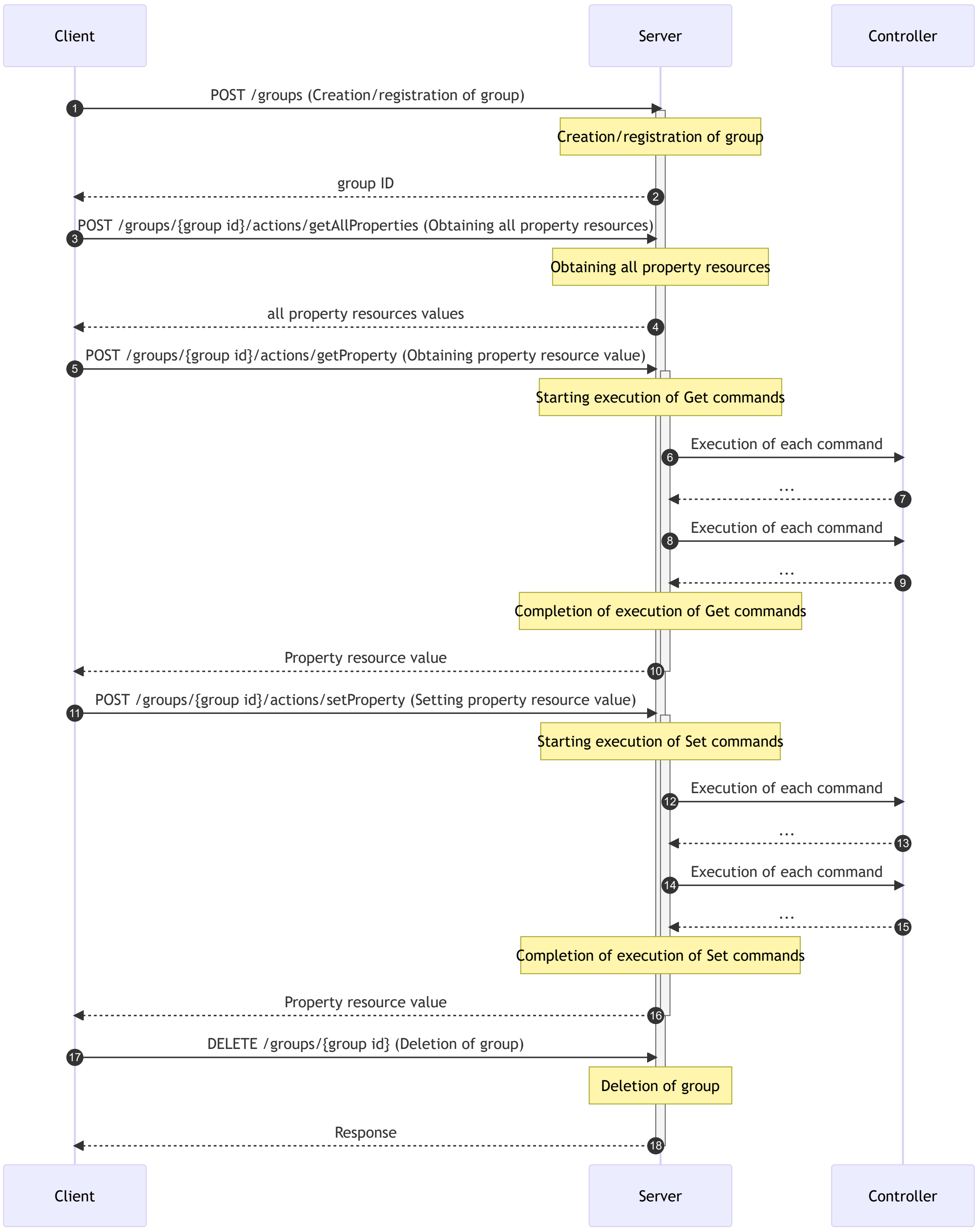

Client
Server
Controller
POST /groups (Creation/registration of group)
Creation/registration of group
group ID
POST /groups/{group id}/actions/getAllProperties (Obtaining all property resources)
Obtaining all property resources
all property resources values
POST /groups/{group id}/actions/getProperty (Obtaining property resource value)
Starting execution of Get commands
Execution of each command
...
Execution of each command
...
Completion of execution of Get commands
Property resource value
POST /groups/{group id}/actions/setProperty (Setting property resource value)
Starting execution of Set commands
Execution of each command
...
Execution of each command
...
Completion of execution of Set commands
Property resource value
DELETE /groups/{group id} (Deletion of group)
Deletion of group
Response
Client
Server
Controller

Figure 7-6 Overview of group execution

If a client wants to specify the devices supporting a class to be operated explicitly, the following designation format can be used (optional).

Example: Operations targeting only devices supporting the general lighting class ("`generalLighting`") in a Virtual Composed Device.

```
```

```
POST /groups/<group id>/actions/getProperty?deviceType=generalLighting
```

Functions to add and delete devices to be grouped after creating a group is provided.

Table 7-2 lists the API related to group. The following explains the APIs individually.

Table 7-2 API related to device grouping

http method	path	description
POST	/groups	Creates a group
GET	/groups	Returns a list of group IDs
GET	/groups /<group id>	Returns a group description
GET	/groups /<group id>/properties	Returns all property resource values of the group
GET	/groups/<group id>/properties/<property resource name>	Returns a property resource value of the group
PUT	/groups/<group id>/properties/<property resource name>	Sets a property resource value of the group
POST	/groups /<group id>/actions/getAllProperties	Returns all property resource values of target devices in the group
POST	/groups /<group id>/actions/getProperty	Returns property values of target devices in the group
POST	/groups /<group id>/actions/setProperty	Sets property values of target devices in the group
DELETE	/groups /<group id>	Deletes a group

POST /groups

Creates and registers a group that containing the target devices to be grouped, generates an identifier (group ID), and returns it to the client. The client requests the server to create a group by specifying a list of device ID of target devices in array format. If the maximum number of groups that can be registered by the server is exceeded, `{ "type": "rangeError", "message": "You can't create group over the registration limit" }`is returned as HTTP body of HTTP status code 400 response.

Thereafter, by default, an operation can be performed on all devices in the group (the same applies if the "`composed`" option is set to "`false`").

If Virtual Composed Devices are supported, specifying "`true`" to "`composed`" allows operations for the properties of the target device group to be performed only on devices having properties and supporting methods. In this case, no response is returned (no error code or error message will be returned) for devices that do not have properties or do not support methods.

If a request is made to a server not supporting "`composed`" by specifying "`true`" to "`composed`", the group ID will not be returned. Instead, an error message (`"type": "typeError"`) is returned with HTTP status code 404 (Not Found).

■ Definition of request

```
{
    "descriptions": {
        "ja": <description in Japanese>,
        "en": <description in English>
    },
    "members": [
        {
            "deviceId": <device id>
        },
        ...
    ],
    "composed": <true/false>
}
```

■ Example of request

```
{
    "descriptions": {"ja": "リビング", "en": "living"},
    "members": [{"deviceId": "0123"}, {"deviceId": "1234"}, {"deviceId": "2345"}],
    "composed": true
}
```

■ Definition of response

```
{
    "id": <group id>
}
```

■ Example of response

```
{
    "id": "00000011"
}
```

Request:

Property	Type	Required	Description
`descriptions`	object	Yes	Description related to the group to be registered
`descriptions.ja`	string	Yes	Explanation of the content of the group to be registered
`descriptions.en`	string	Yes	The content of a group to register in English

Property	Type	Required	Description
`members`	array	Yes	Target device group. List of device IDs in an array format.
`members[].deviceId`	string	Yes	Device ID of the target device
`composed`	boolean	No	Whether or not Virtual Composed Devices are supported. If omitted, it will be considered "false".

Response:

Property	Type	Required	Description
`id`	string	Yes	group ID

GET /groups

Returns a list of group IDs. The group IDs registered with the server are returned in an array format. If no group ID is registered, an empty array will be returned.

■ Definition of response

```
{
    "registrationLimit": <maximum number of registered groups>,
    "groups": [
        {
            "id": <group id>,
            "descriptions": {
                "ja": <description in Japanese>,
                "en": <description in English>
            }
        },
        ...
    ]
}
```

■ Example of response

```
{
    "registrationLimit": 100,
    "groups": [
        {
            "id": "00000011",
            "descriptions": {
                "ja": "リビング",
                "en": "living"
            }
        },
        {
            "id": "00000012",
            "descriptions": {
                "ja": "寝室",
                "en": "bedroom"
            }
        }
    ]
}
```

Response:

Property	Type	Required	Description
`registrationLimit`	number	No	Maximum number of groups that can be registered
`groups`	array	Yes	List of registered group IDs and related information in an array. If no group ID is registered, this is empty (`"groups": []`)
`groups[].id`	string	No	group ID
`groups[].descriptions`	object	No *1	Description related to the registered group
`groups[].descriptions.ja`	string	No *1	Explanation of the content of the registered group
`groups[].descriptions.en`	string	No *1	The content of the registered group in English

*1) Required if groups[].id exists

GET /groups/<group id>

Returns a group description specified with group ID.

■ Example of response

```
{
    "properties": {
        "descriptions": {
            "descriptions": {
                "ja": "groupの説明",
                "en": "explanation of group."
            },
            "writable": true,
            "observable": false,
            "schema": {
                "type": "object",
                "properties": {
                    "ja": {
                        "type": "string"
                    },
                    "en": {
                        "type": "string"
                    }
                }
            }
        },
        "members": {
            "descriptions": {
                "ja": "groupに属する機器のdevice idのリスト",
                "en": "list of device ids in this group."
            },
            "writable": true,
            "observable": false,
            "schema": {
                "type": "array",
                "items": {
                    "type": "object",
                    "properties": {
                        "deviceId": {
                            "type": "string"
                        }
                    }
                }
            }
        },
        "composed": {
            "descriptions": {
                "ja": "仮想混合デバイスの設定",
                "en": "Setting of virtual compound device."
            },
            "writable": false,
            "observable": false,
            "schema": {
                "type": "boolean"
            }
        },
        "preConfigured": {
            "descriptions": {
                "ja": "サーバ事前登録によるGroupか否か",
                "en": "Setting of server preconfigration."
            },
            "writable": false,
            "observable": false,
            "schema": {
                "type": "boolean"
            }
        }
    },
    "actions": {
        "getAllProperties": {
            "descriptions": {
                "ja": "グループに存在する機器の全プロパティを読み出す",
                "en": "read all properties of all devices in this group."
            },
            "schema": {
                "type": "object",
                "properties": {
                    "responses": {
                        "type": "array",
                        "items": {
                            "type": "object",
                            "properties": {
                                "deviceId": {
                                    "type": "string"
                                },
                                "properties": {
                                    "type": "object"
                                }
                            }
                        }
                    }
                }
            }
        },
        "getProperty": {
            "descriptions": {
                "ja": "グループに存在する機器の指定プロパティ値の取得",
                "en": "read the specified property of all devices in this group."
            },
            "input": {
                "type": "object",
                "properties": {
                    "propertyName": {
                        "type": "string"
                    }
                }
            },
            "schema": {
                "type": "object",
                "properties": {
                    "responses": {
                        "type": "array",
                        "items": {
                            "type": "object",
                            "properties": {
                                "deviceId": {
                                    "type": "string"
                                },
                                "body": {
                                    "type": "object"
                                },
                                "statusCode": {
                                    "type": "number"
                                }
                            }
                        }
                    }
                }
            }
        },
        "setProperty": {
            "descriptions": {
                "ja": "グループに存在する機器の指定プロパティ値の設定",
                "en": "write the specified property of all devices in this group."
            },
            "input": {
                "type": "object",
                "properties": {
                    "propertyName": {
                        "type": "string"
                    },
                    "propertyValue": {
                        "oneOf": [
                            {
                                "type": "string"
                            },
                            {
                                "type": "number"
                            },
                            {
                                "type": "object"
                            },
                            {
                                "type": "boolean"
                            },
                            {
                                "type": "array",
                                "items": {
                                    "oneOf": [
                                        {
                                            "type": "string"
                                        },
                                        {
                                            "type": "number"
                                        },
                                        {
                                            "type": "object"
                                        },
                                        {
                                            "type": "boolean"
                                        }
                                    ]
                                }
                            }
                        ]
                    }
                }
            },
            "schema": {
                "type": "object",
                "properties": {
                    "responses": {
                        "type": "array",
                        "items": {
                            "type": "object",
                            "properties": {
                                "deviceId": {
                                    "type": "string"
                                },
                                "body": {
                                    "type": "object"
                                },
                                "statusCode": {
                                    "type": "number"
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Response:

Property	Type	Required	Description

Property	Type	Required	Description
`properties`	object	Yes	—
`properties.descriptions`	object	Yes	Description related to the registered groups
`properties.descriptions.ja`	string	Yes	Explanation of the content of the registered group
`properties.descriptions.en`	string	Yes	The content of the registered group in English
`properties.members`	array	Yes	Target device group
`properties.members[].deviceId`	string	Yes	Device ID of the target device
`properties.composed`	boolean	No	This property indicates whether or not Virtual Composed Devices are supported
`properties.preConfigured`	boolean	No	This property indicates whether the group is registered in advance by the server (true) or the group registered by the client (false). If "true", the group cannot be deleted.
`actions`	object	Yes	—
`actions.getAllProperties`	object	Yes	This action gets all property resources of target device groups
`actions.getProperty`	object	Yes	This action gets property resources of target device groups
`actions.setProperty`	object	Yes	This action sets property resources of target device groups

GET /groups/<group id>/properties

Returns all property resource values of group specified by the group ID

■ Definition of response

```
{
    "descriptions": {
        "ja": <description in Japanese>,
        "en": <description in English>
    },
    "members": [
        {
            "deviceId": <device id>
        },
        ...
    ],
    "composed": <true/false>,
    "preConfigured": <true/false>
}
```

■ Example of response

```
{
    "descriptions": {"ja": "リビング", "en": "living"},
    "members": [{"deviceId": "0123"}, {"deviceId": "1234"}, {"deviceId": "2345"}],
    "composed": true,
    "preConfigured": false
}
```

Response:

Property	Type	Required	Description
`descriptions`	object	Yes	Description related to the registered groups
`descriptions.ja`	string	Yes	Explanation of the content of the registered group
`descriptions.en`	string	Yes	The content of the registered group in English
`members`	array	Yes	List of device IDs of target devices in an array
`members[].deviceId`	string	Yes	Device ID of the target device
`composed`	boolean	No	This property indicates whether Virtual Composed Devices are supported or not
`preConfigured`	boolean	No	This property indicates whether the group is registered in advance by the server or not

GET /groups/<group id>/properties/<property resource name>

Returns a group property resource value of group specified by the group ID

■ Example of request

```
GET /groups/00000013/properties/members
```

■ Definition of response

```
{
    <property resource name>: <property value>
}
```

■ Example of response

```
{
    "members": [{"deviceId": "0123"}, {"deviceId": "1234"}, {"deviceId": "2345"}]
}
```

PUT /groups/<group id>/properties/<property resource name>

Sets a group property resource values of group specified by the group ID. The examples below show that target device groups can be replaced or increased/decreased by updating the device list.

■ Definitions of request and response

```
{
    <property resource name>: <property value>
}
```

■ Examples of request and response

```
{
    "members": [{"deviceId": "1234"}, {"deviceId": "2345"}]
}
```

POST /groups/<group id>/actions/getAllProperties

Returns all property resource values of target devices held by the group specified by group ID. No body designation is required at the time of request. The same results as those obtained by GET method specifying "`<device ID>/properties`" for a device are returned in the value for the "`properties`" key.

■ Definition of response

```
{
    "responses": [
        {
            "deviceId": <device id>,
            "properties": {
                <property resource name 1>: <value 1>,
                <property resource name 2>: <value 2>,
                ...
            }
        },
        ...
    ]
}
```

■ Example of response

```
{
    "responses": [
        {
            "deviceId": "1234",
            "properties": {
                "operationStatus": true, ...
            }
        },
        {
            "deviceId": "2345",
            "properties": {
                "operationStatus": true, ...
            }
        },
    ]
}
```

Response:

Property	Type	Required	Description
`responses`	array	Yes	Response
`responses[].deviceId`	string	Yes	Device ID of the target device
`responses[].properties`	object	Yes	All property resource values of target devices

POST /groups/<group id>/actions/getProperty

Returns property resource values of devices held by the group specified by the group ID. The same contents as those obtained by GET method specifying "`<device ID>/properties/<property resource name>`" for a device are returned in the value for the "`body`" key. At the same time, the HTTP status code is also returned. If an error occurs, the error value and HTTP status code are returned.

If "`composed`" is supported (`true`), response will be returned only if the devices have the same property resource name and support the GET method. In this case, no operation is executed and no response is returned for devices that do not have the property or do not support GET method.

■ Definition of request

```
{
    "propertyName": <property resource name>
}
```

■ Example of request

```
{
    "propertyName": "operationStatus"
}
```

■ Definition of response

```
{
    "responses": [
        {
            "deviceId": <device id>,
            "body": {<property resource name>: <value>
            },
            "status": <status code>
        },
        ...
    ]
}
```

■ Example of response

```
    {
        "responses": [
            {
                "deviceId": "0123",
                "body": {
                    "operationStatus": true
                },
                "status": 200
            },
            {
                "deviceId": "1234",
                "body": {
                    "type": "timeoutError",
                    "message": "the device does not respond"
                },
                "status": 500
            }
        ]
    }
```

Response:

Property	Type	Required	Description
`responses`	array	Yes	Response
`responses[].deviceId`	string	Yes	device ID
`responses[].body`	object	Yes	After successful execution of operation: response (response body). If an execution error is issued: "`type`" (required), "`message`" (required)
`responses[].status`	number	Yes	Response after executing the operation (status code)

POST /groups/<group id>/actions/setProperty

Sets property resource values of the devices held by the group specified by the group ID. The same contents as those obtained by PUT method specifying "`<device ID>/properties/<property resource name>`" for a device are returned in the value for the "`body`" key. At the same time, the HTTP status code is also returned. If an error occurs, the error value and HTTP status code are returned.

If "`composed`" is supported ("`true`"), response will be returned only if the devices have the same property resource name and support the PUT method. In this case, no operation is executed and no response is returned for devices that do not have the property or do not support PUT method.

■ Definition of request

```
{
    "propertyName": <property resource name>,
    "propertyValue": <property value>
}
```

■ Example of request

```
{
    "propertyName": "operationStatus",
    "propertyValue": true
}
```

■ Definition of response

```
{
    "responses": [
        {
            "deviceId": <device id>,
            "body": {<property resource name>: <value>
            },
            "status": <status code>
        },
        ...
    ]
}
```

■ Example of response

```
{
    "responses": [
        {
            "deviceId": "0123",
            "body": {
                "operationStatus": true
            },
            "status": 200
        },
        {
            "deviceId": "1234",
            "body": {
                "type": "timeoutError",
                "message": "the device does not respond"
            },
            "status": 500
        }
    ]
}
```

Response:

Property	Type	Required	Description
`responses`	array	Yes	Response
`responses[].deviceId`	string	Yes	device ID
`responses[].body`	object	Yes	After successful execution of operation: response (response body). If an execution error is issued: "`type`" (required), "`message`" (required)
`responses[].status`	string	Yes	Response after executing the operation (status code)

DELETE /groups/<group id>

Deletes a registered group. The group ID specifies the group ID to be deleted. The response body is not included and only the HTTP status code 204 (No Content) is returned. If a group ID that has already been deleted or does not exist is specified, 404 (Not Found) will be returned.

7.3. Historical data (histories)

The server accumulates the obtained values for each resource of the target devices as historical data along with the obtained time, and provides the required historical data to the client upon request from the client. Historical data consists of pairs of obtained resource values and times. The server itself manages the historical data attributes (history), including the devices that are the target of historical data storage, the timing and duration of historical data recording, and the like. In this version, the server does not provide the client with any means to specify/register devices or resources for storing historical data. The server only provides the client with a means to search historical data sets provided based on targets devices and time periods specified/stored by the server independently. A historical data set is an entire set of historical data that are stored on the server from time to time for a certain history ID.

Figure 7-7 illustrates a series of actions related to history operation.

- To the server, the client can; (1) can request to acquire a list of history IDs, (2) among the listed history IDs obtained from the server, (3) specify a specific history ID to obtain definition information (history description) of the history data linked with the ID (4).
- Then, (5) by specifying "`properties`" following the history ID on the path, the client can obtain all the resource property values of the history linked with the ID from the server (6).
- The client executes to the server, (7)the "`prepareRetrieveData`" action, specifying a history ID (including the period and number of items as needed) to direct the start of preparation for returning a historical data subset desired by client among the historical data sets provided by the server. If the preparation is completed, the server returns to the client the total number of historical data items whose range has been determined and the number of data items per response (page), in addition to the data ID linked with this action (8).

- Thereafter, (9) the client obtains the desired historical data subset from the server by specifying a data ID and page number and executing a "`retrieve`" action (10).
- If the total number of target historical data items is greater than the number per response, the historical data subset is divided. In this case, (11) the subsequent historical data subset can be obtained by specifying the page number after the second page (12); and (13) the final historical data subset can be obtained at the page number of the last page ((total number of items)/(number of items per response)): rounded up to the nearest whole number).
- The valid period of the historical data subset can be freely defined by the server. However, it is generally desirable to set a sufficient period to allow the client to obtain results using the "retrieve" action.

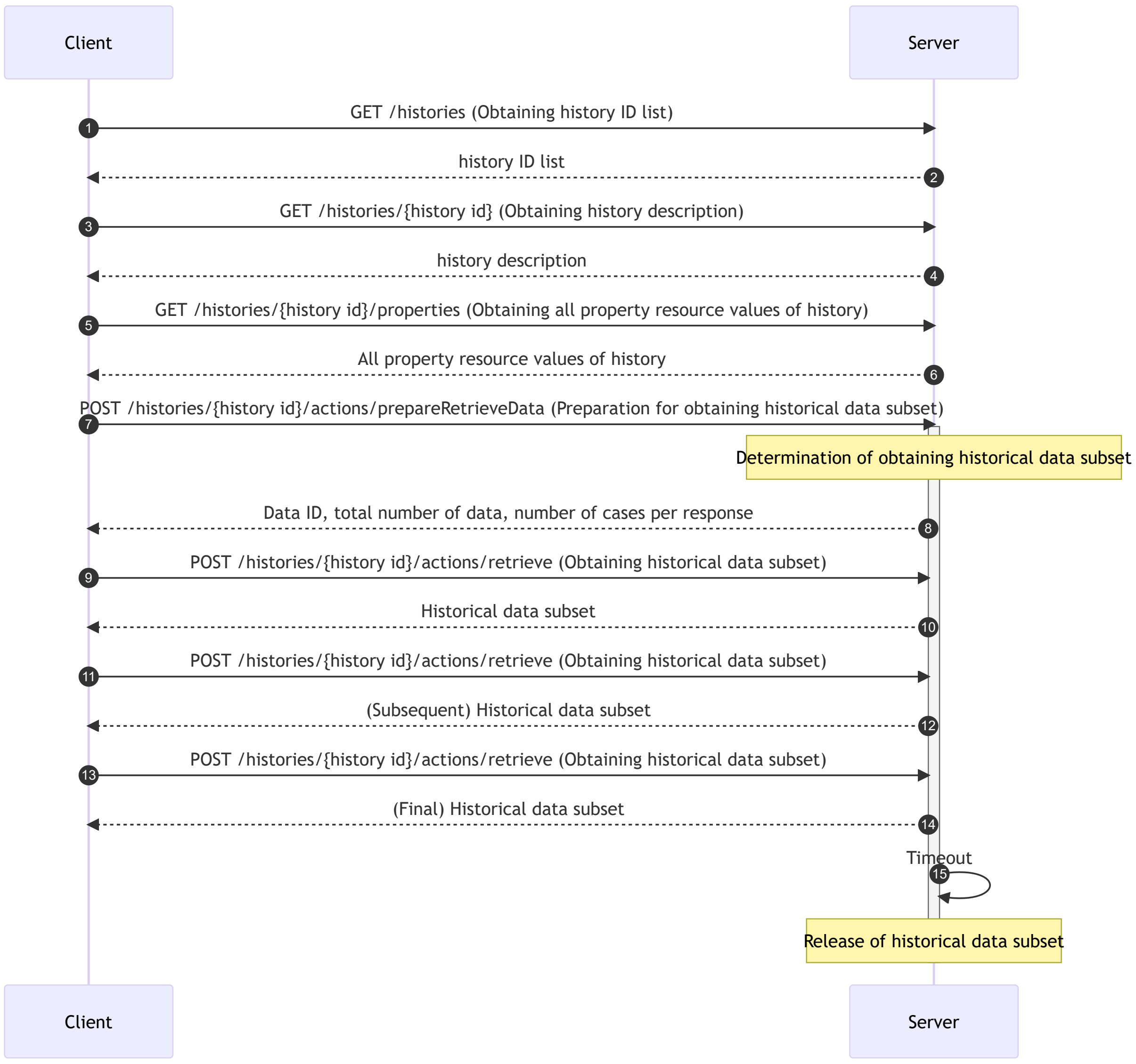

Client
Server
GET /histories (Obtaining history ID list)
history ID list
GET /histories/{history id} (Obtaining history description)
history description
GET /histories/{history id}/properties (Obtaining all property resource values of history)
All property resource values of history
POST /histories/{history id}/actions/prepareRetrieveData (Preparation for obtaining historical data subset)
Determination of obtaining historical data subset
Data ID, total number of data, number of cases per response
POST /histories/{history id}/actions/retrieve (Obtaining historical data subset)
Historical data subset
POST /histories/{history id}/actions/retrieve (Obtaining historical data subset)
(Subsequent) Historical data subset
POST /histories/{history id}/actions/retrieve (Obtaining historical data subset)
(Final) Historical data subset
Timeout
Release of historical data subset
Client
Server

Figure 7-7 Overview of histories execution

Figure 7-8 explains the relationship between the historical data set specified by server and the historical data subset specified by the client. The figure shows that recording began at 8:00 AM and is continuing to save historical data every 30 minutes, and the current time is around 11:45 AM. The historical data set is an aggregate of historical data from 8:00 to 11:30. In contrast, the aggregate of historical data from 9:00 to 10:00 is a historical data subset specified by the client. Depending on the time period specified by the client, the historical data set and the historical data subset may become identical.

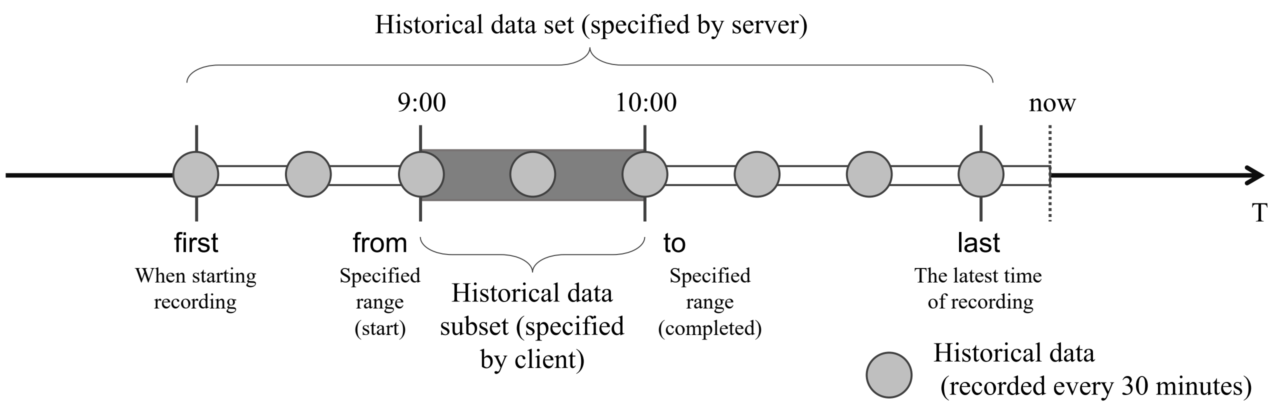

Historical data set (specified by server)
9:00
10:00
now
T
first
When starting recording
from
Specified range (start)
Historical data subset (specified by client)
to
Specified range (completed)
last
The latest time of recording
Historical data (recorded every 30 minutes)

Figure 7-8 Relationship of historical data set and subset

Table 7-3 lists the APIs related to histories. The following explains the APIs individually.

Table 7-3 APIs for historical data

http method	path	description
GET	/histories	Returns a list of history IDs
GET	/histories/<history id>	Returns a history description
GET	/histories/<history id>/properties	Returns all property resource values of the history
GET	/histories/<history id>/properties/<property resource name>	Returns a property resource value of the history
POST	/histories/<history id>/actions/prepareRetrieve	Prepares directions for retrieving history data subset
POST	/histories/<history id>/actions/retrieve	Retrieves historical data subset

*) Note that the setting of property resource values of history is omitted because there are no property resources that can be written in history.

GET /histories

Returns a list of history IDs. The history IDs registered with the server are returned in an array format. If no history ID is registered, an empty array is returned.

■ Definition of response

```
{
    "histories": [
        {
            "id": <history id>,
            "descriptions": {
                "ja": <description in Japanese>,
                "en": <description in English>
            }
        },
        ...
    ]
}
```

■ Example of response

```
{
    "histories": [
        {
            "id": "00000011",
            "descriptions": {
                "ja": "照明000023の動作状態履歴",
                "en": "Operation status history of light 000023"
            }
        },
        {
            "id": "00000012",
            "descriptions": {
                "ja": "太陽光発電（PV）000045の積算発電電力量計測値（kWh）",
                "en": "Cumulative amount of electric energy generated of PV 000045 ([kWh],
30min interval) "
            }
        }
    ]
}
```

Response:

Property	**Type**	**Required**	**Description**
`histories`	array	Yes	List registered history IDs and the like in an array. If there is no registration, empty (`"histories": []`)
`histories[].id`	string	No	history ID
`histories[].descriptions`	object	No *1	Description related to the registered history
`histories[].descriptions.ja`	string	No *1	Explanation of the content of the registered history in Japanese
`histories[].descriptions.en`	string	No *1	Explanation of the content of the registered history in English

*1) Required if `histories[].id` exists

GET /histories/<history id>

Returns a history description specified with history id.

■ Example of response

```
{
    "properties": {
        "descriptions": {
            "descriptions": {
                "ja": "historyの説明",
                "en": "explanation of history."
            },
            "writable": false,
            "observable": false,
            "schema": {
                "type": "object",
                "properties": {
                    "ja": {
                        "type": "string"
                    },
                    "en": {
                        "type": "string"
                    }
                }
            }
        },
        "deviceId": {
            "descriptions": {
                "ja": "履歴データ記録対象機器のdevice ID",
                "en": "device ID of the device for which data are recorded."
            },
            "writable": false,
            "observable": false,
            "schema": {
                "type": "string"
            }
        },
        "deviceType": {
            "descriptions": {
                "ja": "履歴データ記録対象機器のdeviceType",
                "en": "deviceType of the device for which data are recorded."
            },
            "writable": false,
            "observable": false,
            "schema": {
                "type": "string"
            }
        },
        "resourceType": {
            "descriptions": {
                "ja": "リソースタイプ(property, action, event)",
                "en": "resource type (property, action, event)"
            },
            "writable": false,
            "observable": false,
            "schema": {
                "type": "string",
                "enum": [
                    "property",
                    "action",
                    "event"
                ]
            }
        },
        "resourceName": {
            "descriptions": {
                "ja": "リソース名",
                "en": "resource name"
            },
            "writable": false,
            "observable": false,
            "schema": {
                "type": "string"
            }
        },
        "timing": {
            "descriptions": {
                "ja": "履歴データの取得タイミング",
                "en": "timing to record history data"
            },
            "writable": false,
            "observable": false,
            "schema": {
                "type": "object",
                "properties": {
                    "timingType": {
                        "type": "string",
                        "enum": [
                            "onChange",
                            "interval"
                        ]
                    },
                    "intervalValue": {
                        "type": "number"
                    },
                    "intervalUnit": {
                        "type": "string",
                        "enum": [
                            "sec",
                            "min",
                            "hour",
                            "day",
                            "month",
                            "year"
                        ]
                    }
                }
            }
        },
        "first": {
            "descriptions": {
                "ja": "最初の記録時刻",
                "en": "time of the first record"
            },
            "writable": false,
            "observable": false,
            "schema": {
                "type": "string",
                "format": "date-time"
            }
        },
        "last": {
            "descriptions": {
                "ja": "最後の記録時刻",
                "en": "time of the last record"
            },
            "writable": false,
            "observable": false,
            "schema": {
                "type": "string",
                "format": "date-time"
            }
        },
        "total": {
            "descriptions": {
                "ja": "履歴データの総個数",
                "en": "total count of the history data"
            },
            "writable": false,
            "observable": false,
            "schema": {
                "type": "number",
                "minimum": 0,
                "multipleOf": 1
            }
        }
    },
    "actions": {
        "prepareRetrieveData": {
            "descriptions": {
                "ja": "取得用データの準備を指示する",
                "en": "Prepare data to retrieve."
            },
            "input": {
                "type": "object",
                "properties": {
                    "from": {
                        "type": "string",
                        "format": "date-time"
                    },
                    "to": {
                        "type": "string",
                        "format": "date-time"
                    },
                    "count": {
                        "type": "number",
                        "minimum": 1,
                        "multipleOf": 1
                    },
                    "desc": {
                        "type": "boolean"
                    }
                }
            },
            "schema": {
                "type": "object",
                "properties": {
                    "dataId": {
                        "type": "string"
                    },
                    "count": {
                        "type": "number",
                        "minimum": 0,
                        "multipleOf": 1
                    },
                    "countPerPage": {
                        "type": "number",
                        "minimum": 1,
                        "multipleOf": 1
                    }
                }
            }
        },
        "retrieve": {
            "descriptions": {
                "ja": "履歴データを取得する",
                "en": "retrieve histories data."
            },
            "input": {
                "type": "object",
                "properties": {
                    "dataId": {
                        "type": "string"
                    },
                    "page": {
                        "type": "number",
                        "minimum": 1,
                        "multipleOf": 1
                    }
                }
            },
            "schema": {
                "type": "object",
                "properties": {
                    "processStatus": {
                        "type": "string",
                        "enum": [
                            "succeeded",
                            "failed",
                            "inProgress"
                        ]
                    },
                    "resourceType": {
                        "type": "string",
                        "enum": [
                            "property",
                            "action",
                            "event"
                        ]
                    },
                    "resourceName": {
                        "type": "string"
                    },
                    "data": {
                        "type": "array",
                        "items": {
                            "type": "object",
                            "properties": {
                                "time": {
                                    "type": "string",
                                    "format": "date-time"
                                },
                                "value": {
                                    "oneOf": [
                                        {
                                            "type": "string"
                                        },
                                        {
                                            "type": "number"
                                        },
                                        {
                                            "type": "object"
                                        },
                                        {
                                            "type": "boolean"
                                        },
                                        {
                                            "type": "array",
                                            "items": {
                                                "oneOf": [
                                                    {
                                                        "type": "string"
                                                    },
                                                    {
                                                        "type": "number"
                                                    },
                                                    {
                                                        "type": "object"
                                                    },
                                                    {
                                                        "type": "boolean"
                                                    }
                                                ]
                                            }
                                        }
                                    ]
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Response:

Property	Type	Required	Description
`properties`	object	Yes	—
`properties.descriptions`	object	Yes	Description related to the registered history
`properties.descriptions.ja`	string	Yes	Explanation of the content of the registered history in Japanese
`properties.descriptions.en`	string	Yes	Explanation of the content of the registered history in English

Property	Type	Required	Description
`properties.deviceId`	string	Yes	Device ID holding the resource for history
`properties.deviceType`	string	No	Device type of the above
`properties.resourceType`	string	Yes	Type of resource for history. Currently supports property resource "property" only (action resource "action" and event resource "event" are planned to be studied in the future)
`properties.resourceName`	object	Yes	Resource name for history. Resource name of the historical data to be obtained, corresponding to the type of the target resource for history.
`properties.timing`	object	Yes	Timing for obtaining historical data
`properties.timing.timingType`	string	Yes	Type of timing for obtaining historical data. Either "`onChange`" or "`interval`"
`properties.timing.intervalValue`	number	No	Time interval value. Required if the above "`timingType`" is "`interval`"
`properties.timing.intervalUnit`	string	No	Time interval units.Any one of the following: "`sec`", "`min`", "`hour`", "`day`", "`month`" or "`year`". Required if the above "`timingType`" is "`interval`"
`properties.first`	string	Yes	Start time of obtaining historical data. RFC 3339 (ISO 8601) compliant
`properties.last`	string	Yes	Latest time of obtaining historical data. RFC 3339 (ISO 8601) compliant
`properties.total`	number	Yes	Total number of historical data sets
`actions`	object	Yes	—
`actions.prepareRetrieveData`	object	Yes	Preparation for obtaining historical data subset
`actions.retrieve`	object	Yes	Obtaining historical data subset

GET /histories/<history id>/properties

Returns all property resource values of history specified by history ID. Each property includes target device (ID and type), its resource target (type and name), timing to obtain historical data, and period to obtain. Either "`onChange`", which is recorded at irregular intervals, or "`interval`", which is recorded at regular intervals, is used for the timing to obtain historical data, and the data is returned. In the latter "`interval`" case, both the interval value and the time unit are returned. The time at which the historical data started saving is returned as "`first`", while the time when the historical data was last saved is returned as "`last`". The "`last`" and "`total`" may be updated as time elapses or with additional saving of historical data.

■ Definition of response

```
{
    "descriptions": {
        "ja": <description in Japanese>,
        "en": <description in English>
    },
    "deviceId": <device id>,
    "deviceType": <device type>,
    "resourceType": "property | action | event",
    "resourceName": <resource name>,
    "timing": {
        "timingType": "onChange"|"interval",
        "intervalValue": <value>,
        "intervalUnit": <time unit>
    },
    "first": <first time>,
    "last": <last time>,
    "total": <total count>
}
```

■ Example of response 1 (if "timingType" is "`onChange`")

```
{
    "descriptions": {
        "ja": "エアコンabc123の動作状態",
        "en": "operation status of air-conditioner abc123"
    },
    "deviceId": "abc123",
    "deviceType": "homeAirConditioner",
    "resourceType": "property",
    "resourceName": "operationStatus",
    "timing": {
        "timingType": "onChange"
    },
    "first": "2019-04-01T08:00:00+09:00",
    "last": "2019-04-24T22:00:00+09:00",
    "total": 1540
}
```

■ Example of response 2 (if "timingType" is "`interval`")

```
{
    "descriptions": {
        "ja": "エアコンabc123の室内温度",
        "en": "room temperature of the air-conditioner, abc123"
    },
    "deviceId": "abc123",
    "deviceType": "homeAirConditioner",
    "resourceType": "property",
    "resourceName": "roomTemperature",
    "timing": {
        "timingType": "interval",
        "intervalValue": 30,
        "intervalUnit": "min"
    },
    "first": "2019-04-01T08:00:00+09:00",
    "last": "2019-04-24T22:00:00+09:00",
    "total": 1133
}
```

Response:

Property	Type	Required	Description
`descriptions`	object	Yes	Description related to the registered history
`descriptions.ja`	string	Yes	Explanation of the content of the registered history in Japanese
`descriptions.en`	string	Yes	Explanation of the content of the registered history in English
`deviceId`	string	Yes	Device ID holding the resource for history
`deviceType`	string	No	Device type of the above
`resourceType`	string	Yes	Type of resource for history. Currently supports property resource "property" only (action resource "action" and event resource "`event`" are planned to be studied in the future)
`resourceName`	string	Yes	Resource name for history. Resource name of the historical data to be obtained, corresponding to the type of the target resource for history.
`timing`	object	Yes	Timing for obtaining historical data
`timing.timingType`	string	Yes	Type of timing for obtaining historical data. Either "`onChange`" or "`interval`"
`timing.intervalValue`	number	No	Time interval value. Required if the above "`timingType`" is "`interval`"
`timing.intervalUnit`	string	No	Time interval units. Any one of the following: "`sec`", "`min`", "`hour`", "`day`", "`month`" or "`year`". Required if the above "`timingType`" is "`interval`"
`first`	string	Yes	Start time of obtaining historical data. RFC 3339 (ISO 8601) compliant
`last`	string	Yes	Latest time of obtaining historical data. RFC 3339 (ISO 8601) compliant. It may be updated over time.
`total`	number	Yes	Total number of historical data items May be updated by additional saving of historical data.

GET /histories/<history id>/properties/<property resource name>

Returns a history property resource value of history specified by history ID

■ Example of request

```
GET /histories/00000014/properties/deviceId
```

■ Definition of response

```
{
    <property resource name>: <property value>
}
```

■ Example of response

```
{
    "deviceId": "abc123"
}
```

POST /histories/<history id>/actions/prepareRetrieveData

The client specifies the target period (start time "`from`", end time "`to`") of the desired historical data among the historical data sets specified by the history ID, and requests the server to determine the historical data subset to be obtained (the target period is optional). When the server is ready to return the historical data subset, it returns the data ID ("`dataId`"), total number of data items ("`count`"), and number of data items per response ("`countPerPage`") to the client. If no request body is specified, all historical data sets are targeted.

"`from`" and "`to`" are optional and can be specified in the following four ways: In the table, " ✓ " means if specified and "-" means if not specified. In case that "`from`" or "`to`" is specified, either one is included in the historical data recorded at the same time (the same applies to "`first`" and "`last`"). The historical data to be obtained will be returned in order from oldest to newest, regardless of which designation is used. As described above, note that "`last`" may be updated over time, so that it may be different from the value obtained from `GET /histories/<history id>/properties`.

Case	from	to	Target scope
1	✓	✓	Values from "`from`" to "`to`"
2	✓	–	Values from "`from`" to "`last`"
3	–	✓	Values from "`first`" to "`to`"
4	–	–	Values from "`first`" to "`last`" (all historical data sets)

■ Definition of request

```
{
    "from": <time stamp>,
    "to": <time stamp>
}
```

■ Definition of response

```
{
    "dataId": <data id>,
    "count": <count number>,
    "countPerPage": <count per page>
}
```

■ Example of request (designating search start time)

```
{
    "from": "2019-04-01T08:00:00+09:00"
}
```

■ Example of response

```
{
    "dataId": "0023",
    "count": 120,
    "countPerPage": 50
}
```

Request:

Property	Type	Required	Description
`from`	string	No	Search start time. RFC 3339 (ISO 8601) compliant
`to`	string	No	Search end time. RF C3339 (ISO 8601) compliant

Response:

Property	Type	Required	Description
`dataId`	string	Yes	Data ID for obtaining historical data subsets to be obtained. Assigned at execution. This will be automatically deleted after a certain period of time (specified by server) has elapsed.
`count`	number	Yes	Total number of historical data items in the target historical data subset to be obtained. If "`count`" is specified at the time of request, the specified number will become the upper limit.
`countPerPage`	number	Yes	Number of historical data items that can be returned per response. This is the maximum number (in units) of historical data that the server can respond to at one time.

POST /histories/<history id>/actions/retrieve

The client executes acquisition of the historical data subset specified by the data ID.

If the total number of historical data items in the historical data subset exceeds the number of historical data items that can be returned per response, the historical data subset will be divided into multiple responses (pages). To obtain the historical data subset from the client, a page number must be specified in addition to the data ID. However, the page number designation may be omitted for the first page.

In the response, the status of process at the server ("`processStatus`") will be returned. Upon successful completion ("`succeeded`"), the resource type of the historical data (any one of "`property`", "`action`", or "`event`". However, currently only the property resource "property" is supported), the resource name of the historical data, and the historical data ("`data`") will also be returned on a page-by-page basis. Historical data consists of acquisition time ("`time`") and acquisition value ("`value`"). Obtained value can be of various data types depending on the resource type and resource name.

If a page number exceeding the last page is specified, the HTTP status code 404 (Not Found) will be returned.

The "`processStatus`" indicates the server process status of the historical data to be returned by this operation. For the following cases, the HTTP status code 200 will be returned: when a response including historical data is returned, "`succeeded`"; when preparing for server's reasons, "`inProgress`"; when the server process fails/times out (the value is specified by the server), "`failed`". Unlike the bulks case, "`aborted`" is not used because the abort direction function from the client is not supported.

The data ID (and corresponding historical data subset) will be released after a certain period of time elapsed as stipulated by the server.

■ Definition of request

```
{
    "dataId": <data id>,
    "page": <page number>
}
```

■ Example of request

```
{
    "dataId": "0023",
    "page": 2
}
```

■ Definition of response

```
{
    "processStatus": <process status in this history session operation>,
    "resourceType": "property",
    "resourceName": <resource name>,
    "data": [
        {
            "time": <time stamp>,
            "value": <recorded resource value>
        },
        ...
    ]
}
```

■ Example of response (If succeeded)

```
    {
        "processStatus": "succeeded",
        "resourceType": "property",
        "resourceName": "roomTemperature",
        "data": [
            {"time": "2019-04-01T08:xx:xx+09:00", "value": 18},
            {"time": "2019-04-01T08:xy:yy+09:00", "value": 19},
            {"time": "2019-04-01T08:zz:zz+09:00", "value": 21},
            ...
        ]
    }
```

■ Example of response (waiting state during server processing)

```
    {
        "processStatus": "inProgress"
    }
```

■ Example of response (server process failure or timeout)

```
    {
        "processStatus": "failed"
    }
```

Request:

Property	Type	Required	Description
`dataId`	string	Yes	The data ID for obtaining historical data subset to be obtained
`page`	number	No	Page number of the divided historical data subset. Page 1 can be omitted.

Response:

Property	Type	Required	Description

Property	Type	Required	Description
`processStatus`	string	Yes	Overall progress: any one of "`inProgress`" (server in progress), "`succeeded`" (server process is successfully responded), "`failed`" (server processes are executed with failure or timeout)
The following is only used if processStatus is "`succeeded`": "Required" is the standard at the time of use.			
`resourceType`	string	Yes	Resource type of historical data. Property resource "`property`" ("`action`" and "`event`" are planned to be studied in the future)
`resourceName`	string	Yes	Resource name of historical data
`data`	array	Yes	List historical data in the target historical data subset to be obtained as an array. If the total number of data items is 0, empty (`"data": []`)
`data[].time`	string	No	Time to be obtained. RFC 3339 (ISO 8601) compliant
`data[].value`	object	No	Obtained values. Omitted if data is missing ("`time`" will not be omitted)

7.4. Guidelines for additional expansion of the device list

For response details if obtaining a device list, it is recommended not to change/delete the contents defined in "Table 5-2 Detailed response if obtaining device list". However, vendors may add elements independently. The name in such a case should be prefixed with "`vnd`".

The contents defined in "Table 5-2 Detailed response if obtaining device list" should be handled as follows.

deviceType

It can be defined independently. However, it should be prefixed with "`vnd`". Although the devices are already defined in the APPENDIX Detailed Requirements for ECHONET Device objects, if independently defining devices not defined in the "Device specification section" (separate document), "`vnd`" should be used as the prefix.

If defining "`deviceType`", see also 7.5.1.

Ex.:

"`vndLightingWithSensor`"

protocol

It can be defined independently. However, type and version should be described.

Ex.:

```
"type": "originalProto", "version": "v1.0"
```

manufacturer

`"code": ""` (empty string). Descriptions should be described by independent definitions.

7.5. Guidelines for expansion of device information (device description)

Regarding device information (Device Description) described in 5.7, it is recommended to use the device information defined in the "Device specification section" (separate document) without changing the name. It is acceptable for each device to delete properties/property values and actions that are not supported.

If adding new devices or adding new properties to existing device information, the device information may be expanded by the vendor independently. For the purpose of avoiding duplication of names if the definition of the Device specification section is expanded in the future, the following shows a summary of the guidelines for adding/expanding device information by the vendor independently.

7.5.1. Adding new devices

It is acceptable to newly define devices not defined in the "Device specification section" (separate document), including ECHONET Lite devices not defined in the "Device specification section" (separate document), virtual devices described in 4.4, and devices with specifications other than ECHONET Lite.

If adding a new device, prefix the deviceType name with "`vnd`". It is not necessary to prefix the names of properties and actions with "`vnd`". If there is no definition corresponding to the APPENDIX Detailed Requirements for ECHONET Device Objects, "`eoj`" and "`epc`" may be omitted.

7.5.2. Expansion for existing device information

If making changes to device information already defined in the "Device specification section" (separate document), under any of the following three conditions, it is ideal to consider it as a vendor-specific expansion.

1. To add a property or action with a different function than an existing property or action. If adding a new property or action to devices that have already been defined in the "Device specification section" (separate document), the name of the property or action should be prefixed with "`vnd`". If there is no definition corresponding to the APPENDIX Detailed Requirements for ECHONET Device Objects, "`epc`" may be omitted.

   Ex.:
   "`vndYUV`" (RGB for general lighting class)

2. If properties or actions are defined with the same name as an existing property or action, but the schema definition has no common property values at all. However, if even one property value is duplicated, this condition should not be applicable.

   Ex.:
   the type of an existing definition is "`number`" and the type of independent definition is "`string`"
   The name of the property or action should be prefixed with "`vnd`", as in the 1.

3. If adding an enumerator to the property value of an existing enumerated type (`enum`). If adding a new enumerator to a property value of an enumerated type (`enum`) defined in the "Device specification section" (separate document), "`vnd`" should be used as the prefix.

   Ex:
   adding an enumerator "`vndExtremeCooling`" to the `"enum":[ "auto", "cooling", "heating"]` of the operation mode

For the details on the property object, it is recommended not to change the contents defined in "Table 5-4 Details of property objects". However, vendors may add elements independently. The name in such a case should be prefixed with "`vnd`".

Ex.:
`vndOriginalProtoCode`

The contents defined in "Table 5-4 Details of property objects" should be handled as follows.

epc

Can be omitted (Not required for devices other than ECHONET Lite devices)

epcAtomic

Can be omitted (Not required for devices other than ECHONET Lite devices)

descriptions

Described by independent definition.

urlParameters

Can be described by an independent definition. However, it should be described using the JSON Schema.

schema

Can be described by an independent definition. However, it should be described using the JSON Schema.

7.6. Guidelines for combined uses of defined devices

If combining multiple defined devices to use them as a single device, it is recommended to define them based on the guidelines described below.

7.6.1. Guidelines for deviceType names

List the deviceType names of the defined devices in descending alphabetical order and then define them in lowerCamel. It is recommended to use "_" as a separator in the "`deviceType`" name.

[Example definition]

If combining temperature sensor ("`temperatureSensor`") and CO2 sensor ("`co2Sensor`"):

```
"deviceType": "co2Sensor_temperatureSensor"
```

7.6.2. Guidelines for property names

To avoid duplication of property names within the devices to be defined in combination, for necessary properties among the device properties to be used in the combination, the property name should be prefixed with "`deviceType`" name.

[Example definition]

For the cases where deviceType: "`temperatureSensor`", property name: "`value`";

Property name: "`temperatureSensorValue`"

7.6.3. Guidelines for definition of Device Description

To clarify the devices whose properties are cited, it is recommended to describe EOJ of the device for each property. For defined devices not having EOJ, deviceType can be described. The location of the description should be in the definition of each property, and "key" should use "`baseEoj`" for EOJ, while "`baseDeviceType`" for "`deviceType`" (deviceType may be listed alongside for devices having EOJ). The listing order of the properties is arbitrary, and only the property names should be changed and described following to the guidelines shown in 7.6.2.

[Definition Example 1]

If all of the devices to be combined have EOJ

```
{
    "deviceType": "co2Sensor_temperatureSensor",
    "descriptions": {
        "ja": "CO2 温度センサ",
        "en": "CO2 Temperature sensor"
    },
    "properties": {
        "co2SensorValue": {
            "baseEoj": "0x001B",
            "baseDeviceType": "co2Sensor",  // Can be omitted
            "epc": "0xE0",
            "descriptions": {
                "ja": "CO2濃度計測値",
                "en": "Measured value of CO2 concentration"
            },
            ...snip...
        },
        "temperatureSensorValue": {
            "baseEoj": "0x0011",
            "baseDeviceType": "temperatureSensor", // Can be omitted
            ...Omitted hereafter
```

[Definition Example 2]

If not all of the devices to be combined have EOJ

```
{
    "deviceType": "co2Sensor_vndPm25Sensor",
    "descriptions": {
        "ja": "CO2 PM2.5センサ",
        "en": "CO2 PM2.5 sensor"
    },
    "properties": {
        "co2SensorValue": {
            "baseEoj": "0x001B",
            "baseDeviceType": "co2Sensor", // Can be omitted
            "epc": "0xE0",
            "descriptions": {
                "ja": "CO2濃度計測値",
                "en": "Measured value of CO2 concentration"
            },
            ...snip...
        },
        "vndPm25SensorValue": {
            "baseDeviceType": "vndPm25Sensor",
            ...Omitted hereafter
```

7.7. echoCommand

Some services using Web API may want to use the EOJ, ESV, EPC, and EDT values defined in ECHONET Lite. To respond to such requests, the APIs shown in the table below are defined. A command using the values of EOJ, ESV, EPC, and EDT is called "echoCommand".

http method	path	Description
GET	/nodes	Obtaining list of node profile information
POST	/nodes/<node id>	Sending echoCommand

GET /nodes

Obtains node profile information (identification number and instance list) for ECHONET Lite devices connected to the home LAN as "`id`" and "`instances`".

■ Definition of response

```
{
    "nodes": [
        {
            "id":<node profile identification number>,
            "instances": [
                {
                    "eoj":<device object EOJ>
                }
            ]
        },
      ...
    ]
}
```

■ Example of response

```
{
    "nodes": [
        {
            "id": "FE...0001",
            "instances": [
                {
                    "eoj": "0x013001"
                }
            ]
        },
        {
            "id": "FE...0002",
            "instances": [
                {
                    "eoj": "0x026B01"
                },
                {
                    "eoj": "0x028101"
                },
                {
                    "eoj": "0x028201"
                }
            ]
        }
    ]
}
```

Response:

Property	Type	Required	Description
`node`	array	Yes	List of node profile information
`nodes[].id`	string	Yes	Node profile identification number (EPC:0x83)
`nodes[].instances`	array	Yes	Instance list
`nodes[].instances[].eoj`	string	Yes	Device object EOJ

POST /nodes/<node id>

Specifies the "`id`" value obtained by "`GET /nodes to <node id>`", and sends echoCommand described in JSON data as body data. The value of each element of echoCommand should be described by a character string in hexadecimal notation (e.g., "`0x80`" and "`0xA0`"). OPC and PDC are not described because they can be calculated from the contents of echoCommand. Since EDT is variable length data, the "`edt`" value should be described as an array whose elements are the values of each byte of EDT (e.g., "`0x123456`" is ["0x12", "0x34", "0x56"]). "`deoj`" is required only for the body data of the request. "`seoj`" is required only for the body data of the response.

■ Definition of request

```
{
    "echoCommand": {
        "deoj":<DEOJ>,
        "esv":<ESV>,
        "operations": [
            {
                "epc":<EPC>,
                "edt": [<EDT DATA>
                ]
            }
        },
        ...
    ]
}
```

■ Definition of response

```
{
    "echoCommand": {
        "seoj":<SEOJ>,
        "esv":<ESV>,
        "operations": [
            {
                "epc":<EPC>,
                "edt": [<EDT DATA>
                ]
            }
        },
        ...
    ]
}
```

■ Example of request: obtaining operation status of air conditioner

```
{
    "echoCommand": {
        "deoj": "0x013001",
        "esv": "0x62",
        "operations": [
            {
                "epc": "0x80"
            }
        ]
    }
}
```

■ Example of response: obtaining operation status of air conditioners

```
{
    "echoCommand": {
        "seoj": "0x013001",
        "esv": "0x72",
        "operations": [
            {
                "epc": "0x80",
                "edt": [
                    "0x30"
                ]
            }
        ]
    }
}
```

■ Example of request: setting operation status of air conditioner to OFF

```
    {
        "echoCommand": {
            "deoj": "0x013001",
            "esv": "0x61",
            "operations": [
                {
                    "epc": "0x80",
                    "edt": [
                        "0x31"
                    ]
                }
            ]
        }
    }
```

■ Example of response: setting operation status of air conditioners to OFF

```
    {
        "echoCommand": {
            "seoj": "0x013001",
            "esv": "0x71",
            "operations": [
                {
                    "epc": "0x80"
                }
            ]
        }
    }
```

Request:

Property	Type	Required	Description
`echoCommand`	object	Yes	echoCommand object
`echoCommand.deoj`	string	Yes	DEOJ
`echoCommand.esv`	string	Yes	ESV

Property	Type	Required	Description
`echoCommand.operations`	array	Yes	List of operations
`echoCommand.operations[].epc`	string	Yes	EPC
`echoCommand.operations[].edt`	array	No	EDT (not required if ESV=0x62)
`echoCommand.operations[].edt[]`	string	No	EDT values per byte

Response:

Property	Type	Required	Description
`echoCommand`	object	Yes	echoCommand object
`echoCommand.seoj`	string	Yes	SEOJ
`echoCommand.esv`	string	Yes	ESV
`echoCommand.operations`	array	Yes	List of operations
`echoCommand.operations[].epc`	string	Yes	EPC
`echoCommand.operations[].edt`	array	Yes	EDT
`echoCommand.operations[].edt[]`	string	Yes	EDT values per byte

8. Conclusion

This document indicates various guidelines that need to be considered if designing Web API for formulating a system that can be effectively used by an external client through particular services, which are mapping in-home ECHONET Lite models on a server (e.g. cloud), and converting in-home ECHONET Lite devices to Web API. In addition to basic use cases such as device operation described in the previous guideline, this document focuses on applied use cases (e.g., batch instructions for multiple commands, device grouping, and historical data devices), to broaden the range of services that can be provided as well as to provide more sophisticated services. In the future, while gradually increasing the types of devices supported, further efforts will be made to improve applied use cases and to develop enhanced functionalities and environments that are friendly to service operators.

9. Acknowledgments

To create this document, we received tremendous support in research and writing from Hiroyuki Fujita at the Kanagawa Institute of Technology, who participated in our working groups. We deeply appreciate their efforts. We would also like to express our sincere gratitude to everyone who participated in the discussions.