ECHONET Lite Web API
ガイドライン
API 仕様部

Version 1.1.2

改定履歴

日付	版	説明
2018/10/03	Ver.1.00	制定、一般公開
2020/03/27	Ver.1.1.0 draft	バージョニング方針変更：1.**->1.*.*
		本書を「API 仕様部」と名称変更
		ガイドライン構成（3 章)、応用サービス（7 章）追加
		旧 3.7「対象ユースケースの絞り込み」を削除
		5.10 に「Webhook による INF 実現例」を追加
		4 章のユースケース内容を再整理
		6.5 に「PATCH」メソッドを追加
		6.4 EPC/EDT/Action 関連の説明追記
		表 5-1 に基本モデル API 一覧を追記
		表 6-3 に機器オブジェクト操作 API 一覧を追記
		全体的に「ユーザ」「クライアント」の表現見直し
2020/08/27	Ver.1.1.0	5.7 に Action Object 説明追記
		6.5 の SetGet 例を Set 例へ変更
		全体的に「リソース」「サービス」の表現見直し
		5.5 にサービス例として bulks, histories を追加
2020/10/19	Ver.1.1.1	7.2 の group description 内容を修正（誤記対応）
2021/06/25	Ver.1.1.2	5.7, 7.5.2 の querySchema を urlParameters へ変更

目次

図目次

表目次

第１章 はじめに

本書は、ECHONET Lite 規格を活用し Web API によるサービス提供を実現するために考慮すべき観点や参考事例についてまとめたガイドラインとなる。ECHONET Lite のモデルを、サーバ環境を介してクライアント等へ提供し、提携サービスや応用アプリケーションの開発など利用促進に繋げるべく、Web API を用いたシステム構築やアプリケーション実装の際に参考にすべき指針について整理する。図 1-1は、本ガイドラインにて想定する全体モデルであり、各種サービス事業者（クライアント）が ECHONET Lite Web API 対応クラウド（サーバ）に対して統一的なスタイルでアクセスし、 当該クラウドを介してユーザ宅内・構内に置かれた IoT 機器への操作・監視などが可能になる。本書の提供により、エコーネットの更なる普及・拡大が進むことを期待する。

以降、第２章で本ガイドラインが対象とする範囲について定め、第３章でガイドライン文書の構成、第４章で検討候補となるユースケースについて整理する。第５章では、Web API 化における基本方針として、リソース設計やインタフェース設計、認証・許可方式などについて、第６章では、ECHONET Lite 仕様をベースに Web API への対応（変換）ルールやエラー表現などについて述べる。第７章にて、応用モデルとして、より高機能な API を提供するための指針について言及する。

本書は設計指針を示すガイドラインの位置づけ（「API 仕様部」）とし、Web API 化される具体的な機器オブジェクトのデータ型や（プロパティに相当する）各種サービス定義などは、「機器仕様部」として別書にて提供する。

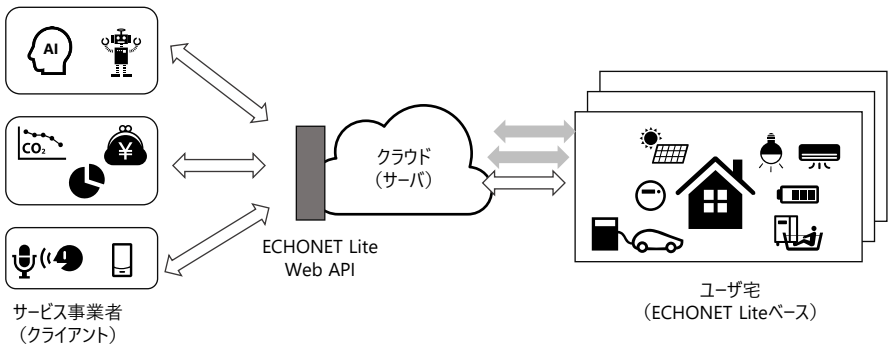

図 1-1　本ガイドラインの想定モデル

1．1 用語

Web API	HTTP(S)プロトコルを用いたシステム間のインタフェースであり、リクエスト／レスポンス方式でデータをやりとりする。本書では、特に呼び出される側のシステム（サーバ側）で提供されるものを指す。
RESTful	URI と HTTP メソッドを用いて操作するリソースを特定する設計原則
JSON	軽量なテキストベースの言語非依存データ交換形式
OAuth2.0	権限の認可(authorization)を行うためのオープンスタンダード

1．2 参照規格

本仕様で参照する規格を以下に挙げる。本仕様書に明示的な説明がない事柄については、規格文書に従う。

[EL] The ECHONET Lite Specification Version 1.01 以降
[ELOBJ] ECHONET Specification APPENDIX: ECHONET 機器オブジェクト詳細規定 Release J 以降
[HTTP] Hypertext Transfer Protocol (HTTP/1.1) RFC 7230～7235, https://tools.ietf.org/html/rfc7230 ～ 7235
[JSON] The JavaScript Object Notation (JSON) Data Interchange Format RFC 8259, https://tools.ietf.org/html/rfc8259
[OAUTH] OAuth 2.0 for Native Apps (RFC 8252), The OAuth 2.0 Authorization Framework: Bearer Token Usage (RFC6750)

第２章 ECHONET Lite Web API の対象範囲

本章では、対象範囲として想定するシステム構成について図 2-1 に示す。

基本的には、1 つの宅内に（1 台以上複数可の）ECHONET Lite 対応コントローラが存在し、その配下に ECHONET Lite 対応の機器が複数台接続されるモデルを想定する（図中左)。コントローラと機器の組み合わせは複数組存在しても良い。通常はコントローラを介してベンダ保有のサーバ X（クラウド）へ接続されるケースが想定されるが、コントローラを介さず直接サーバ X へ接続可能な場合も対象に含んで良い（図中中央)。さらに、非 ECHONET Lite 機器であってもサーバ X へマッピングすることで、本書で示す Web API 形式のモデルへ変換可能であれば、こうした機器も対象可能とする（図中右)。これらコントローラや機器をどのようにサーバ X 上へマッピングするかについては、実装依存であるとし、本ガイドラインのスコープ外とする。

また、本書では、サーバ X は事前にサービス・アプリ提供者であるクライアント A を知っており、適切な認証・認可を与えている（もしくは与えることが可能である）ことを前提としている。具体的な認証・認可手順は、サーバ毎の実装形態にて規定する事項とし、事例紹介に留める。

クライアント A、クライアント B などのクライアント間でのネゴシエーションや、サーバ X やその他のサーバ間でのネゴシエーションも実施して構わないが、本ガイドラインでは対象としない。本書では、議論の単純化のために、クライアントとサーバ間にてサーバ側が提供する Web API についてのみ規定する方針とした。

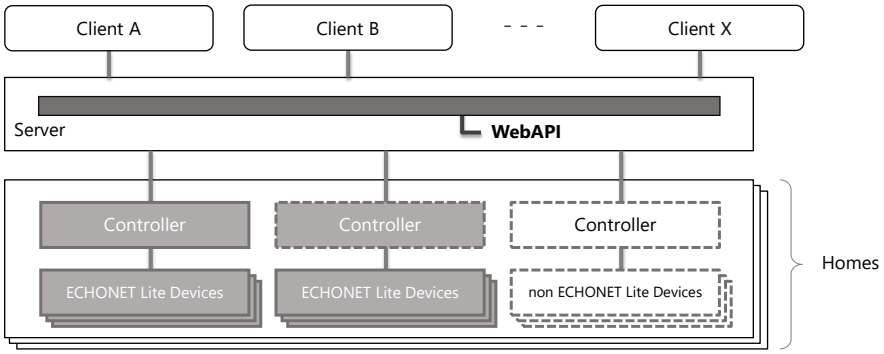

図 2-1　本ガイドライン対象の基本システム構成図

あらためて、本ガイドラインが規定する API の範囲を図 2-2 に示す。ECHONET Lite Web API は、サーバがクライアントに対して提示する Web API のみを対象としており、サーバから宅内との通信部については対象外（ベンダ固有）とする。

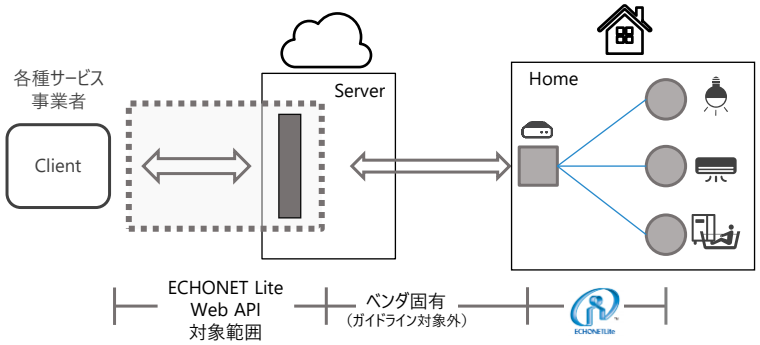

図 2-2 本ガイドラインの対象範囲

第３章 ガイドラインの構成

ECHONET Lite Web API ガイドラインは、「API 仕様部」（本書）と「機器仕様部」（別書）から構成される。

「API 仕様部」は、同ガイドラインが対象とするユースケースや Web API モデルの指針、ECHONET Lite 仕様から Web API へのマッピング指針などが提供される。

「機器仕様部」は、主に、機器毎の Device Description（搭載データ型などのスキーマ規定 : 後述）を例示したもので、使用するデータ型やネーミング指針なども提供される。

各仕様部の更新・メンテナンス方針として、API 仕様部の更新は、応用サービスなどの機能拡充や仕様追加などに伴い実施され、機器仕様部の更新は、対象機器やプロパティの追加などに伴い実施されるものとしている。

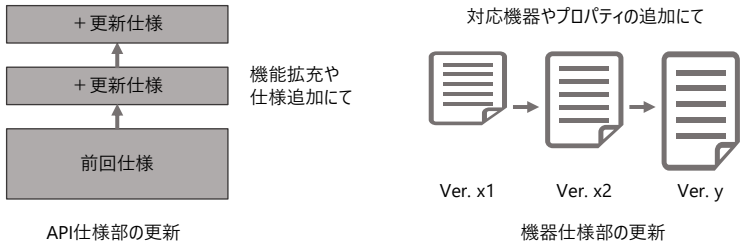

図 3-1　API 仕様部・機器仕様部の更新方針

本ガイドラインのバージョニングは、セマンティックバージョニングを採用する。Version X.Y.Z（メジャーバージョン、マイナーバージョン、パッチバージョン）の形式とし、各々、API 変更の際に互換性がなくなる場合はメジャーバージョンを上げ、後方互換性を保ちつつ機能を追加・変更する場合はマイナーバージョンを上げ、後方互換性を伴うバグ修正などはパッチバージョンを上げる。

第４章 ECHONET Lite Web API のユースケース

本章では、本ガイドライン策定におけるユースケースについてまとめる。

４．１ 状態取得・制御・通知

ECHONET Lite の基本操作（ESV）である GET/SET/INF をマッピングするユースケース（図４-１）。GET については、宅内機器までの応答遅延を考慮し、サーバ内でキャッシングしたデータをクライアントへ返却するケースも含む。SET は、基本的に宅内機器への操作を想定する。INF のように、機器にて生じた状態変化通知などをできるだけ即座にサーバを介してクライアントへ伝えるケースも想定される。

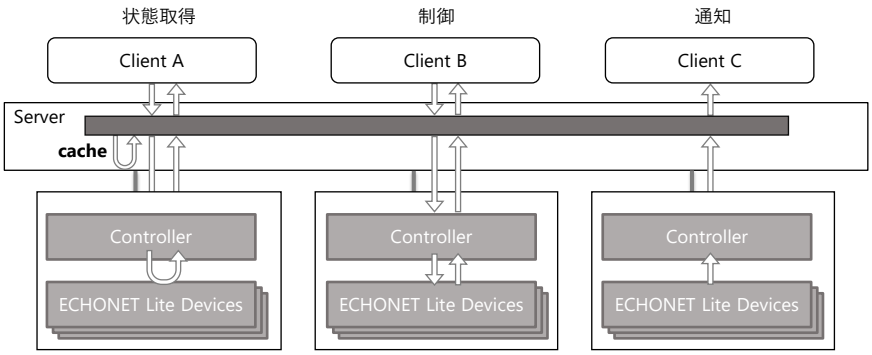

図 ４-１ ユースケース：状態取得・制御・通知

４．２ 機器の一覧取得・管理情報取得

機器の一覧（リスト）を取得するユースケース（図 ４-２の左）。ある機器ユーザ宅の機器全ての一覧を取得するケースや、エアコンのみといった機種指定による機器の一覧を取得するケースが考えられる。さらに、サーバが収容している全ての機器ユーザが保有する機器全ての一覧や、その一部を切り取って扱うケースなども考えられる。

また、ECHONET Lite コントローラは機器オブジェクトとしてコントローラクラスを搭載しており、オプション機能として同コントローラが管理対象とする機器のリストなど、機器管理情報を保持することが可能となっている。こうした機器管理情報をサーバ側へマッピングし、クライアントから取得可能とするケースが考えられる。機器管理情報は、サーバ上でキャッシュし、機器追加・削除などの変更時のみ通知によりサーバ内にキャッシュした機器管理情報を更新するケースも想定される（図 ４-２の右）。

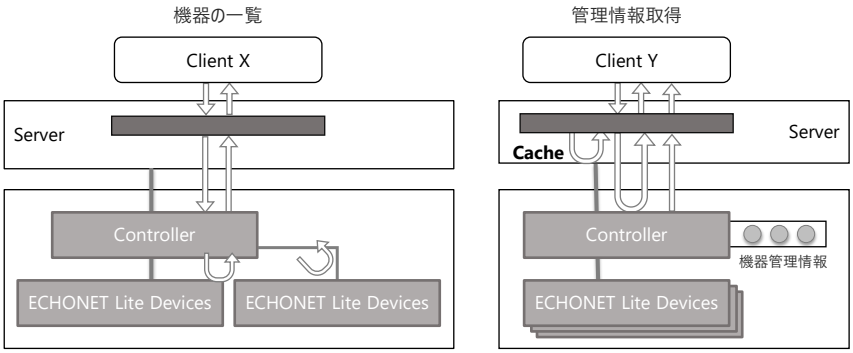

図 4-2 ユースケース：機器の一覧取得・管理情報取得

４．３ 機器の一括操作

ECHONET Lite では、コントローラ配下の全機器への一括操作や、エアコン全てといった機種指定による一括操作がサポートされている。これらをクライアントから操作可能とするユースケース。一人のユーザだけでなく、複数の機器ユーザ宅内の機器を一斉に操作するケースも考えられる。

（宅内もしくは複数の機器ユーザを跨がり）複数の機器を一斉に操作する場合、同一のプロパティへの操作を行うケースもあれば、機器ごとに異なるプロパティへの操作を行うケースもある。また、操作指示（GET/SET）についても機器ごとに異なるケースも考えられる。一台の機器に対して複数の操作を行うケースもありうる。いずれも、一度の指示で一台以上の機器に対して複数の操作が行えるケースとして考えることができる。

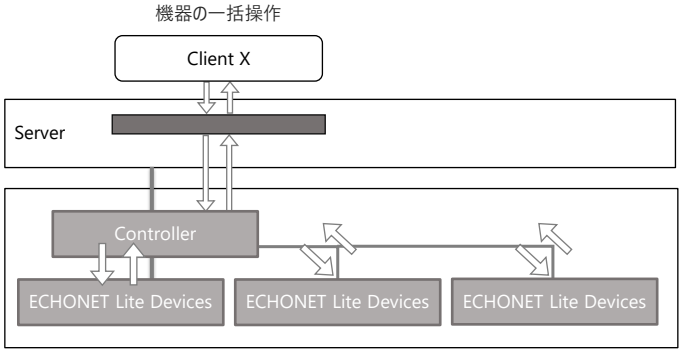

図 4-3 ユースケース：一括操作

４．４ 仮想機器

宅内の機器をサーバ上へマッピングする際、様々な情報加工を施すことで、見かけ上、機器の機能を拡張することが可能なユースケース。仮想機器として見せかけるバリエーションとしてここでは3種の想定例について列挙する（図 4-4）。

例えば、ある機器オブジェクト仕様 Appendix Release X 対応のエアコンをサーバへマッピングする際、Release X 時から最新の Release Y においても規格書の該当内容に関するプロパティについて変更が無い場合は、Release Y 対応エアコンとしてクライアントへ見せかけることが可能となる。旧

モデル対応エアコンであっても、サーバ上での適切な変換処理により、新モデル対応エアコンと同様に扱えるなら、新モデルにしか対応しないクライアント提供アプリにも対応が可能となるメリットがある（図左）。

また、宅内のエアコンとなんらかのセンサを組み合わせて、新センサ付仮想エアコンとみなして、クライアントのアプリへ提供することも考えられる。エアコンとセンサを個別の機器として扱っても良いが、一体化することによりアプリの幅が広がる可能性がある（図中央）。

さらに、エアコン自体が保持しない機能であっても、サーバ上で付加機能を加えることで拡張エアコンとしてクライアントへ公開するケースも考えられる。天気予報付きエアコンのようなものがあっても良いかもしれない（図右）。

仮想機器を扱う事例のバリエーションとして、非 ECHONET Lite 機器を扱うことも考えられる。

複数のエネルギー関連機器が保有するエネルギー関連情報（例えば、発電能力、蓄電能力、負荷情報など）を収集し、サーバ内で集約することで仮想（EMS）情報機器としてクライアントへ提供することも考えられる。

複数の機器を同一目的の下にとりまとめ、例えば、エネルギー情報の取得や、充放電といったエネルギー制御などを可能とする操作 API を規定することで、ひとつの仮想的な機器として扱うことが考えられる。

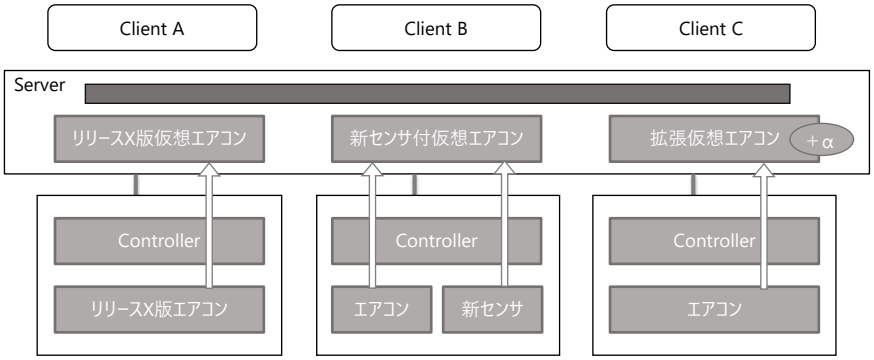

図 4-4　ユースケース：仮想機器

４．５ サーバ蓄積ログ

宅内機器やセンサから得られる情報をサーバ上にて蓄積ログとして集積し、生活行動分析などに繋げるサービスも考えられる。

これらは、直接宅内 ECHONET Lite 機器をマッピングしたものではなく、機器から収集される情報を集約・加工したものをクライアントへ提供する形式となる（図 4-5）。

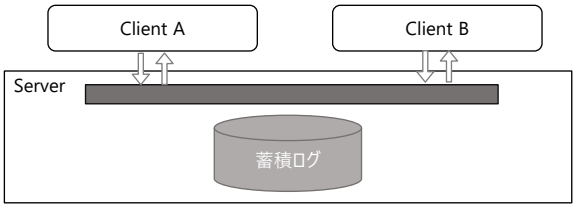

図 4-5 ユースケース：サーバ蓄積ログ

４．６ 認証・認可

今回規定するような顧客情報を扱うケースでは、そもそもクライアントがサーバから Web API によるサービス提供を受けるにあたり、なんらかの認証や認可を得る必要があるのが一般的である。具体的には、OAuth2.0 [OAUTH]などのプロトコルを用いてセッションを確立した上で、サーバは、クライアント毎にアクセス許諾するリソース集合を準備し、実際にサービス提供することになる。

また、サーバ上では、顧客情報の追加／削除や、顧客保有のコントローラや機器の追加／削除を実施可能な仕組みが必要となる。こうした顧客管理、機器管理機能を Web API にて外部提供することも考えられるが、サーバと顧客間であらかじめ必要な作業を行う形式も考えられる（図 4-6）。

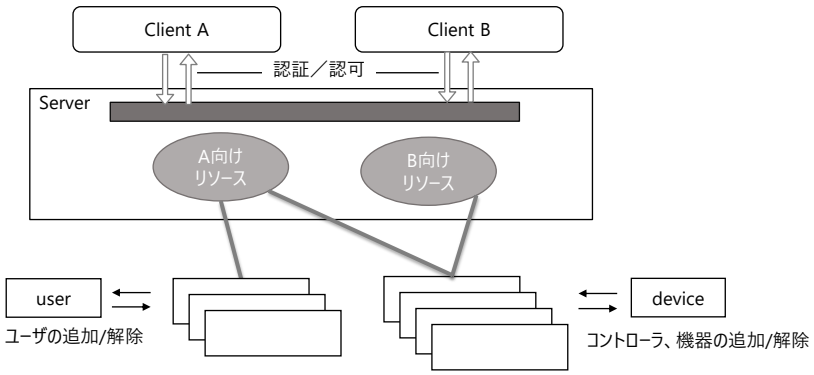

図 4-6 ユースケース：認証・認可

第５章 Web API モデルの指針

５．１ 基本方針

本ガイドラインでは、Web API のモデルとして、基本的に REST（REpresentational State Transfer）を採用する。REST は、URI に対して HTTP [HTTP]メソッドをベースにシンプルな制御が可能であり、現在実質的な Web API の標準になっている。データ形式には現在主流である JSON [JSON]を採用し、Rest(ful) API となるよう全体設計する。

REST の 4 原則である、①HTTP によるステートレス性、②URI によるアドレス可視性、③HTTP メソッドによる統一インタフェース、④JSON によるリソース間の接続性に留意しつつ、API を規定する。

リソースの識別方法は、下記を基本とする。

```
形式　スキーム://ホスト名/リソースへのパス
例　　https://www.example.com/elapi/v1/devices/12345678
```

この例では、HTTPS スキームにて、ホスト `www.example.com` が保有する ID が 12345678 の機器データのリソースを指定している（ポート番号については省略）。このように、パスで階層化してリソースを指定する REST API を基本モデルとする。また、クエリーパラメータでリソースを指定するクエリーAPI も補完的に選択可能とする。具体的なリソースの指定方法については、次節以降記述する。

リソースに対するアクションの多くは、CRUD（Create、Read、Update、Delete）で表現される作成、参照、更新、削除の各操作を参考に、リソースの URI に対する HTTP メソッドや URI のクエリーパラメータにて実装されるモデルを想定する。

前述以外の基本方針として、下記を定める。

- HTTP のバージョンは HTTP/1.1[HTTP]を採用する。
- JSON 形式は、RFC 8259 に従う。Content-Type として「application/json」を指定し、文字コードは UTF-8 を使用する。クライアント側では、Accept というリクエストヘッダでメディアタイプ「application/json」を指定する。
- スキームには、HTTPS を使用する。
- 日時形式には、RFC 3339（ISO 8601）を採用し、タイムゾーンを考慮した下記表記に従う。
  2018-01-02T12:34:56+09:00
  ただし、年月日、時、分、秒など個別に指定すべきケースは個々の記述に従う。
- リソースに指定されるプロパティ名称など属性名の命名規則として、単一語の場合は小文字を使用するが、複合語の場合は（ローワー）キャメルケースを使用する。本ガイドラインでは説明の都合、任意の値や記述を表現する場合には山括弧「<>」を使用する。

- ECHONET Lite の機器オブジェクトの Device Description 形式（後述）は、W3C で策定中の WoT モデルを一部参考として記述する。

次節以降、下記のサービス指定・操作例について記述する。

- アプリケーション名
- API バージョン、API バージョン一覧取得
- 対象サービス種一覧取得
- 機器の一覧取得
- 機器情報（Device Description）取得
- 機器オブジェクトのプロパティ値操作（SET/GET など）
- キャッシュの扱い
- 機器オブジェクトのプロパティ値通知（INF）
- 認証・認可

以下で示す事例では、HTTP(S)リクエストでは、リクエストライン、リクエストヘッダ、メッセージボディ、HTTP(S)レスポンスでは、ステータスライン、レスポンスヘッダ、メッセージボディの内容のうち、（リクエスト／レスポンス）ヘッダ部については特別明記する必要がある場合を除き例示を省略する。

表 5-1 に、基本モデルに関する API 一覧を示す。以降、個別に説明する。

表 5-1 本書規定の基本モデルに関する API

http method	path	description
GET	/elapi	API バージョン一覧取得
GET	/elapi/v1	対象サービス種一覧取得
GET	/elapi/v1/<サービス指定省略可）>/devices	機器一覧取得
GET	/elapi/v1/devices/<device id>	機器情報（Device Description）取得（サービス種省略）

５．２ アプリケーション名

本書にて規定する ECHONET Lite Web API を用いたアプリケーション名称を「elapi」と呼称する。トップ階層に「/elapi」というアプリケーション名をパスとして埋め込むことで、ECHONET Lite Web API のコンポーネントであることを示す。

なお、ベンダにより「elapi」とは異なる別のアプリケーション名称を使用したい場合は、ベンダ所望の名称を用いても構わない。以降、本書で示される事例では、アプリケーション名称を「elapi」と呼称するが、ベンダ所望の名称とする場合は適時読み替えること。

本書にて特定パス収容方式を選定した理由

API の指定方式として、下記のような記述例が候補となる。
① 特定のパスに収容する場合 `https://www.service.xx/elapi/xxx`

② ホスト名で区別する場合　`https://elapi.service.xx/xxx`

Web アプリケーションの実装形態やサービス規模／負荷分散も考慮すべきだが、リバースプロキシなどを用いれば接続先 Web アプリケーションサーバの切り替えは可能なため、本書では①特定パスに収容する方式に統一する。なお、上述の通り、ベンダ所望のアプリケーション名称を用いる場合は、②の形式であっても構わない。

5．3 API バージョン

本ガイドラインで定める API バージョンはリビジョン番号形式で管理するものとし、現時点では v1 とする。今後、細かな仕様追加や修正などを実施する場合であっても、後方互換性がとれる範囲で規定する場合は、バージョンを上げずに運用するものとする。将来、互換性の無い仕様変更が必要となった場合は、仕様を区別するため、v2 を策定する方針とする（メジャーバージョンが 1 から 2 へ更新されたものとする）。なお、バージョン指定は、URI にバージョン情報を埋め込む方法を採用する。

なお、前述の「アプリケーション名」と同様、ベンダ固有のバージョニングを行う場合は、この規定に従う必要はない。ベンダにて規定される方針に従い対応すること。以降、本書で示される事例では、API バージョンを「v1」とするが、ベンダ所望の形式とする場合は適時読み替えること。

URI 指定によるバージョン管理を選択した理由

バージョン指定方法には、下記のように大きく 3 通りの方法がある。

① URI に含める： https://www.service.xx/elapi/v1
② クエリーパラメータに含める：https://www.service.xx/elapi/xx/80234512?v=1.5
③ HTTP ヘッダに含める：`Accept: application/vnd.echonet.v2+json`

②はバージョン情報が省略可能というメリットがあるが、デフォルトバージョンが最新版とすると、API 変更時にトラブルを発生しやすい。また、③はベンダ固有のメディアタイプとしてサーバへ要求し、対応可能な場合は「`Content-Type: application/vnd.echonet.v2+json`」と共に「`Vary: Accept`」をヘッダに付与して応答する必要があるが、`Content-Type` が `application/json` と一致しないと（独自のメディアタイプを）受け付けないライブラリが存在するケースもある。

本書では、識別しやすく、多くの Web API 提供サービスで広く使われていることを理由に、①を採用する。

5．4 API バージョン一覧取得

トップの URI に対して「/elapi」を指定し、GET メソッドにて要求することで、サーバが対応するバージョン一覧を取得できる。

■ リクエスト
```
GET /elapi HTTP/1.1
```

■ レスポンス
```
{
    "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"
        }
    ]
}
```

上記例では、v1 と v2 のふたつのバージョンをサポートしていることを示している。最低 1 つのバージョンはサポートすること。本書に対応したバージョンは v1 とし、将来のバージョン更新は v2 を予定する。開発レベルなどに応じて、status には、下記の状態を指定可能とする。

- CURRENT：最新の安定版
- SUPPORTED：安定版（最新版ではない）
- DEPRECATED：既に廃止となった版
- EXPERIMENTAL：開発・実験中の版

updated で、更新日時を指定可能とし、サーバ側システムに何らかの変更を実施したタイミングを記述可能とする。id は必須、status, updated はオプション指定とする。

５．５ 対象サービス種一覧取得

「/elapi/<version id>」を指定し、GET メソッドにて要求することで、サーバが「/elapi/<version id>」直下で対応するサービス種一覧を取得できる。

■ リクエスト
```
GET /elapi/v1 HTTP/1.1
```

■ レスポンス
```
{
    "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
    }
```

```
    ]
}
```

上記例では、devices, controllers, sites, users, groups, bulks, histories の 7 種類のサービス種をサポートしていることを示している。本ガイドラインではこれらのうちいくつかのサービス種について例示するが、同様のサービス種をサーバが任意に規定できるものとする。サービス種は、必ず 1 種類以上提供すること。クライアントに対して提供しないサービス種は記載しないこと。各サービス種の説明文を descriptions（必須）にて、各オブジェクトの総数を total（オプション）で示す。説明文の記述は任意の文字列とし、総数は整数値とする。基本的に、クライアント（提供先）の要求・権限や提供サービス内容などに応じて、各クライアントへ提供するサービス種を変更して（切り替えて）構わない。

以降、本ガイドラインでは devices, bulks, groups, histories のケースについて規定する。他のサービス種については任意に規定可能とし、本書ではスコープ外とする。

5．6 機器一覧取得

「/elapi/v1/<サービス指定（省略可）>/devices」 を指定して GET することで機器一覧を取得できる。

■ リクエスト

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

■ レスポンス

```
{
    "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": "",
        }
    ]
}
```

表 5-2 機器一覧取得時の応答内容

Property	Type	Required	Description	Example
id	string	Yes	機器固有の ID	"0xFE000006…AB"
deviceType	string	Yes	機器の種類	"generalLighting"
protocol	object	Yes	使用プロトコル	―
protocol.type	string	Yes	ECHONET Lite バージョン番号	"ECHONET_Lite v1.13"
protocol.version	string	Yes	Appendix リリース番号	"Rel.J"
manufacturer	object	Yes	メーカ情報	―
manufacturer.code	string	Yes	メーカコード	"0x******"
manufacturer.descriptions	object	Yes	メーカ名称	―
manufacturer.descriptions.ja	string	Yes	名称（日本語）	"A ベンダ"
manufacturer.descriptions.en	string	Yes	名称（英語）	"Vendor Name A"

id は、機器固有の ID であり、システム内（/devices 配下）にて重複しないユニークな値である必要がある。値の例としては、機器自体が保持する固有情報（MAC アドレスや機器オブジェクトの固有 ID など）や、コントローラが割り当てる値、サーバ内にて固有に割り当てる値などが想定される。上記例では先頭に”0x”を付加した 16 進数の文字列にて記載しているが、”0x”の無い任意の文字列で構わない。

deviceType は、機器オブジェクトのクラス名に相当する名称となる。 ECHONET Lite の機器オブジェクトに対応付けられるクラス名称については、「機器仕様部」（別書）にて記載する。

protocol は、type で ECHONET Lite のバージョン番号を、version で機器オブジェクトの Appendix リリース番号（EPC=0x82）を ASCII 文字列にて記載する。

manufacturer は、ECHONET Lite の機器オブジェクトに対応付けられる機器の場合は、機器オブジェクト・スーパークラスのメーカコード（EPC=0x8A）のプロパティ値を code に、description にて日本語(ja)、英語(en)での名称を記載する。

表 5-2 の Required 列の扱いについては、機器の独自拡張を実施する場合は７．４節記載の通り、柔軟に変更可能とする。

機器一覧は多数となる場合がある。サーバが一度にクライアントへ返却する機器数を制限したい場合には、下記のように、後続機器の有無（"hasMore"）、一度に返却する数（"limit"）、先頭からのオフセット位置（"offset"）を合わせて返却しても良い。

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

```
}
```

また、クライアント側からオフセット位置と返却数を指定するクエリー形式もサポートしても良い。

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

5．7 機器情報（Device Description）取得

「/elapi/v1/<サービス指定（省略可）>/devices/<device id>」 を指定して GET することで、機器に対応する仕様を JSON 形式の 記述「Device Description」として取得可能とする。Device Description は、機器が実装している機能（properties, actions, events）の定義である。

ECHONET Lite の機器オブジェクトに対応付けられる機器の場合、各機器の Device Description は「機器仕様部」（別書）に記述されている（現在、主要機器について記述）。

■ リクエスト

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

■ レスポンス

```
{
    "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>...
    }
}
```

表 5-3 機器情報取得時の応答内容

Property	Type	Required	Description	Example
deviceType	string	Yes	device の種類を示す。ECHONET Lite の機器オブジェクト名(EOJ)に対応する。値に関しては「機器仕様部」（別書）の「5. 機器毎の Device Description」を参照のこと。	"generalLighting"
eoj	string	No	ECHONET Lite の EOJ のクラスコードを string で Hex 表記（ID として使用しない）	"0x0130"
descriptions	object	Yes	ECHONET Lite で定義された機器	—

			オブジェクトの名称	
descriptions.ja	string	Yes	ECHONET Lite の機器オブジェクト	"一般照明"
descriptions.en	string	Yes	device object name of ECHONET Lite in English	"General Lighting"
properties	object	Yes	property object の列挙。property1, property2 は、property resource name（「機器仕様部」（別書）記載）	
actions	object	No	action object の列挙。 action1, action2 は、action resource name（「機器仕様部」（別書）記載）。	
events	object	No	event object の列挙。 event1, event2 は、event resource name。	

Property object

Property object は機器の Property 定義を記述する。ECHONET Lite で GET または SET をサポートするエコーネットプロパティに対応する。本 Web API で property を取得するには GET メソッド、設定するには PUT メソッドを使用する。property object の構成を以下に示す。

```
{
    "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>
    }
}
```

表 5-4 Property Object の内容

Property	Type	Required	Description	Example
epc	string	No	対応する ECHONET Lite の EPC を string で Hex 表記する	"0x80"
epcAtomic	string	No	ECHONET Lite として Atomic operation が必要な EPC	"0xCD"

descriptions	object	Yes	ECHONET Lite で定義されたエコーネットプロパティの名称	—
descriptions.ja	string	Yes	ECHONET Lite のプロパティ名	"動作状態"
descriptions.en	string	Yes	property name of ECHONET Lite in English	"Operation Status"
writable	boolean	Yes	書き込みが可能か不可能かを示す。ECHONET Lite の SET に対応	true, false
observable	boolean	Yes	状態変化を通知できるか否かを示す。ECHONET Lite の INF に対応	true, false

urlParameters	object	No	GET で parameter を渡す必要がある場合に query を利用する。parameter の情報を JSON Schema 形式で記述	{
				"day": {
				"ja": "xx",
				"en": "yy"},
				"schema": {
				"type": "number",
				"minimum": 0,
				"maximum": 99},
				"required": false}
				}

schema	object	Yes	property data の情報を JSON Schema 形式で記述	{ "type": "string", "format": "time" },
note	object	No	Property に関する付加情報	—
note.ja	string	No	付加情報を日本語で記述する	"秒の指定は無視される"
note.en	string	No	付加情報を英語で記述する	"number of seconds is ignored"

Action object

Action object は、property では表現しにくい、機器が提供する機能を記述する。例として、SET のみの property 操作、property 定義をしない機器の内部状態の変更、複数の property のアトミックな変更、時間を要する機器状態の変更(例 時間をかけて照明のフェージングを行う)が挙げられる。また、機器の内部状態に無関係な純粋な関数機能(入力引数だけに対して演算した結果を出力で返す)を action 定義しても良い。ECHONET Lite の機器オブジェクト定義に action は存在しないが、例えば、電気自動車充電器の”積算電力量リセット設定”プロパティに SET することで積算充電電力量を 0 にリセットする機能を、プロパティの代わりに action で定義することが考えられる。本 Web API で action を実行するには POST メソッドを使用する。action object の構成を以下に示す。

```
{
    "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>
    }
}
```

表 5-5 Action Object の内容

Property	Type	Required	Description	Example
epc	string	No	対応する ECHONET Lite の EPC を string で Hex 表記する。対応する EPC が存在しない場合は省略	"0xB3"
descriptions	object	No	action の名称	—
descriptions.ja	string	No	action 名（日本語）	"一括停止設定"
descriptions.en	string	No	action name in English	"All stop setting"
input	object	No	action の入力パラメータ	—
input.type	object	No	入力パラメータの型。object 型固定	"object"

input.properties	object	No	入力パラメータの property object 列	{
				"mode": “sleep”,
				"speed": "fast"
				}

input.required	array	No	必須 property object 列	["mode"]
schema	object	No	action の出力を示す。出力が不要な場合は空とする	{}
note	object	No	action に関する付加情報	—
note.ja	string	No	付加情報を日本語で記述する	"ECHONET Lite では Set only property",
note.en	string	No	付加情報を英語で記述する	"Access rule of the corresponding

				ECHONET Lite property is Set only"

Event object

本版ガイドラインでは Event object について具体的な記述例を示さない。次版以降での追記を予定している。他の通知実現方法（ロングポーリングや Webhook を使用）については、５．１０にて例示する。

５．８ 機器オブジェクトのプロパティ値操作（SET/GET など）

エコーネットでは、プロパティに対する操作を ESV（ECHONET Lite サービス）として規定している。エコーネットにおけるプロパティ値をサーバ上へマッピングしてリソースとして公開し、ESV を HTTP メソッド（CRUD 操作）に対応付けて操作することで、Restful な API 操作が可能となる。機器オブジェクトのプロパティレベルの具体的なマッピングおよび操作については、次章にて議論する。

５．９ クライアントのキャッシュの扱い

クライアントからサーバを経由して宅内機器を操作する（状態取得や制御を実施する）場合、処理によっては、応答まで長い時間を要するケースがある。HTTP キャッシュ（RFC 7234）は必須ではないが、クライアントからサーバに対して値を取得するケースにおいて、キャッシュしたリソースの再利用は有効となるため、参考として例示する。

メーカコードなどの静的データ（基本的に長期不変・固定情報）や、前回アクセスから一定期間データ更新が行われないデータ（動的情報）を扱う場合などにおける HTTP キャッシュの活用例について以下に示す。

HTTP キャッシュのモデルには、Validation Model（検証モデル）と Expiration Model（期限切れモデル）がある。

Validation Model

データ更新されていた場合のみ取得するモデルとなる。

- サーバ：最終更新日付かエンティティタグを必要に応じて使用する

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

- クライアント：最終更新日付にて条件付きリクエストを用いる時

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

- クライアント：エンティティタグにて条件付きリクエストを用いる時

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

- サーバ：変更なければステータスコード 304（と空ボディ）、あれば 200（と変更内容）を返信

Expiration Model

キャッシュの有効期限切れ時のみ取得するモデルとなる。

- サーバ：期限切れ日時か現在時刻からの秒数を指定する。

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

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

前者は、指定された期限までは応答で返されたデータを、クライアントがキャッシュして利用可能であることを示す。
後者は、リアルタイム性が低いケースであり、Date ヘッダからの経過秒数を指定する。
両方使う場合は、Cache-Control 優先となる。
HTTP ヘッダ内の日時形式（RFC 7231 (HTTP1.1)の HTTP-data 記載の 3 種）に対応すること。

- クライアント：上記レスポンスヘッダの値を参考に、次回リクエストの発行タイミングを考える。

上記はキャッシュの有効活用を想定したが、逆に、キャッシュを全く使用しない（させない）ケースも存在する。

（明示的に）キャッシュさせたくないケース

Cache-Control ヘッダを用いることにより、クライアントに毎回アクセスを要求する（クライアントにキャッシュや再利用しないように指示する）ことができる。

- サーバ：検証が必要（現在でも有効か否かをサーバに問い合わせ、確認がとれない限り再利用してはならない）と明示

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

- サーバ：キャッシュをしてはならないことを明示

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

５．１０ 機器オブジェクトのプロパティ値通知（INF）

本ガイドラインで想定する Web API では、基本的にクライアントからの要求に即座に返答する通常の同期的な HTTP REST を想定しているため、機器側のタイミングで非同期的に情報伝達を行う INF を利用するには工夫が必要である。

INF の取扱いについて

最も単純には、サーバ側において機器から INF を受信した際にその値を用いてキャッシュを更新しておき、クライアントからはそれに定期的にアクセスする（ポーリングを行う）ことでその更新を検知する方法が考えられる。このやりとりには５．９にて説明した Validation Model を用いることができる。この手法は非常に簡便だがいくつかの問題がある。一つには情報伝達タイミングがクライアントからのリクエストのタイミングに限られるために、サーバが非同期的に INF を受け取った瞬間にその値をクライアントに伝達することができず常に遅延が生じるという点である。もう一つには、クライアントがその値に関心があるかどうかにかかわらず、サーバでは INF を受け取りうるプロパティ全てに対して履歴を保存する必要があるという点である。

従って、INF を適切に処理するにはサーバ側からクライアントへメッセージを送信するプッシュ通知型の通信方式についてもサポートされることが望ましい。サーバからプッシュ通知を行う手法はいくつか知られている。

- W3C 関連：
  - Push API（Web Push。単一ブラウザ通知。サーバから Push 可能。Android 対応、iOS 非対応）
  - Service Workers
  - Web Notification（単一ブラウザ通知。Android 対応、iOS 非対応）
- スマホアプリ関連（ネイティブ Push）：
  - APNS（Apple Push Notification Service）
  - GCM（Google Cloud Messaging）、FCM（Firebase Cloud Messaging）
- その他：
  - WebSocket
  - MQTT
  - Webhook

また、REST で提供される手段を用いながらもコネクションを比較的長時間持続させるロングポーリングと呼ばれる方法もある。
本節では、ロングポーリングと WebSocket、MQTT、そして Webhook による実現事例について紹介する。

ロングポーリングによる INF 実現例

本来 HTTP REST でのアクセスは、リクエストを受けたら即座にレスポンスを返すことが期待されている。「ロングポーリング」とはこのレスポンスを遅らせ、その間サーバとクライアントの間にコネクションを維持するという手法である。これにより、サーバ側で何か情報が発生した段階で即座にそれを送信することができる。通常の HTTP アクセス手段を流用して実装できる汎用性の高い方法であるが、サーバとの同時接続数が増える傾向にあるため、その対策を講じることや、切断時の再接続をうまくハンドリングする必要性が生じる。

第 6 章で詳細説明される GET メソッドによるプロパティ値の取得方法を踏まえ、INF の受信タイミングをロングポーリングで受け取ることを考える。このリクエストを通常の GET と区別するため、INF 向けには他のメソッド、例えば POST で取得することで、サーバ側にロングポーリングをリクエストすることとする。サーバ側は、値が更新されるまではレスポンスを返さないように実装する。

■ リクエスト
```
POST /elapi/v1/devices/<device id>/properties/<Property resource name> HTTP/1.1
```

■ レスポンス（更新されるまで返答しない。内容は GET の場合と同一である）
```
{
    <property resource name>:<data>
}

or
```

■ レスポンスが object の場合
```
{
    <property resource name>: {
        <element name>: <data> ,
        <element name>: <data> ,
        ...
    }
}
```

さらに、なんらかの要因により接続が切れて再接続するまでの間にサーバが機器から INF を受け取ってしまった時にも、その情報を再接続時に受け取ることを可能にすることが望ましい。これにはリクエストヘッダに If-Modified-Since ヘッダを付与し、その時点以降に変更が生じているならばサーバは即座にレスポンスを返すように実装すればよい。この部分はサーバ側に値を記録しておかねばならないという点で冒頭に述べたキャッシュによる実装に似通っているが、① ロングポーリングでの INF 取得を仮定する場合は切断から再接続までの時間はさほど長くないと想定できるため、履歴を保存しておくべき時間を長くする必要性は低いこと、 ② POST でアクセスされていないプロパティの値を保存する必要はないなどの点から、依然単純なキャッシュ実装より優れている。

WebSocket による INF 実現例

WebSocket をはじめコネクションベースの方法を用いる場合は、通信モデルとして Pub/Sub を用いると簡便である。Pub/Sub ではトピックと呼ばれる文字列を介して通信を行う。このトピックと

してこれまで説明してきたリソース名を用いることで、本 Web API の基本概念を大幅に変更することなく INF 向け拡張を実装することができる。

Pub/Sub モデルをプロトコルレベルでサポートしている場合（MQTT や WAMP など）では、メッセージの配送だけを専門に担当するブローカーやルーターと呼ばれるサーバを用いることが多いが、本節で説明する、WebSocket 上で実装する Pub/Sub では REST からの連続性に留意し、サーバがブローカー機能を兼ね、通信はサーバとクライアント間で起こるものとする。

リクエストとレスポンスの例（サブプロトコルとして「echonet」という文字列を採用している）

■ リクエスト
```
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
```

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

■(クライアント)サブスクリプション
```
{
    "method": "subscribe",
    "path": "/elapi/v1/devices/<device id>/properties/operationStatus"
}
```
■(サーバ)サブスクリプション Ack
```
{
    "method": "subscribeAck",
    "path": "/elapi/v1/devices/<device id>/properties/operationStatus",
}
```
■(サーバ)パブリケーション
```
{
    "method": "publish",
    "path": "/elapi/v1/devices/<device id>/properties/operationStatus",
    "value": false
}
```
■(クライアント)アン・サブスクリプション
```
{
    "method": "unsubscribe",
    "path": "/elapi/v1/devices/<device id>/properties/operationStatus",
}
```
■(サーバ)アン・サブスクリプション Ack

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

MQTT による INF 実現例

MQTT は TCP 上で Pub/Sub モデルを実装した軽量プロトコルで、IoT に適した様々な特性を持っている。その詳細は省くが、これまで説明してきた方法との重要な差異としては、MQTT ネットワークの構成には、メッセージの生成と消費を行うクライアント以外にメッセージの配送を担当するブローカーが必要であるという点である。従って、本節で説明する INF の受け取りに MQTT を用いる場合にも、別にブローカーが必要となる。これまで説明してきたサーバも、その API を利用するクライアントも、MQTT ブローカーの観点からは MQTT クライアントであるが、サーバの方は Publish のみを行う MQTT クライアント(Publisher)、クライアントの方は Subscribe のみを行う MQTT クライアント(Subscriber)となる。

なお、MQTT は TCP 上に直接実装されたバイナリのプロトコルであり、HTTP を用いていないため、正確には Web API ではない。Web クライアント、例えばブラウザから MQTT を利用したい場合は、AWS IoT や Mosquitto など、MQTT over WebSocket を行うことが可能なブローカーを用いる必要がある。

コンセプトは非常にシンプルで、前節で説明した WebSocket でも用いた Pub/Sub が素直に実装されるが、そのシーケンスは QoS と呼ばれる、メッセージの到達保証の度合いをコントロールするパラメータ、およびそれに伴う再送の回数により異なってくる。WebSocket による INF 実現例で提示したリクエストとレスポンスの例と最も近いのは QoS=1 のときであるので、同様のことを MQTT で実現した場合のシーケンス図を次に示す（バイナリプロトコルである MQTT では、リクエストラインのようにメッセージを平易に記述する手法があるわけではない）。

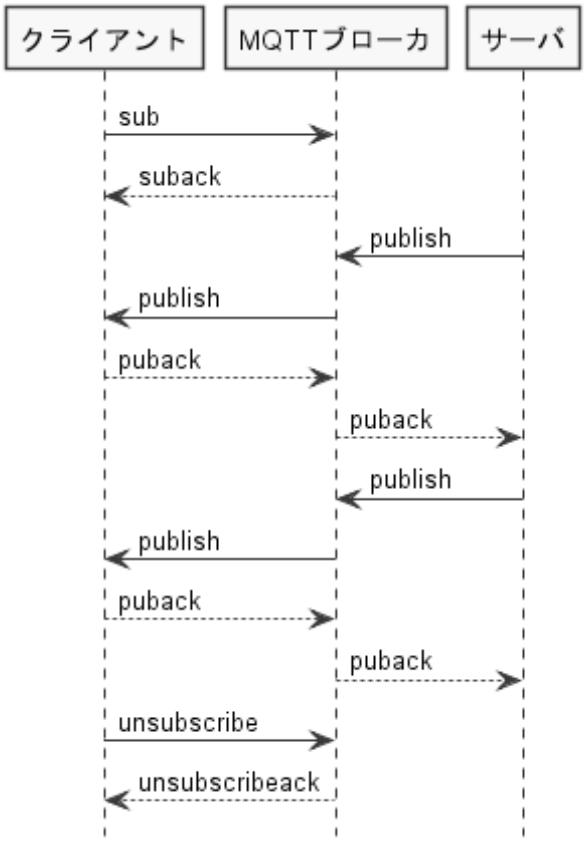

図 5-1 MQTT による INF 実現例

このように、ブローカーが挟まっている以外はほとんど WebSocket と同様である。Publish/Subscribe を行うトピックも、WebSocket のリクエストとレスポンスの例で提示した path の部分（REST で用いたリソース文字列そのもの）とし、ペイロードとして適宜 value を入れて実装すればよい。

Webhook による INF 実現例

Webhook とは、クライアントがあらかじめ指定した URL にサーバが POST メソッドを利用して通知を行う仕組みである。
まず、サーバが webhook をサポートしているかを確認する。レスポンスはサーバがサポートしている通知手段の情報である。以下の例は、サーバが Webhook をサポートしているが、通知の設定は初期状態の場合である。

リクエスト：サーバが Webhook をサポートしているかを確認する

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

レスポンス：Webhook をサポートしているが、subscription は未設定

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

通知を設定する場合は、method で subscribe を指定し、path で通知対象プロパティを、callBackUrl で通知先 URL を指定する。通知先の認証等が必要な場合は、その情報も記述する。以下の例は、通知先の apiKey を設定する場合である。別のリクエストを発行し、path 毎に異なる callBackUrl や apiKey などを設定することも可能である。path で指定したプロパティが既に subscribe されている場合は、callBackUrl や apikey を上書きする。

リクエスト（subscibe するプロパティを設定）

```
POST elapi/v1/notifications

{
    "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"}
    }
}
```

プロパティを指定して通知を解除する場合は method で unsubscribe を指定する。

リクエスト（subscribe を解除）

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

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

Webhook の通知設定がされている場合に通知情報を取得する例を以下に示す。プロパティ operationStatus と operationMode は https://sh-center.org/postreceive に通知し、プロパティ faultStatus は https://sh-center.org/faultStatus に通知する設定であることがわかる。

リクエスト（通知を依頼したプロパティの確認）

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

レスポンス

```
{
    "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"}
            }
        ]
    }
}
```

通知されるデータはこのようになる

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

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

複数のプロパティを一括して subscribe もしくは unsubscribe するには、第 7 章記載の bulks を利用する。

次に、もう 1 つ実現例を説明する。

通知先の URL の登録、解除は何らかの手段（例えば上記の実現例の手順）で行われていることを前提に、Webhook による INF の通知方法を説明する。

HTTP メソッドは、例えば POST メソッドを使用する。また、リクエストが Webhook であることと対象のリソース名を通知する。リソース名の通知には、例えば、リクエストのヘッダフィールドやボディにリソース名を記述する方法がある。ヘッダフィールドにリソース名をセットする場合、ヘッダ名は、例えば X-Elapi-notification とする。ヘッダフィールドやボディに記述するリソース名は、

対象プロパティ・URL の登録時にクライアントから指定されたリソース名を使用するなどして、クライアントが識別可能なものとする。以下では、ボディにリソース名を記述する例として、resource キーに対して、resource URL を記述するものとしたが、device ID、property などのキーを設けるものとしてもよい。

クライアントが複数のサーバから INF を受信する場合、複数のサーバのリソース名（device ID）が重複するケースが考えられる。そのため、①クライアントが、対象プロパティ・URL の登録時に、サーバごとに異なる URL を指定する、②INF の HTTP リクエストのヘッダあるいはボディにサーバを識別する識別子を記述する、などの対応によりクライアントがリソースを一意に特定することができるようにする。サーバを識別する識別子としては、例えば、ホスト名や、対象プロパティ・URL の登録の際の HOST ヘッダの内容を用いる。以下の例では、クライアントがサーバごとに異なる URL を指定しているものとして、INF の HTTP リクエストのヘッダ、ボディにはサーバを識別する識別子を記載しないものとする。

Webhook による INF の例（ヘッダフィールドにリソース名を記述する場合）

■ リクエスト

```
POST <クライアントが登録した任意の URL> HTTP/1.1

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

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

■ レスポンス
リクエストと同様

Webhook による INF の例（ボディにリソース名を記述する場合）

■ リクエスト

```
POST <クライアントが登録した任意の URL> HTTP/1.1

Content-Type: application/json

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

■ レスポンス
リクエストと同様

一般照明の例（ヘッダフィールドにリソース名を記述する場合）

■ リクエスト

```
POST <クライアントが登録した任意の URL> HTTP/1.1

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

{
    "operationStatus": true
}
```

■ レスポンス

リクエストと同様

一般照明の例（ボディにリソース名を記述する場合）

■ リクエスト

```
POST <クライアントが登録した任意の URL> HTTP/1.1

Content-Type: application/json

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

■ レスポンス

リクエストと同様

５．１１ 認証・認可

API 利用者の特定や、呼び出し回数のカウントにより過剰なアクセス時に通信遮断するなど、認証が必要となるケースは多いと考えられる。

本 Web API ガイドラインでは、認証・認可の仕組みについて参考事例として紹介する。Web API における認証・認可方式は、トークン認証や OAuth2.0 認可などが挙げられるが、どの方式を採用するかは、ユースケースによって異なりうる。

ここでは、OAuth2.0 を使用した例を示す。

Grant Type は 4 種（Authorization Code, Implicit, Resource Owner Password Credentials, Client Credentials）あり、用途によって自由に選択可能とする。

（下記は、Resource Owner Password Credentials の例）

OAuth のエンドポイントとして、「/elapi/v1/oauth2/token」を POST メソッドで呼び出すことで、クライアント認証を実施する。リクエストメッセージボディに username や password の値を含み、上記 API リクエストに成功すると、クライアントには「トークン」の情報が返却される。

トークンは、以降の操作に必要な認証用文字列であり、認証以降のすべてのリクエスト時にこのトークンを付与する。

クライアントのリクエスト例：

```
POST /elapi/v1/oauth2/token HTTP/1.1
Host: xxx.com
Authorization: Basic ******
Content-Type: application/x-www-form-urlencoded

grant_type=password&username=usr1&password=xx&scope=xx
```

サーバのレスポンス例：

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
    "access_token": "****",
    "token_type": "bearer",
    "expires_in": 360000,
    "refresh_token": "xxxx"
}
```

クライアントのリクエスト例（リクエストヘッダに入れる場合）：

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

クライアントのリクエスト例（リクエストボディに入れる場合）：

```
POST /elapi/v1/devices HTTP/1.1
Host: xxx.com
Content-Type: application/x-www-form-urlencoded

access_token=****
```

クライアントのリクエスト例（クエリーパラメータに入れる場合）：

```
GET /elapi/v1/devices?access_token=**** HTTP/1.1
Host: xxx.com
```

クライアントのリクエスト例（リフレッシュの場合）：

```
POST /elapi/v1/oauth2/token HTTP/1.1
```

```
Host: xxx.com
Authorization: Basic ******
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token&refresh_token=xxxx
```

サーバのレスポンス例（有効期限切れ）：

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache

{
    "error": "invalid_token"
}
```

第６章 ECHONET Lite 仕様のマッピング指針

本章では ECHONET Lite のフレームや機器オブジェクトの各仕様をどのようにマッピングすべきか、その指針について示す。

６．１ ECHONET Lite フレームのマッピング

ECHONET Lite は、下位通信層非依存の通信プロトコルである。一般的に、IPv4 や IPv6 使用時にはトランスポート層に UDP（必須）、TCP（オプション）を用いて、ECHONET Lite フレーム（表 6-1）をペイロードに搭載して通信を行う。

表 6-1 ECHONET Lite フレームの対応

フィールド	説明	Web API での対応
EHD	ECHONET Lite ヘッダ	非サポート
TID	Transaction ID	非サポート
SEOJ	Source ECHONET オブジェクト	非サポート
DEOJ	Destination ECHONET オブジェクト	機器オブジェクト（インスタンス）の id にて識別
ESV	ECHONET Lite サービス	後述
OPC	処理対象プロパティカウンタ	非サポート
EPC	ECHONET プロパティ	後述
PDC	プロパティデータカウンタ	非サポート
EDT	ECHONET プロパティ値データ	後述

表 6-1 に、ECHONET Lite フレームを構成する各フィールドに関する本 Web API での対応指針について記載する。主に、DEOJ（相手先機器オブジェクト）と ESV（サービス種）、EPC（プロパティ）、EDT（プロパティ値データ）を対象にマッピングすることで、Web API への変換を実現する。EHD や TID、PDC は宅内のコントローラと機器間でのみ関連する事項とし、サーバから外部へは公開されない情報とする。TID は、コントローラと機器間の通信管理に使用され、クライアントとサーバ間のセッション管理の一部として扱うことも考えられる。しかしながら、クライアントからのリクエストに対して、サーバ側がコントローラや機器に対して再送を含む様々な形式で通信を実施するケースも考えられ、サーバ実装に依存することから、TID は本 Web API へのマッピングのスコープ外とする。

６．２ DEOJ のマッピング

ECHONET Lite のオブジェクトは、ノードプロファイルオブジェクトと機器オブジェクトの二種類が存在するが、機器オブジェクトのみ扱う。これは、Web API を介して操作を行うクライアント側アプリケーションは通常、機器オブジェクトへの操作が主体となるためである。通信部の確認など

ノードプロファイルオブジェクトへの操作が必要になるケースも考えられるが、本バージョンのガイドラインではスコープ外とする。

コントローラは機器の一種として扱われる。

下記では、複数の機器を指定する場合の記述例について示し、一覧取得に使用するものとする。なお、一括制御に適用する複数機器の指定方法については、第 7 章の bulks や groups を参照すること。

コントローラすべてを指定したい場合：

全コントローラを指定したい場合は、後述の特定機種すべてを指定したい場合の方法に従いクエリーパラメータとして type=controller を指定する、もしくは、サーバが controller というサービス種をサポートする場合は「/elapi/v1/<サービス指定（省略可）>/controllers」と指定することができる。

機器すべてを指定したい場合：

全機器（全コントローラを含む）を指定したい場合は、「/elapi/v1/<サービス指定（省略可）>/devices」と指定することができる。

特定機種すべてを指定したい場合：

ある機種をすべて指定したい場合は、
「/elapi/v1/<サービス指定（省略可）>/devices?type=homeAirConditioner」のように、機種のタイプを指定すればよい。

インスタンスコードが 0x00 の場合の扱い：

ECHONET Lite では、あるノード上のある機種オブジェクトに関して、インスタンスコードを 0x00 とすることで、同機器オブジェクトのインスタンスすべてを指定可能であるが、本 Web API へのマッピングは行わないものとする

6．3 ESV のマッピング

ECHONET Lite における ESV は、表 6-2 のように定義されている。

表 6-2 ESV の対応

ESV	サービス内容	記号	Web API での対応
0x60	プロパティ値書き込み要求（応答不要）	SetI	PUT メソッド（リクエスト）
0x61	プロパティ値書き込み要求（応答要）	SetC	
0x62	プロパティ値読み出し要求	Get	GET メソッド（リクエスト）
0x63	プロパティ値通知要求	INF_REQ	後述
0x6E	プロパティ値書き込み・読み出し要求	SetGet	PUT メソッド（リクエスト）
0x71	プロパティ値書き込み応答	Set_Res	PUT メソッド（レスポンス）

0x72	プロパティ値読み出し応答	Get_Res	GET メソッド（レスポンス）
0x73	プロパティ値通知	INF	後述
0x74	プロパティ値通知（応答要）	INFC	後述
0x7A	プロパティ値通知応答	INFC_Res	後述
0x7E	プロパティ値書き込み・読み出し応答	SetGet_Res	PUT メソッド（レスポンス）
0x50	プロパティ値書き込み要求不可応答	SetI_SNA	PUT メソッド（レスポンス）
0x51	プロパティ値書き込み要求不可応答	SetC_SNA	PUT メソッド（レスポンス）
0x52	プロパティ値読み出し不可応答	Get_SNA	GET メソッド（レスポンス）
0x53	プロパティ値通知不可応答	INF_SNA	後述
0x5E	プロパティ値書き込み・読み出し不可応答	SetGet_SNA	PUT メソッド（レスポンス）

ESV の種類は、要求、応答、不可応答に大別されるが、Set 操作や Get 操作を HTTP メソッドへマッピングする場合、要求（リクエスト）と応答・不可応答（レスポンス成功・失敗）に対応付けることができる。

Set 系 ESV は、値の更新や制御に相当するため、PUT メソッドへ対応づける。SetI, SetC, SetGet の 3 種類は区別なく、応答にてプロパティ値も返却される形式に統一し、PUT メソッドによるリクエストとレスポンスを使用する。また、AIF 認証仕様にて規定されている機器の応答待ち時間に関連し、Ack の返却、タイムアウト処理などをどのようにマッピングするかは、本バージョンのガイドラインではスコープ外とする。

Get 系 ESV は、値の取得・読み出しであるため、GET メソッドへ対応づける。サーバ上にキャッシングされるデータの扱いについては後述する。

INF 系 ESV は、通常の Web API ではプル・ベースとなるため、リアルタイム通知を受けることは難しいが、擬似的に GET メソッドを用いてポーリングすることや、WebSocket、MQTT、Web Notification API、Webhook などのプッシュ・ベース通信モデルを活用することで実現する。

６．４ EPC、EDT のマッピング

EPC、EDT は、機器オブジェクト仕様とリンクして定義づけられている。EPC は、URI にてリソース定義され、EDT は、リクエストボディやレスポンスボディにてやりとりされる。具体的には、５．７記載の Device Description をベースに、GET 可能なプロパティは、「/elapi/v1/devices/<device id>/properties/<property resource name>」へマッピングされる。プロパティ名称はプロパティリソース名に変換され、「機器仕様部」（別書）にて詳細規定されている。SET 対象となるプロパティは、GET 可能な場合は properties 配下のプロパティリソース名にて扱われるが、GET できない場合は、actions 配下に操作名称を別途定義して扱うことになる。

次節以降、詳細に見ていく。

なお、以降、エンドポイントを示す URI のうち、prefix 部（/elapi/v1）は表記を簡潔にする都合、省略する場合がある。

6．5 ECHONET Lite 機器オブジェクトのマッピング方式および操作

機器オブジェクトの操作に関する基本 API を表 6-3 に示す。

表 6-3 機器オブジェクトの操作に関する API

http method	path	description
GET	/devices/<device id>/properties	GET 対象プロパティ値の一括取得
GET	/devices/<device id>/properties/<property resource name>?<query>	指定プロパティの値取得
PUT	/devices/<device id>/properties/<property resource name>	指定プロパティの値設定
PATCH	/devices/<device id>/properties	SET 対象プロパティ値の複数設定
POST	/devices/<device id>/echoCommands	指定プロパティ値の設定

ECHONET Lite では、二種類のオブジェクト（ノードプロファイルオブジェクトと機器オブジェクト）を定義しているが、本 Web API ガイドラインでは、前述の通り、機器オブジェクトのみを扱う。機器オブジェクトのスーパークラスは、基本的に個々のプロパティがマッピングされるが、プロパティマップなど、JSON 化により自明となる情報は省略される。スーパークラスのプロパティのマッピングについては、「機器仕様部」（別書）参照のこと。

以下、機器オブジェクトの各種操作 API について基本方針を示す。

GET /devices/<device id>/properties

「/elapi/v1/<サービス指定（省略可）>/devices/<device id>/properties」を指定し GET することで、GET 対象となるすべてのプロパティについて値を取得可能とする。

■ リクエスト
```
GET /elapi/v1/devices/<device id>/properties HTTP/1.1
```

■ レスポンス
```
{
    <property resource name>: <property value>,
    <property resource name>: <property value>
    ...
}
```

一般照明の例

■ リクエスト
```
GET /elapi/v1/devices/<device id>/properties HTTP/1.1
```

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

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

「/elapi/v1/<サービス指定（省略可）>/devices/<devic id>/properties/<property resource name>?<query>」を指定し GET することで、指定したプロパティ値を取得可能とする。query は通常は不要だが、スマートメータなど一部機器の特定プロパティで SET(EL)と GET(EL)の atomic operation が必要な場合に SET の data を指定するために query を利用する。key は property 毎に定義する。

■ リクエスト
```
GET /elapi/v1/devices/<device id>/properties/<property resource name>?<query> HTTP/1.1
```

■ レスポンス
```
{
    <property resource name>: <data>
}

or
```

■ レスポンスが object の場合
```
{
    <property resource name>: {
        <element name>: <data> ,
        <element name>: <data> ,
        ...
    }
}
```

一般照明の例

■ リクエスト
```
GET /elapi/v1/devices/<device id>/properties/operationMode HTTP/1.1
```

■ レスポンス
```
{
    "operationMode": "color"
}
```

■ リクエスト
```
GET /elapi/v1/devices/<device id>/properties/rgb HTTP/1.1
```

■ レスポンスが object の場合
```
{
    "rgb": {
        "r": 20,
        "g": 255,
        "b": 0
    }
}
```

■ リクエストに query がある場合
```
GET /elapi/v1/devices/<device
id>/properties/normalDirectionIntegralElectricEnergyLog1?day=0 HTTP/1.1
```

■ レスポンス
```
{
    "normalDirectionIntegralElectricEnergyLog1": {
        "day": 0,
        "energy": [
            20,
            34,
            59,
            109
        ]
    }
}
```

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

「/elapi/v1/<サービス指定（省略可）>/devices/<device id>/properties/<property resource name>」を指定し PUT することで、指定したプロパティ値を設定可能とする。サーバはコントローラを経由して対象機器のプロパティリソースに対応するプロパティに対して ECHONET Lite の SET 操作後に更に GET 操作した値を取得し、クライアントへ返却する。

■ リクエスト
```
PUT /elapi/v1/devices/<device id>/properties/<property resource name> HTTP/1.1
{
    <property resource name>: <data>
}

or
```

■ リクエストが object の場合
```
{
```

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

■ レスポンス
リクエストと同様

一般照明の例

■ リクエスト
```
PUT /elapi/v1/devices/<device id>/properties/operationMode HTTP/1.1
{
    "operationMode": "color"
}
```
■ レスポンス
```
{
    "operationMode": "color"
}
```

■ リクエストが object の場合
```
PUT /elapi/v1/devices/<device id>/properties/rgb HTTP/1.1
{
    "rgb": {
        "r": 20,
        "g": 255,
        "b": 0
    }
}
```

■ レスポンス
```
{
    "rgb": {
        "r": 20,
        "g": 255,
        "b": 0
    }
}
```

上記手順は、「機器仕様部」（別書）記載の各機器を対象に、Device Description の内容に基づいた操作を実行することを想定している。詳細については「機器仕様部」（別書）を参照すること。

一方、「機器仕様部」（別書）で規定する Device Description 内に所望のプロパティ定義が存在しない場合や、ベンダ独自のメーカ依存コードを使用する場合など、本ガイドラインの規定範囲外となる

EPC 操作を可能にする補完機能を導入する（オプション扱い。７．４節にて後述）。

PATCH /devices/<device id>/properties

RFC5789 (PATCH Method for HTTP) にサーバが対応している場合、「/elapi/v1/<サービス指定（省略可）>/devices/<device id>/properties」を指定し PATCH することで、指定した複数のプロパティ値を設定可能とする。指定されなかった他のプロパティは元のままである。

リクエストに設定できないプロパティが含まれることが、サーバで判定できる場合（設定するプロパティ値が範囲外であったり、プロパティリソース名が存在しないなど）は、HTTP レスポンスのステータスコードとして 400 番台を返す。この場合、サーバからコントローラへのリクエストの送信は行われない。

リクエストに設定できないプロパティが含まれることが、コントローラや機器の処理時に明らかになった場合（機器側の内部状態により、プロパティの更新ができないなど）は、HTTP レスポンスのステータスコードとして 500 番台を返す。

必要に応じてエラー応答をレスポンスボディ（JSON 形式）にて返却可能とする（オプション）。設定できなかったプロパティは、PATCH で指定したプロパティ値と共に、エラータイプとメッセージ（任意の文字列）を組として応答する。

■ リクエスト

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

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

■ レスポンス

```
{
    <property resource name>: <propertyValue>,  // PATCH 成功プロパティ
    <property resource name>: <propertyValue>,
    ... ,
    "errors": [ <error object>, <error object>... // PATCH 失敗プロパティ（オプション）
    ]
}
```

■ error object

```
{
    <property resource name>: <property value>,  // PATCH 時に指定されたプロパティ
    "type": <error type>,
    "message": <error message>
}
```

一般照明の例

■ リクエスト（範囲値外の値や、無効なプロパティが含まれる場合）

```
PATCH /elapi/v1/devices/<device id>/properties HTTP/1.1
{
    "operationMode": "color",      // 正常
    "rgb": {
        "red": 20,
        "green": 300,      // 範囲値外の値
        "blue": 0
    },
}
```

■ レスポンス

```
HTTP/1.1 400 Bad Request
{
    "operationMode": "color",      // 正常値（400 番台のエラーなので、リクエスト送信はされない）
    "errors": [    // 異常値（PATCH 要求時と同じプロパティ値を返す）
        {
            "rgb": {
                "red": 20,
                "green": 300,
                "blue": 0
            },
            "type": "rangeError",
            "message": "'green' value is out of range."
        }
    ]
}
```

■ リクエスト（機器から一部のプロパティが処理できず不可応答が返ってきた場合）

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

■ レスポンス

```
HTTP/1.1 500 Internal Server Error
{
    "operationMode": "color",    // 設定成功
    "errors": [    // 設定失敗
        {
            "rgb": {    // 応答時のプロパティ値は実装マター（本例は PATCH 要求時と同じプロパティ値）
                "red": 20,
                "green": 128,
                "blue": 0
            },
```

```
            "type": "deviceError",
            "message": "SetC_SNA"
        }
    ]
}
```

POST /devices/<device id>/echoCommands

「/elapi/v1/<サービス指定（省略可）>/devices/<device id>/echoCommands」を指定し POST することで、指定したプロパティ値を設定可能とする。「機器仕様部」（別書）で規定する Device Description 内に所望のプロパティ定義が存在しない場合や、メーカ依存コードを使用する場合など、本書の規定範囲外となる EPC 操作を可能にする補完機能として導入する。

ESV、EPC、EDT コードは[EL]にて定義される 16 進数の文字列（例：”0x80”）やその配列にて指定する（数字や 10 進数文字列などの使用は本ガイドラインでは対象外とする）。また、複数プロパティも指定可能とする。

POST リクエスト時に要求用 ESV コード、POST レスポンス時に応答・通知用 ESV コードや不可応答用 ESV コードを使用し、リクエストに応じて「epc」や「edt」を返す。レスポンス時の ESV が Get_SNA の場合は、値を読み出せなかったプロパティの値に関して edt を null にて返し、読み出せたプロパティの値を edt で返す。また、Set_SNA の場合は、値を設定できたプロパティの値に関して edt を null にて返し、設定できなかったプロパティの値を edt で返す。機器へ到達しないエラーが発生する場合は６．７に従い、適切にエラーを返却すること。

下記では、要求用 ESV コードとして Get（ESV=0x62）と SetC（ESV=0x61）、応答用 ESV コードとして Get_Res（ESV=0x72）と Set_Res（ESV=0x71）を対象とした例について示している。

Get の例

■ リクエスト

```
POST /elapi/v1/devices/<device id>/echoCommands HTTP/1.1
{
    "request": {
        "esv": "0x62",
        "operations": [
            {
                "epc": "0xF0"
            }
        ]
    }
}
```

■ レスポンス

```
{
    "response": {
        "esv": "0x72",
        "operations": [
            {
```

```
                "epc": "0xF0",
                "edt": [
                    "0x12",
                    "0x34",
                    "0x56"
                ]
            }
        ]
    }
}
```

Set の例

■ リクエスト

```
POST /elapi/v1/devices/<device id>/echoCommands HTTP/1.1
{
    "request": {
        "esv": "0x61",
        "operations": [
            {
                "epc": "0xF0",
                "edt": [
                    "0x12",
                    "0x34",
                    "0x56"
                ]
            }
        ]
    }
}
```

■ レスポンス

```
{
    "response": {
        "esv": "0x71",
        "operations": [
            {
                "epc": "0xF0",
                "edt": null
            }
        ]
    }
}
```

表 6-4 コード指定によるリクエスト内容

Property	Type	Required	Description	Example
request	object	Yes	リクエストを指定	—
request.esv	string	Yes	ESV を 16 進数の文字列で指定	"0x62" or "0x6E"

request.operations	array	Yes	複号電文を配列で指定可能	—
request.operations.epc	string	Yes	EPC を 16 進数の文字列で指定	"0xF0"
request.operations.edt	array	Yes(Set のみ)	EDT を 16 進数の文字列で指定。配列指定可能	["0x12","0x34", "0x56"]

表 6-5 コード指定によるレスポンス内容

Property	Type	Required	Description	Example
response	object	Yes	レスポンスを指定	—
response.esv	string	Yes	ESV を 16 進数の文字列で指定	"0x72" or "0x7E"
response.operations	array	Yes	複号電文の応答を配列で指定可能	—
response.operations.epc	string	Yes	EPC を 16 進数の文字列で指定	"0xF0"
response.operations.edt	array	Yes	EDT を 16 進数の文字列で指定。配列指定可能。もしくは null	["0x12","0x34","0x56"] or null

複数プロパティを操作対象にする場合の記述例についても下記に示す。

複数プロパティを操作対象にする場合の例

■ リクエスト

```
POST /elapi/v1/devices/<device id>/echoCommands HTTP/1.1
{
    "request": {
        "esv": "0x62",
        "operations": [
            {
                "epc": "0xF0",
            },
            {
                "epc": "0xF2"
            }
        ]
    }
}
```

■ レスポンス

```
{
    "response": {
        "esv": "0x72",
        "operations": [
            {
                "epc": "0xF0",
                "edt": [
                    "0x12",
                    "0x34",
                    "0x56"
                ],
```

```
            },
            {
                "epc": "0xF2",
                "edt": [
                    "0x42"
                ]
            }
        ]
    }
}
```

6．6 Action

action（アクション）を呼び出すには、対象アクションリソース名を含む URI に対して POST メソッドを実行する。アクションは、機器の操作やサービス操作のうち、プロパティリソースを定義して行う操作として定義しづらい場面などで使用される。

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

■ リクエスト

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

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

■ レスポンス

```
HTTP/1.1 200 OK

<output data>
```

6．7 エラー処理

HTTP では、レスポンス時のステータスコードを表 6-6 のように規定している。これらを考慮し、サーバは適切にエラーコードなどを返却すべきである。

表 6-6 HTTP Status Code

Status Code	名称	意味
100 番台	Informational	処理が継続中

200 番台	Successful	正常終了
300 番台	Redirection	転送の要求
400 番台	Client Error	クライアント原因のエラー
500 番台	Server Error	サーバ（宅内コントローラや機器との通信含む）原因のエラー

より詳細なエラー情報を返却できる場合には、表 6-7 を参考に適切なステータスコードを返却しても良い（オプション）。

表 6-7 HTTP Status Code の活用事例

Status Code	名称	意味（例）
200	OK	PUT、GET 時の成功
201	Created	POST 時の成功
204	No Content	PUT、DELETE 時の成功
301	Moved Permanently	リソースは恒久的に移動
304	Not Modified	リソースは未更新
400	Bad Request	リクエストが不正、データ形式間違い
401	Unauthorized	認証が必要
404	Not Found	リソース（該当するパスやプロパティ）が存在しない。
405	Method Not Allowed	リソースに指定されたメソッドが許可されない
406	Not Acceptable	Accept ヘッダとマッチしない
409	Conflict	リソースが矛盾（ID が衝突など）
415	Unsupported Media Type	データ形式は正しいがサーバが対応しない
500	Internal Server Error	サーバサイドでエラーが発生
503	Service Unavailable	サーバが一時的に停止

本ガイドラインでは、上記ステータスコードが 4XX 系、5XX 系の場合は更に、必要に応じて ECHONET Lite Web API の呼び出しや、サービスに起因するエラー応答もレスポンスボディ（JSON 形式）にて返却可能とする（オプション）。
下記に示す通り、エラータイプとメッセージ（任意の文字列）を組として応答する。

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

表 6-8 サービスレベルのエラー応答

Property	Type	Required	Description	Example
type	string	Yes	Error の Type を示す。サーバが Error と判断する場合（rangeError/referenceError/typeError/timeoutError）	"rangeError"

			と機器がエラーと判断する場合（deviceError）がある。	
message	string	Yes	ERROR の詳細を記述する任意の String data	

type には、下記種類を想定する。

表 6-9 エラータイプの種類

ErrorType	Description	Example
クライアント（クライアント）に起因		
rangeError	設定する値が仕様の範囲外の場合	number, integer：値が min と max の間にない場合 enum: 値が存在しない場合
referenceError	対象とする device ID や property resource name が存在しない場合	
typeError	設定する値の型が Device Description に記述された型と一致しない場合	
サーバ（コントローラや機器の通信含む）に起因		
timeoutError	機器から一定時間内に返答がない場合	通信系エラー
deviceError	機器から受信したデータが error に対応する値の場合。機器から SNA を受信した場合	SetGet_SNA, Set_SNA を受信した場合

例

```
{
    "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 機器情報

「機器仕様部」（別紙）にて記載する。

第７章 応用サービス機能

これまで紹介してきた仕様は、主に ECHONET Lite プロトコル自身が有する基本機能を Web サービスへマッピングした際の展開モデルが中心だった。本章では、こうした基本機能の組み合わせや別の情報の追加・データ加工などによって、より便利な応用サービスを実現するための事例について紹介していく。

なお、以降、リソースを表すパスのうち、接頭語となる ”/elapi/v1” については記載を省略する。

７．１ 複数命令の一括指示（bulks）

サーバは、任意のリソースを対象とした命令を複数列挙した命令セット（bulk）をクライアントからの要求に基づき作成し登録することで、一括で複数命令を指示・実行する機能（bulk 実行）をクライアントに対して提供する。

図 7-1 を用いて、bulk 実行に関する一連の動作について例示する。

- bulk 実行のためには、まず、①クライアントが、対象とする命令（method, path, [body]（body は省略可能）から構成）を複数列挙した命令セット（bulk）をサーバへ登録申請する。サーバは、②同 bulk を登録後、③クライアントへ同 bulk の識別子となる bulk ID を返却する。
- 次に、④クライアントが同 bulk ID を指定して execute アクションを指示することで、⑤サーバにて同 bulk 実行の有効期間が開始となり、⑥サーバはクライアントへ execution ID を返却する)。
- 以降、⑦サーバは、宅内・構内のコントローラ（機器含む）へ bulk を構成する各命令を送信し実行結果の取得を進める（⑧⑨)。
- ⑩クライアントが execution ID を指定してサーバに対して bulk 実行のレスポンス取得を試みる（⑪にてサーバは応答する)。クライアントは、この操作を必要に応じて bulk 実行が完了するまで複数回実施する。
- ⑫最後の命令実行が完了した後、⑬サーバでの bulk 実行は完了する。以降、⑭クライアントが bulk 実行のレスポンス取得を試みたとき、取得したレスポンス内の"processStatus"は終了を示す値となる（⑮)。ただし、サーバ規定による bulk 実行有効期間を超えてクライアントが同レスポンス取得を試みた場合は、サーバよりエラーが返される。
- この bulk 実行有効期間は、サーバにより自由に規定可能とするが、通常は、bulk 実行が完了し、クライアントによる getResults アクションを用いた結果取得が余裕をもって実施できる十分な期間が設定されることが望ましい。
- なお、bulk ID は再利用可能なため、再度 bulk 実行を行いたい場合は、④以降の手順を繰り返し（新たな execution ID を獲得し）実行することができる。
- 最終的に、登録した bulk が不要となった場合は、bulk ID を指定してサーバ登録された bulk を消去する（⑰⑱⑲)。

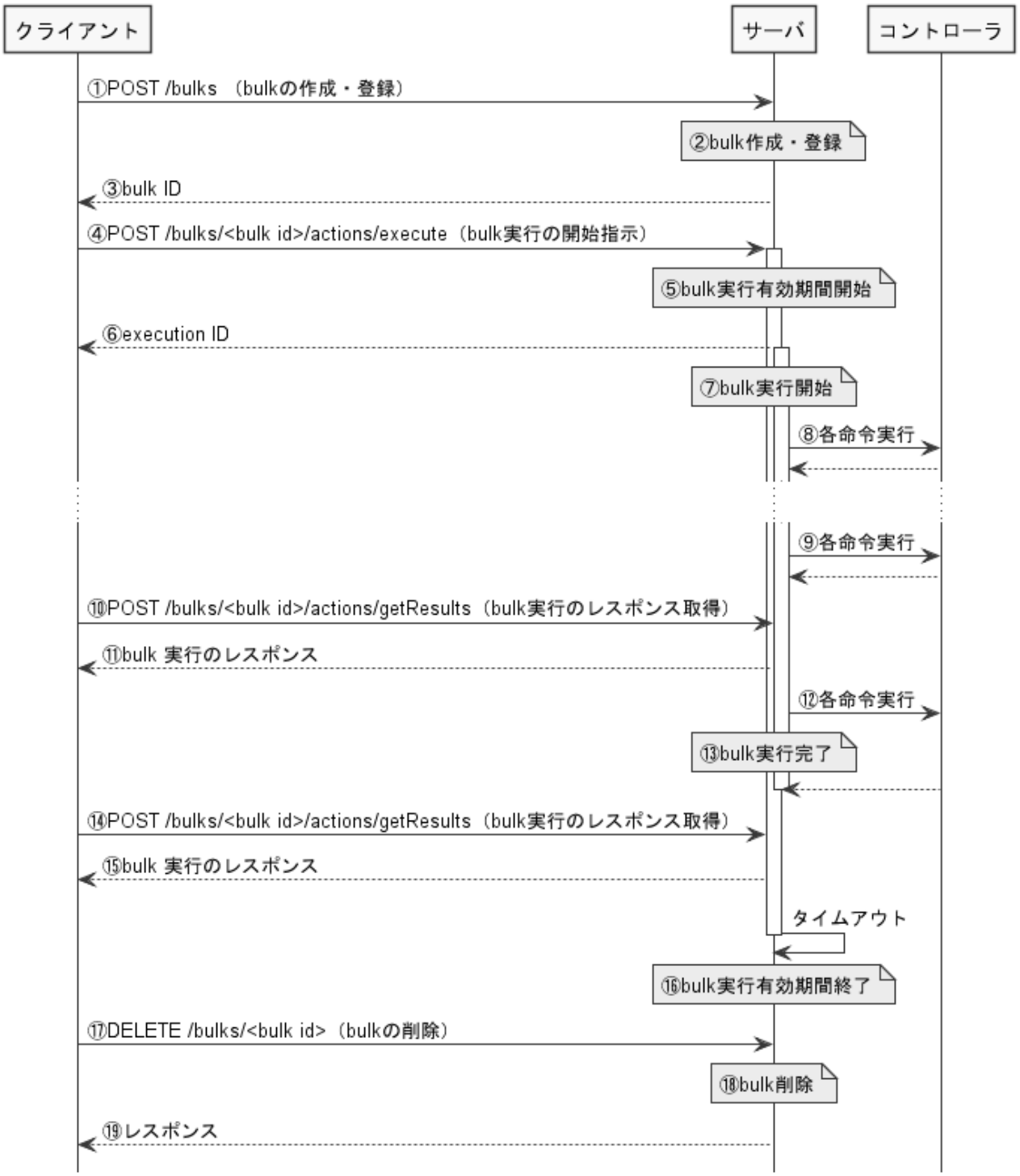

図 7-1　bulk 実行の全体概要

サーバが宅内・構内のコントローラ（機器含む）へ命令実行する際、bulk の処理モード（"processMode"）は二種類のうちいずれかを指定することができる（図 7-2）。①の bulk 登録時に"concurrent"モードを指定した場合は、"requests"キー内の配列にて指定した順に命令を（応答を待たずに）実行し、"sequential"モードを指定した場合は、"requests"キー内の配列にて指定した順に

命令を（正常応答を待ちつつ）実行する。

"processMode"無指定の場合は、"concurrent"モードにて実行される。

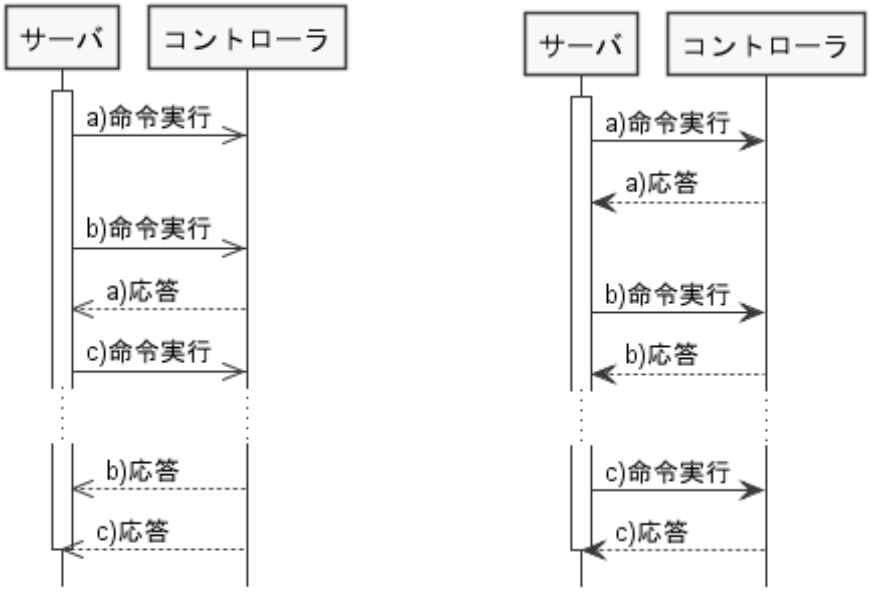

図 7-2 concurrent モード（非同期）と sequential モード（同期）

⑪における bulk 実行のレスポンス内容より、bulk 実行状況について個別および全体の視点から確認することができる。

個別状況は、"responses"キー内の配列に列挙される各命令について（"requests"キーで指定した配列の順にて）、各命令の実行状況を"progress"キーによって確認できる。

値は、unexecuted（未実行。初期値）、executing（実行中）、succeeded（成功完了）、failed（失敗完了）、timedout（タイムアウト）、aborted（中断）のいずれかの値となる。

"procssMode"が"concurrent"の場合は、クライアントから明示的に abort されるか、サーバ側での処理時間が（サーバ規定の）全体タイムアウト値を超えない限り、全ての命令が実行される。

図 7-3 に"concurrent"モードでの処理について例示する。図では、命令 a) から e) を含んだ bulk 内の各命令を順次実行中であり、a) は成功、b) は失敗に終わり、c) の実行でタイムアウト、d) は実行中で、e) は未実行の状態であることを示している（クライアントが呼び出した時点は d) 実行中の位置）。

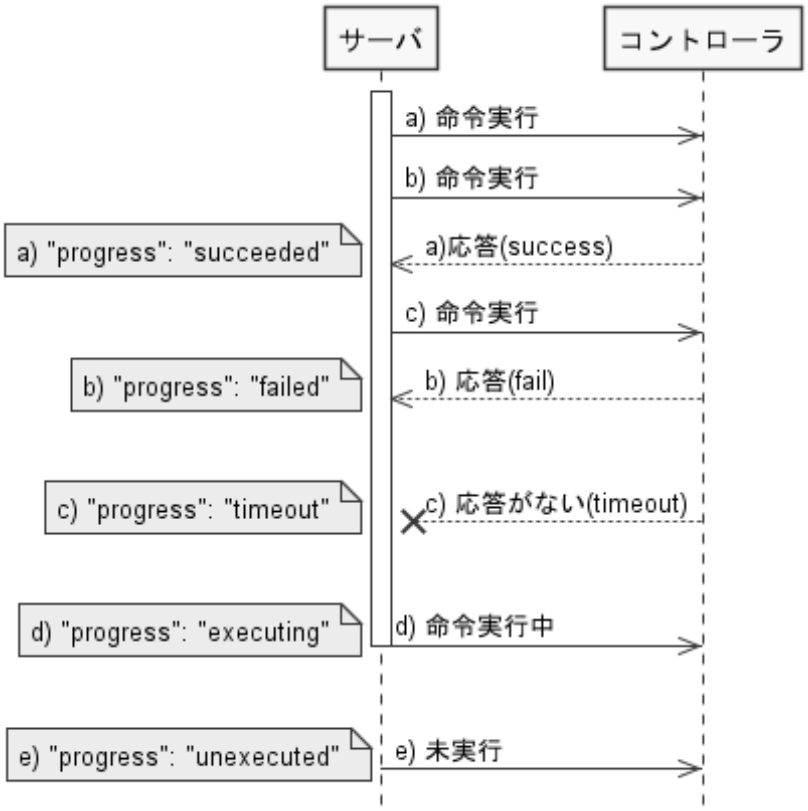

図 7-3 concurrent モード : d) まで実行中

"procssMode"が"sequential"の場合は、各命令のサーバ実行が失敗するか（failed もしくは timeout）、クライアントから明示的に abort されるか、サーバ側での処理時間が（サーバ規定の）全体タイムアウト値を超えない限り、全ての命令が実行される。すなわち、正常応答（succeeded）が続く限り、全ての命令が実行される。

図 7-4 に"sequential"モードでの処理について例示する。図では、命令 a) から e) を含んだ bulk の各命令を順次実行中であり、a) b) は成功し、c) 以降は未実行の状態を示している（クライアントが呼び出した時点は b)直後の位置）。

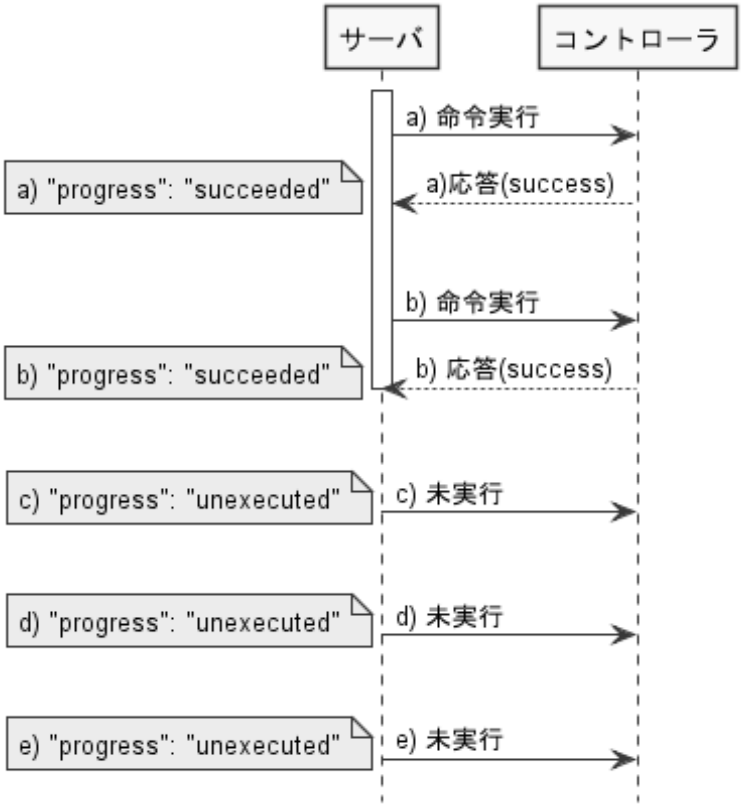

図 7-4 sequential モード : b) まで実行中

続いて、図 7-5 では、c) が失敗に終わったため、d) e) は未実行から中断状態へ移行したことを示している。c) で失敗しているため、以降の命令実行は行われない。

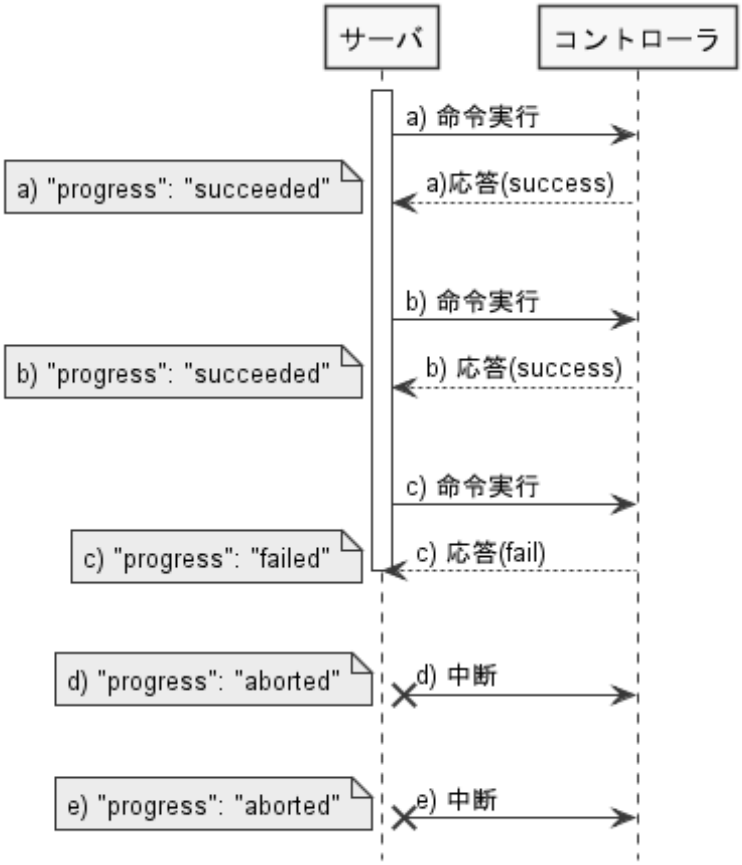

図 7-5 sequential モード : c) 実行後停止

"processMode"はいずれの場合も、クライアントから abort が送信された場合は、サーバ上にて指定された execution ID に対応する残存の実行処理を中断し、"progress"が実行中（executing）および未実行（unexecuted）の命令は中断（aborted）へ移行する。bulk 実行有効期間は、前述の通りサーバ上で一定時間後に自動的に削除されない限り継続するため、クライアントは中断時の状況について execution ID を伴う getResults アクションにて取得することが可能である。

⑪のレスポンス内に列挙される各命令の"progress"を全て確認すれば、全体の動作状況を把握することは可能だが、対象命令数が多数となる場合、クライアントの状況確認処理が煩雑となってしまう。クライアントが簡易に bulk 実行に関する全体の進行状況を把握できるよう、"processStatus"が提供されている。

"processStatus"は、"inProgress"(実行中), "succeeded"(すべて成功にて実行完了), "failed"(1 つ以上の失敗またはタイムアウトにて実行完了), "aborted"(1 つでも中断がある場合。"failed"より優先)のいずれかの値をとる。

全体の進行状況に応じた"processStatus"の値については下記の通りとなる。

- 全ての命令が未実行または 1 つ以上の命令が実行中の場合 : "inProgress"
- bulk 実行途中でクライアントから abort 発行にて中断された場合、サーバシステム全体でのタイムアウト値を超えて中断された場合、"processMode"が"sequential"モードにて"failed"または"timeout"が発生した場合のいずれか : "aborted"
- 全ての命令実行が完了した場合 :
  - ➢ 全ての実行が成功→"succeeded"

➢ 1 つ以上の実行が失敗またはタイムアウトした場合→"failed"

さらに、bulk の各命令実行状態("progress")を確認することができる。実行状況として、"unexecuted"（未実行）,"executing"（実行中）,"succeeded"（成功完了）,"failed"（失敗完了）,"timeout"（タイムアウト）,"aborted"（中断）を用意している。

タイムアウト値の設定は、サーバにて適時規定されていることを前提する。各命令の実行タイムアウト値 t_i（i は 1 から total）や、全命令の実行タイムアウト値 T（bulk 実行有効期間）については、必ずしも各実行タイムアウト値の総和（$\sum_{i=1}^{n} ti$）以上である必要はないが、前述の通り、十分余裕を持った期間に設定されることが望ましい。

表 7-1 に、bulk に関する API 一覧を示す。以降、個別に説明する。

表 7-1 複数命令の一括指示に関する API

http method	path	description
POST	/bulks	bulk 作成
GET	/bulks	bulk ID の一覧取得
GET	/bulks/<bulk id>	bulk description 取得
GET	/bulks/<bulk id>/properties	bulk の全プロパティリソース値の取得
GET	/bulks/<bulk id>/properties/<property resource name>	bulk のプロパティリソース値の取得
PUT	/bulks/<bulk id>/properties/<property resource name>	bulk のプロパティリソース値の設定
POST	/bulks/<bulk id>/actions/execute	bulk 実行開始
POST	/bulks/<bulk id>/actions/abort	bulk 実行中断
POST	/bulks/<bulk id>/actions/getResults	bulk 実行結果の取得
DELETE	/bulks/<bulk id>	bulk 削除

POST /bulks

bulk を作成・登録し、識別子（bulk ID）を生成しクライアントへ返却する。

対象リソース（命令）の共通 URI 部を"base"（省略可能）にて指定し、bulk 処理モード（"processMode"）の指定や、各命令（"requests"の配列形式）を列挙してクライアントからサーバへ要求する。"processMode"は省略可能であり、省略時は、"concurrent"が適用される。各命令は、"responses"キー内の配列に列挙される各命令"method"(HTTP リクエストメソッド)、"path"("base"からの相対リソース位置。"base "が無指定の場合は絶対リソース位置), ["body"(引数となる HTTP リクエストボディ。省略可)]の組から構成される。サーバ上での bulk 作成・登録が成功すると、サーバからクライアントへ HTTP ステータスコード 201 と HTTP ボディとして bulk ID が返却される。サーバにて登録可能な最大 bulk 数を超える場合は、HTTP ステータスコード 400 と HTTP ボディとして{ "type": "rangeError", "message": "You can't create bulks over the registration limit" }が返却される。

■リクエスト定義

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

■レスポンス定義

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

■リクエスト例

```
{
    "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}}
    ]
}
```

■レスポンス例

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

リクエスト：

Property	Type	Required	Description
descriptions	object	Yes	登録する bulk に関する記述
descriptions.ja	string	Yes	登録する bulk の内容説明

descriptions.en	string	Yes	the content of a bulk to register in English
base	string	No	操作対象となるリソースへの共通ベース URI（省略可）
processMode	string	No	"concurrent" または"sequential"。省略時には"concurrent"モードとなる。前者は requests 内で登録する命令セットを並列実行し、後者は逐次実行する。並列実行では、各命令を記述順に順次実行するが、各応答は待たずに実行される。逐次実行では、命令の正常応答を待ってから、次の命令を実行する
requests	array	Yes	各命令を配列にて列挙
requests[].method	string	Yes	HTTP リクエストメソッド
requests[].path	string	Yes	操作対象とするリソースへのパス。base が指定された場合は、相対パスにて指定し、省略された場合は絶対パス（URI）にて指定
requests[].body	object	No	操作時に必要なリクエストボディ

レスポンス：

Property	Type	Required	Description
id	string	Yes	bulk ID

GET /bulks

builk ID の一覧取得。サーバに登録された bulk ID が配列形式にて返却される。登録が無い場合は、空の配列が返却される。

■レスポンス定義

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

■レスポンス例

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


レスポンス：

Property	Type	Required	Description
registrationLimit	number	No	登録可能な bulk の最大数
bulks	array	Yes	登録済の bulk ID 等を配列にて列挙。登録が無い場合は空（"bulks": []）
bulks[].id	string	No	bulk ID
bulks[].descriptions	object	No	登録済の bulk に関する記述
bulks[].descriptions.ja	string	No	登録済の bulk の内容説明
bulks[].descriptions.en	string	No	the content of the registered bulk in English

GET /bulks/<bulk id>

bulk ID で指定された bulk description を取得する。

■レスポンス例


```
{
    "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": {
                            "type": [
                                "string",
                                "number",
                                "object",
                                "array",
                                "boolean",
                                "null"
                            ]
                        }
                    }
                }
            }
        }
    },
    "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": {
                            "type": [
                                "string",
                                "number",
                                "object",
                                "array",
                                "boolean",
                                "null"
                            ]
                        },
                        "progress": {
                            "type": "string",
                            "enum": [
                                "completed",
                                "unexecuted",
                                "processing",
                                "aborted"
                            ]
                        },
                        "statusCode": {
                            "type": "number"
                        }
                    }
                }
            }
        }
    }
}
```


レスポンス：

Property	Type	Required	Description
properties	object	Yes	—
properties.descriptions	object	Yes	登録済の bulk に関する記述
properties.descriptions.ja	string	Yes	登録済の bulk の内容説明
properties.descriptions.en	string	Yes	the content of the registerd bulk in English
properties.base	string	No	操作対象となるリソースへの共通ベース URI（bulk 生成時に無指定の場合は、省略）

properties.processMode	string	Yes	concurrent"または"sequential"。bulk 生成時に無指定の場合は"concurrent"モードとなる。前者は requests 内で登録する命令セットを並列実行し、後者は逐次実行する。並列実行では、各命令を記述順に順次実行するが、各応答は待たずに実行される。逐次実行では、命令の正常応答を待ってから、次の命令を実行する
properties.requests	array	Yes	各命令を配列にて列挙
properties.requests[].method	string	Yes	HTTP リクエストメソッド
properties.requests[].path	string	Yes	操作対象とするリソースへのパス。base 以下のパス。base が存在する場合は相対パス、省略された場合は絶対パス（URI）
properties.requests[].body	object	No	操作時に必要なリクエストボディ。requests[].method が GET の場合は省略される
actions	object	Yes	アクションに関する記述
actions.execute	object	Yes	bulk 実行開始命令
actions.abort	object	Yes	bulk 実行中断命令
actions.getResults	object	Yes	bulk 実行結果の取得

GET /bulks/<bulk id>/properties

bulk ID で指定された全プロパティリソースの値を取得する。

■レスポンス定義

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

■レスポンス例

```
{
    "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}}
    ]
}
```

レスポンス：

Property	Type	Required	Description
descriptions	object	Yes	登録済の bulk に関する記述
descriptions.ja	string	Yes	登録済の bulk の内容説明
descriptions.en	string	Yes	the content of the registerd bulk in English
base	string	No	操作対象となるリソースへの共通ベース URI（bulk 生成時に無指定の場合は、省略）
processMode	string	Yes	concurrent"または"sequential"。bulk 生成時に無指定の場合は"concurrent"モードとなる。前者は requests 内で登録する命令セットを並列実行し、後者は逐次実行する。並列実行では、各命令を記述順に順次実行するが、各応答は待たずに実行される。逐次実行では、命令の正常応答を待ってから、次の命令を実行する
requests	array	Yes	各命令を配列にて列挙
requests[].method	string	Yes	HTTP リクエストメソッド
requests[].path	string	Yes	操作対象とするリソースへのパス。base 以下のパス。base が存在する場合は相対パス、省略された場合は絶対パス（URI）
requests[].body	object	No	操作時に必要なリクエストボディ。requests[].method が GET など、不要な場合は省略される

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

bulk ID にて指定された bulk のプロパティリソース値の取得。

■リクエスト例

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

■レスポンス定義

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

■レスポンス例

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

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

bulk ID にて指定された bulk のプロパティリソース値の設定。下記例では、descriptions 部の書き換えが可能なことを示している。

■リクエスト定義およびレスポンス定義

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

■リクエスト例およびレスポンス例

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

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

bulk 実行の開始。リクエスト時、ボディの指定は不要。bulk の実行では、全てのリクエストの実行が完了するまでに時間がかかることを想定し、実行結果については非同期で取得するモデルを採用している。具体的には、クライアントは POST actions/execute で bulk 実行の開始を指示し execution ID を取得し、この ID を指定して POST actions/getResults を呼び出すことで結果を取得する。

execution ID は temporary な id のため、検索や削除の手段は提供していない。サーバが一定時間後に削除する実装とする（execution ID の有効期限はサーバ実装依存とする）。processStatus 完了なら次受付。でなければエラー。

■レスポンス定義

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

■レスポンス例

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

レスポンス：

Property	Type	Required	Description
executionId	string	Yes	実行時に割り当てられる execution ID

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

指定した execution ID に対応する bulk 実行のレスポンス取得。

■リクエスト定義

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

■リクエスト例

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

■レスポンス定義：

```
{
    "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>},
        ...
    ]
}
```

■レスポンス例：


```
{
    "base": "https://xxx.xxx/elapi/v1/",
    "total": 3,
    "processStatus": "inProgress",
    "responses": [
        {
            "method": "GET",
            "path": "devices/0123/properties/operationStatus",
            "body": {"operationStatus": true},
            "progress": "succeeded",
            "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"
        }
    ]
}
```


レスポンス：

Property	Type	Required	Description
base	string	No	操作対象となるリソースへの共通ベース URI (bulk 生成時に無指定の場合は、省略)
total	number	Yes	実行命令のトータル数
processStatus	string	Yes	全体の進行状況。 "inProgress"(実行中), "succeeded"(すべて成功にて実行完了), "failed"(1 つ以上の失敗またはタイムアウトにて実行完了), "aborted"(1 つでも中断がある場合。"failed"より優先)のいずれかの値
responses	array	Yes	各命令の応答を配列にて列挙
responses[].method	string	Yes	HTTP リクエストメソッド
responses[].path	string	Yes	操作対象とするリソースへのパス。base 以下のパ

			ス。base が存在する場合は相対パス、省略された場合は絶対パス（URI）
responses[].body	object	No	操作実行成功後には応答（レスポンスボディ）。実行エラー時にはエラー詳細を記述する場合は"error"と"message"のオブジェクト形式（オプション）。また、操作未実行時や実行中の場合はリクエストボディと同じ内容（requests[].method が GET で、未実行・実行中の場合は省略される）
responses[].progress	string	Yes	操作の実行状況。操作未実行時は"unexecuted"、実行中は"executing"、操作実行後、成功完了時には "completed"、失敗完了時には"failed"、タイムアウト時には"timeout"、実行中断は"aborted"のいずれかの値
responses[].status	string	No	操作実行後の応答（ステータスコード）。操作未実行時、実行中は省略される

DELETE /bulks/<bulk id>

登録済みの bulk の削除。bulk ID には、削除対象となる bulk ID を指定する。
Body は含まず、HTTP ステータスコード 204(No Content)のみ返す。
既に削除済みもしくは存在しない bulk ID が指定された場合は、404(Not Found)を返す。

7．2 機器のグルーピング（groups）

サーバは、クライアントからの要求に基づき、複数の機器をまとめてグループ化し登録することで、リソースの分類・整理や対象機器群に対する個別・共通命令などを実行可能にする機能（group 操作機能）をクライアントに対して提供する。

図 7-6 を用いて、group 操作に関する一連の動作について例示する。

- group 操作のためには、まず、①クライアントが、グループ化対象となる機器を複数列挙した機器リスト（group）をサーバへ登録申請する。サーバは、②同 group を登録後、③クライアントへ同 group の識別子となる group ID を返却する。
- 次に、④⑦⑫のうちいずれかを選択し、所望の group 操作を行うことができる。
- ④では、クライアントから同 group ID を指定してグループ化された機器への全プロパティリソース値の取得、⑦では、指定プロパティリソースに対する値の取得、⑫では、指定プロパティリソースに対する値の設定が可能となる。
- 最終的に、登録した group が不要となった場合は、group ID を指定してサーバ登録された group を消去する（⑰⑱⑲）。

デフォルトでは、④⑦⑫のようなプロパティリソース操作を実施する際、対象機器が当該プロパティを有し実行可能なメソッドであれば実施され、当該プロパティを有しない場合やメソッドに対応し

ていない場合は HTTP ステータスコード 404 (Not Found) や 405 (Method Not Allowed) を伴い、エラーメッセージを返却する。プロパティ操作が実行された場合であっても、実機からの応答時間がサーバ規定のタイムアウト値を超える場合は、HTTP ステータスコード 500 (Internal Server Error) を伴い、エラーメッセージ（"type": "timeoutError"）を返却する。

生成するグループの特質によっては、上記のようなエラーケースで 400 系、500 系コードやエラーメッセージの返却をクライアントが不要としたい場合がある。例えば、1 台以上の蓄電池クラスと 1 台以上の住宅用太陽光発電クラスを組み合わせたグループを作成し、機器全体を仮想的な混合デバイス（以降、「仮想混合デバイス」と呼ぶ）として扱いたい場合、2 つのクラス共通のプロパティを操作する場合は、全ての機器に対して操作を実施し、一方のクラスのみが搭載するプロパティを操作する場合は、不要な他方のクラス（機器）へ操作を行わない（エラー処理を省略する）処理を期待するケースが考えられる。オプション機能として、こうした処理を可能にする仕組みを提供する。

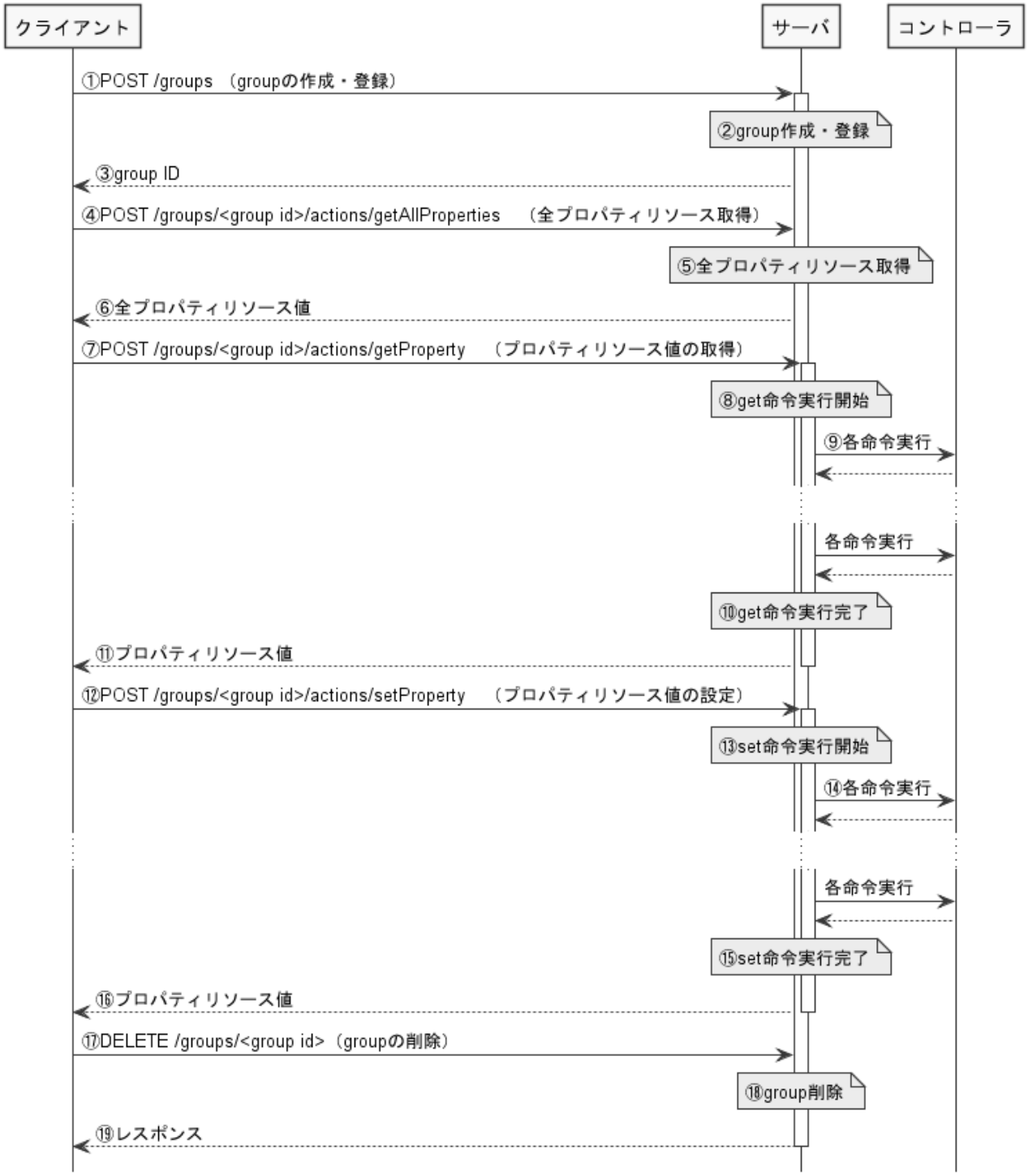

図 7-6 group 実行の全体概要

なお、明示的に操作対象となるクラスに対応した機器を指定したい場合は、下記のような指定形式も可能とする（オプション）。

例）仮想混合デバイスのうち一般照明クラス（generalLighting）対応機器のみを対象とした操作

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

また、一旦グループを作成した後に、グループ化対象となる機器の追加・削除機能も提供する。
表 7-2 に、group に関する API 一覧を示す。以降、個別に説明する。

表 7-2 機器のグルーピングに関する API

http method	path	description
POST	/groups	group 作成
GET	/groups	group ID の一覧取得
GET	/groups /<group id>	group description 取得
GET	/groups /<group id>/properties	group の全プロパティリソース値の取得
GET	/groups/<group id>/properties/<property resource name>	group のプロパティリソース値の取得
PUT	/groups/<group id>/properties/<property resource name>	group のプロパティリソース値の設定
POST	/groups /<group id>/actions/getAllProperties	対象機器群の全プロパティリソース値の取得
POST	/groups /<group id>/actions/getProperty	対象機器群のプロパティ値の取得
POST	/groups /<group id>/actions/setProperty	対象機器群のプロパティ値の設定
DELETE	/groups /<group id>	group 削除

POST /groups

グループ化したい対象機器を保持するグループ（group）を作成・登録し、識別子（group ID）を生成しクライアントへ返却する。
対象機器は device ID の配列形式にて列挙してクライアントからサーバへ要求する。サーバにて登録可能な最大 group 数を超える場合は、HTTP ステータスコード 400 と HTTP ボディとして
`{ "type": "rangeError", "message": "You can't create groups over the registration limit" }`が返却される。
以降、デフォルトでは、グループ化された機器全てに対して、一括で処理が可能となる（オプションである"composed"を false 指定した場合も同様）。
仮想混合デバイスがサポートされている場合は、"composed"に true を指定することで、以降、グループ対象機器群のプロパティに対する操作は、当該プロパティを有し実行可能なメソッドに対応している機器に対してのみ実行され、そうでない機器に対しては当該操作が実行されず、レスポンスも返却されない（エラーコードやエラーメッセージを返さない）。
"composed"がサポートされていないサーバに対して、"composed"に true を指定してリクエストした場合、group ID は返却されず、HTTP ステータスコード 404（Not Found）を伴い、エラーメッセージ（"type": "typeError"）が返却される。

■リクエスト定義

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

■リクエスト例

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

■レスポンス定義

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

■レスポンス例

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

リクエスト：

Property	Type	Required	Description
descriptions	object	Yes	登録する group に関する記述
descriptions.ja	string	Yes	登録する group の内容説明
descriptions.en	string	Yes	the content of a group to register in English
members	array	Yes	対象機器群。device ID を配列にて列挙
members[].deviceId	string	Yes	対象機器の device ID
composed	boolean	No	仮想混合デバイスのサポートの有無。省略された場合は false となる。

レスポンス：

Property	Type	Required	Description
id	string	Yes	group ID

GET /groups

group ID の一覧取得。サーバに登録された group ID が配列形式にて返却される。登録が無い場合は、空の配列が返却される。

■レスポンス定義

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

■レスポンス例

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

レスポンス：

Property	Type	Required	Description
registrationLimit	number	No	登録可能な group の最大数
groups	array	Yes	登録済の group ID 等を配列にて列挙。登録が無い場合は空（"groups": []）
groups[].id	string	No	group ID
groups[].descriptions	object	No *1	登録済の group に関する記述
groups[].descriptions.ja	string	No *1	登録済の group の内容説明
groups[].descriptions.en	string	No *1	the content of the registered group in English

*1) groups[].id が存在する場合に必須

GET /groups/<group id>

group ID で指定された group description を取得する。

■レスポンス例

```
{
    "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"
            }
        }
    },
    "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": {
                        "descriptions": {
                            "ja": "プロパティ名",
                            "en": "property  name"
                        },
                        "type": "string"
                    }
                }
            },
            "schema": {
                "type": "object",
                "properties": {
                    "responses": {
                        "type": "array",
                        "items": {
                            "type": "object",
                            "properties": {
                                "deviceId": {
                                    "type": "string"
                                },
                                "body": {
                                    "type": "object"
                                },
                                "statusCode": {
                                    "descriptions": {
                                    "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": {
                        "type": [
                            "string",
                            "number",
                            "object",
                            "array",
                            "boolean",
                            "null"
                        ]
                    }
                }
            },
            "schema": {
                "type": "object",
                "properties": {
                    "responses": {
                        "type": "array",
                        "items": {
                            "type": "object",
                            "properties": {
                                "deviceId": {
                                    "type": "string"
                                },
                                "body": {
                                    "type": "object"
                                },
                                "statusCode": {
                                    "type": "number"
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

レスポンス：

Property	Type	Required	Description
properties	object	Yes	—
properties.descriptions	object	Yes	登録済の groups に関する記述
properties.descriptions.ja	string	Yes	登録済の group の内容説明
properties.descriptions.en	string	Yes	the content of the registered group in English
properties.members	array	Yes	対象機器群
properties.members[].deviceId	string	Yes	対象機器の device ID

properties.composed	boolean	Yes	仮想混合デバイスのサポートの有無
actions	object	Yes	—
actions.getAllProperties	object	Yes	対象機器群の全プロパティリソースの GET
actions.getProperty	object	Yes	対象機器群のプロパティリソースの GET
actions.setProperty	object	Yes	対象機器群のプロパティリソースの SET

GET /groups/<group id>/properties

group ID で指定された group の全プロパティリソースの値を取得する。

■レスポンス定義

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

■レスポンス例

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

レスポンス：

Property	Type	Required	Description
descriptions	object	Yes	登録済の groups に関する記述
descriptions.ja	string	Yes	登録済の group の内容説明
descriptions.en	string	Yes	the content of the registered group in English
members	array	Yes	対象機器の device ID を配列にて列挙
members[].deviceId	string	Yes	対象機器の device ID
composed	boolean	Yes	仮想混合デバイスのサポートの有無

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

group ID にて指定された group のプロパティリソース値の取得。

■リクエスト例

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

■レスポンス定義

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

■レスポンス例

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

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

group ID にて指定された group のプロパティリソース値の設定。下記例では、機器リストの更新により対象とする機器群の入れ替えや増減が可能なことを示している。

■リクエスト定義およびレスポンス定義

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

■リクエスト例およびレスポンス例

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

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

group ID にて指定された group が保持する対象機器群への全プロパティリソース値の取得。リクエスト時、ボディの指定は不要。

機器単体に対して<device ID>/properties を指定し GET メソッドを用いて取得される結果と同じ内容が"properties"キーに対応する値にて返却される。

■レスポンス定義

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

■レスポンス例

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

レスポンス：

Property	Type	Required	Description
responses	array	Yes	レスポンス
responses[].deviceId	string	Yes	対象機器の device ID
responses[].properties	object	Yes	対象機器の全プロパティリソース値

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

group ID にて指定された group が保持する機器群へのプロパティリソース値の取得。
機器単体に対して<device ID>/properties/<property resource name>を指定し GET メソッドを用いて取得される結果と同じ内容が"body"キーに対応する値にて返却されると共に、HTTP ステータ

スコードも返却される。エラーが発生する場合は、エラー値が返却されると共に、HTTP ステータスコードも返却される。

"composed"がサポートされている場合（true の場合）は、同 property resource name を有し GET メソッドに対応している場合のみ応答し、そうでない機器に関して操作を実行せず、結果も返却しない。

■リクエスト定義

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

■リクエスト例

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

■レスポンス定義

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

■レスポンス例

```
{
    "responses": [
        {
            "deviceId": "0123",
            "body":
                {
                    "operationStatus": true
                },
            "status": 200
        },
        {
            "deviceId": "1234",
            "body":
                {
                    "type": "timeoutError",
```

```
                "message": "the device does not respond"
            },
        "status": 500
    }
  ]
}
```

レスポンス：

Property	Type	Required	Description
responses	array	Yes	レスポンス
responses[].deviceId	string	Yes	device ID
responses[].body	object	Yes	操作実行成功後には応答（レスポンスボディ）。実行エラー時は"type"(必須)、"message"（必須）
responses[].status	string	Yes	操作実行後の応答（ステータスコード）

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

group ID にて指定された group が保持する機器群へのプロパティリソース値の設定。
機器単体に対して<device ID>/properties/<property resource name>を指定し SET メソッドを用いて取得される結果と同じ内容が"body"キーに対応する値にて返却されると共に、HTTP ステータスコードも返却される。エラーが発生する場合は、エラー値が返却されると共に、HTTP ステータスコードも返却される。
"composed"がサポートされている場合（true の場合）は、同 property resource name を有し PUT メソッドに対応している場合のみ応答し、そうでない機器に関して操作を実行せず、結果も返却しない。

■リクエスト定義

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

■リクエスト例

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

■レスポンス定義

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

■レスポンス例

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

レスポンス：

Property	Type	Required	Description
responses	array	Yes	レスポンス
responses[].deviceId	string	Yes	device ID
responses[].body	object	Yes	操作実行成功後には応答（レスポンスボディ）。実行エラー時は"type"(必須)、"message"（必須）
responses[].status	string	Yes	操作実行後の応答（ステータスコード）

DELETE /groups/<group id>

登録済みの group の削除。group ID には、削除対象となる group ID を指定する。
ボディは含まず、HTTP ステータスコード 204(No Content)のみ返す。
既に削除済みもしくは存在しない group ID が指定された場合は、404(Not Found)を返す。

7．3 履歴データ（histories）

サーバは、対象機器の各リソースに関する取得値を取得時刻とともに履歴データとして蓄積し、クライアントからの要求に基づき必要な履歴データをクライアントに対して提供する。

履歴データは、取得リソース値と取得時刻の組から構成される。サーバは、履歴データ保存対象となる機器や履歴データの記録タイミング・記録期間などを含んだ履歴データ属性（history）に関する管理を自ら行う。今回のバージョンでは、クライアントから履歴データ保存対象となる機器や取得リソースをサーバへ指定・登録する手段は提供していない。クライアントに対しては、サーバが独自に規定・保存した対象や期間に基づき提供される履歴データセットについて検索手段を提供するのみとする。履歴データセットは、ある history ID に基づきサーバ上で随時保存される全期間の履歴データ集合となる。

図 7-7 を用いて、history 操作に関する一連の動作について例示する。

- クライアントはサーバに対して、①history ID の一覧を取得要求可能であり、②サーバから得られた history ID 一覧の中から③特定の history ID を指定して同 ID に紐付く履歴データの定義情報（history description）を取得できる（④）。
- 続いて、クライアントは⑤パス上にて history ID に続き"properties"を指定することで、サーバから同 ID に紐付く history の全リソースプロパティ値を取得できる（⑥）。
- クライアントはサーバに対して、⑦history ID を指定して（必要に応じて期間や件数を含めた）prepareRetrieveData アクションを実行することで、サーバが提供する履歴データセットの中からクライアント所望の履歴データサブセットを返却できるよう準備開始を指示する。⑧サーバは、同準備が完了すると、このアクションに紐づく data ID に加え、範囲確定された履歴データの総件数や 1 応答（ページ）あたりの件数をクライアントへ返却する（⑨）。
- 以降、⑩クライアントは data ID とページ番号を指定し retrieve アクションを実行することで、サーバより所望の履歴データサブセットを取得する（⑪）。
- 対象となる履歴データの総件数が 1 応答あたりの件数よりも多い場合は、履歴データサブセットは分割されるため、⑫2 ページ目以降の番号を指定することで後続の履歴データサブセットを取得でき（⑬）、⑭終了ページ番号目（（総件数）／（1 応答あたりの件数）：小数点以下切り上げ）にて最終となる履歴データサブセットを取得できる。
- 履歴データサブセットの有効期間は、サーバにより自由に規定可能とするが、通常は、クライアントによる retrieve アクションを用いた結果取得が余裕をもって実施できる十分な期間が設定されることが望ましい。

図 7-7 histories 実行の全体概要

図 7-8 に、サーバ規定による履歴データセットと、クライアント指定による履歴データサブセットとの関係について説明する。図は、8 時から記録を開始し、30 分毎に履歴データの保存を継続中で、現在時刻は 11 時 45 分頃を示している。履歴データセットは 8 時から 11 時 30 分までの履歴データ集合である。これに対し、9 時から 10 時までの履歴データ集合は、クライアントから指定された履歴データサブセットとなる。クライアントから指定される期間の内容によっては、履歴データセットと履歴データサブセットが一致する場合もある。

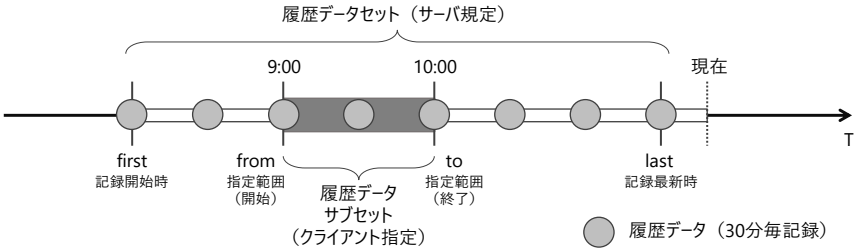

図 7-8 履歴データセット・サブセットの関係

表 7-3 に、histories 関する API 一覧を示す。以降、個別に説明する。

表 7-3 履歴データに関する API

http method	path	description
GET	/histories	history ID の一覧取得
GET	/histories/<history id>	history description 取得
GET	/histories/<history id>/properties	history の全プロパティリソース値の取得
GET	/histories/<history id>/properties/<property resource name>	history のプロパティリソース値の取得
POST	/histories/<history id>/actions/prepareRetrieve	履歴データサブセット取得の準備指示
POST	/histories/<history id>/actions/retrieve	履歴データサブセットの取得

*）なお、history のプロパティリソース値の設定は、history にて書き込み可能なプロパティリソースが存在しないため省略。

GET /histories

履歴 ID の一覧取得。サーバ登録されている履歴 ID が配列形式にて返却される。登録が無い場合は、空の配列が返却される。

■レスポンス定義

```
{
    "histories": [
        {
            "id": <history id>,
            "descriptions": {
                "ja": <description in Japanese>,
```

```
            "en": <description in English>
        }
    },
    ...
]
}
```

■レスポンス例

```
{
    "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 の積算発電電力計測値(kWh)",
                "en": "Cumulative amount of electric energy generated of PV 000045 ([kWh], 30min interval) "
            }
        }
    ]
}
```

レスポンス：

Property	Type	Required	Description
histories	array	Yes	登録済みの history ID 等を配列にて列挙。登録が無い場合は空（"histories": []）
histories[].id	string	No	history ID
histories[].descriptions	object	No *1	登録済の history に関する記述
histories[].descriptions.ja	string	No *1	登録済の history の内容説明
histories[].descriptions.en	string	No *1	the content of the registered history in English

*1) histories[].id が存在する場合に必須

GET /histories/<history id>

history id で指定された history description を取得する。

■レスポンス例


```
{
    "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,
                    "multpleOf": 1
                },
                "desc": {
                    "type": "boolean"
                }
            }
        },
        "schema": {
            "type": "object",
            "properties": {
                "dataId": {
                    "type": "string"
                },
                "count": {
                    "descriptions": {
                        "type": "number",
                        "minimum": 0,
                        "multpleOf": 1
                    },
                    "countPerPage": {
                        "type": "number",
                        "minimum": 1,
                        "multpleOf": 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,
                        "multpleOf": 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": {
                                        "type": [
                                            "string",
                                            "number",
                                            "object",
                                            "array",
                                            "boolean",
                                            "null"
                                        ]
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

レスポンス：

Property	Type	Required	Description
properties	object	Yes	—
properties.descriptions	object	Yes	登録済の history に関する記述
properties.descriptions.ja	string	Yes	登録済の history の内容説明
properties.descriptions.en	string	Yes	the content of the registered history in English
properties.deviceId	string	Yes	履歴対象リソースを保持するデバイスの ID
properties.deviceType	string	No	上記デバイスの種類
properties.resourceType	string	Yes	履歴対象リソースの種類。現時点でのサポートはプロパティリソース "property"のみ （将来的にはアクションリソース "action"、イベントリソース "event"についても検討予定）
properties.resourceName	object	Yes	履歴対象リソース名。履歴対象リソースの種類に対応し、取得対象となる履歴データのリソース名
properties.timing	object	Yes	履歴データの取得タイミング
properties.timing.timingType	string	Yes	履歴データの取得タイミングのタイプ。"onChange", "interval"のいずれか
properties.timing.intervalValue	number	No	時間間隔値。上記 timingType が "interval"の場合、必須
properties.timing.intervalUnit	string	No	時間間隔単位。"sec", "min", "hour", "day", "month", "year" のいずれか。上記 timingType が"interval"の場合、必須
properties.first	string	Yes	履歴データの取得開始時刻。RFC3339 準拠
properties.last	string	Yes	履歴データの取得最新時刻。RFC3339 準拠
properties.total	number	Yes	履歴データセットの総件数
actions	object	Yes	—
actions.prepareRetrieveData	object	Yes	履歴データサブセット取得の準備
actions.retrieve	object	Yes	履歴データサブセットの取得

GET /histories/<history id>/properties

history ID で指定された history の全プロパティリソース値を取得する。各プロパティには、対象となるデバイス (ID や種類) やそのリソース対象 (種類や名称)、履歴データの取得タイミング、取得期間などが含まれる。履歴データの取得タイミングは不定期のタイミングで記録される"onChange"と、一定間隔で記録される"interval"のいずれかが使用され、返却される。後者の場合、その間隔の値と時間単位が共に返却される。履歴データの保存を開始した時刻を"first"、最後に保存した時刻を"last"で返却する。"last"や"total"は、時間経過や履歴データの追加保存により、更新されても良い。

■レスポンス定義

```
{
    "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>
}
```

■レスポンス例 1 ("timingType"が"onChange"の場合)

```
{
    "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
}
```

■レスポンス例 2（"timingType"が"interval"の場合）


```
{
    "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
}
```


レスポンス：

Property	Type	Required	Description
descriptions	object	Yes	登録済の groups に関する記述
descriptions.ja	string	Yes	登録済の group の内容説明
descriptions.en	string	Yes	the content of the registered group in English
deviceId	string	Yes	履歴対象リソースを保持するデバイスのID
deviceType	string	No	上記デバイスの種類
resourceType	string	Yes	履歴対象リソースの種類。現時点でのサポートはプロパティリソース"property"のみ（将来的にはアクションリソース"action"、イベントリソース "event"についても検討予定）
resourceName	string	Yes	履歴対象リソース名。履歴対象リソースの種類に対応し、取得対象となる履歴データのリソース名
timing	object	Yes	履歴データの取得タイミング
timing.timingType	string	Yes	履歴データの取得タイミングのタイプ。"onChange", "interval"のいずれか
timing.intervalValue	number	No	時間間隔値。上記 timingType が"interval"の場合、必須
timing.intervalUnit	string	No	時間間隔単位。"sec", "min", "hour", "day", "month", "year" のいずれか。上記 timingType が"interval"の場合、必須

first	string	Yes	履歴データの取得開始時刻。RFC3339 準拠
last	string	Yes	履歴データの取得最新時刻。RFC3339 準拠。時間経過に伴い更新されても良い。
total	number	Yes	履歴データの総件数。履歴データの追加保存により更新されても良い。

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

history ID にて指定された history のプロパティリソース値の取得。

■リクエスト例

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

■レスポンス定義

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

■レスポンス例

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

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

クライアントは、history ID にて指定される履歴データセットのうち、所望する履歴データの対象期間（開始時刻 "from"、終了時刻 "to"）を指定し、サーバに対して取得対象となる履歴データサブセットの確定を要求する（対象期間はオプション）。サーバは履歴データサブセットを返却できる準備が整ったら、クライアントへ data ID（"dataId"）と総件数（"count"）、1 応答あたりの件数（"countPerPage"）を返却する。リクエストボディを指定しない（省略した）場合は、すべての履歴データセットを対象とする。

"from","to"はオプションであり、指定の仕方は下記の 4 通りある。表中の✓は指定時、－は未指定時を意味する。また、"from"や "to"の指定時は、いずれも同時刻にて記録された履歴データは対象に含む（"first"や"last"も同様）。取得される履歴データは、いずれの指定であっても古い順に並んだ形で返却される。なお、前述の通り、"last"は時間経過と共に更新される場合もあるため、GET /histories/<history id>/properties にて取得した値と異なる可能性もあることに留意すること。

ケース	from	to	対象範囲

1	✓	✓	from から to までの値
2	✓	—	from から last までの値
3	—	✓	first から to までの値
4	—	—	first から last までの値（履歴データセットすべて）

■リクエスト定義

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

■レスポンス定義

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

■リクエスト例（検索開始時刻を指定）

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

■レスポンス例

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

リクエスト：

Property	Type	Required	Description
from	string	No	検索開始時刻。RFC3339 準拠
to	string	No	検索終了時刻。RFC3339 準拠

レスポンス：

Property	Type	Required	Description
dataId	string	Yes	取得対象履歴データサブセット取得用 data ID。実行時に割り当てられる。一定時間（サーバ規定）経過後に自動

			で削除される
count	number	Yes	取得対象履歴データサブセット内の履歴データ総件数。リクエスト時に"count"を指定した場合は、その指定数が上限となる
countPerPage	number	Yes	1 応答あたりに返却可能な履歴データの件数。サーバが一度に応答できる履歴データの最大数（単位）となる。

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

クライアントは、data ID で指定される履歴データサブセットの取得を実行する。

履歴データサブセット内の履歴データの総件数が、1 応答あたりに返却可能な履歴データの件数を上回る場合は、履歴データサブセットは複数の応答（ページ）に分割される。クライアントから履歴データサブセットを取得するには、data ID に加え、ページ番号を指定する必要がある。ただし、1 ページ目についてはページ番号の指定を省略することができる。

レスポンスでは、サーバでの処理状態（"processStatus"）が返却され、成功終了時（"succeeded"）には、これに加え、履歴データのリソース種（"property"、"action"、"event"のいずれか。ただし、現在、プロパティリソース"property"のみサポート）、履歴データのリソース名、履歴データ（"data"）がページ単位にて返却される。履歴データは取得時刻（"time"）と取得値（"value"）から構成される。取得値はリソース種・リソース名に応じて、様々なデータ型をとりうる。

最終ページを超えるページ番号を指定した場合は、HTTP ステータスコード 404(Not Found)を返す。

"processStatus"は、本操作で返却される履歴データのサーバ処理状態を示す。履歴データを含む応答が返却される場合は"succeeded"、サーバ都合で準備中の場合は"inProgress"、サーバ処理が失敗／タイムアウト（値はサーバにより規定）した場合は"failed"を、いずれも HTTP ステータスコード 200 で返す。bulks のケースと異なり、クライアントからの中断指示機能はサポートしていないため、"aborted"は使用しない。

data ID（および対応する履歴データサブセット）は、サーバ規定による一定期間経過後に解放される。

■リクエスト定義

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

■リクエスト例

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

■レスポンス定義

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

■レスポンス例（成功時）

```
{
    "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},
        ...
    ]
}
```

■レスポンス例（サーバ処理中にて待ち状態）

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

■レスポンス例（サーバ処理失敗またはタイムアウト）

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

リクエスト：

Property	Type	Required	Description
dataId	string	Yes	取得対象履歴データサブセット取得用 data ID
page	number	No	全体の進行状況。"inProgress"(サーバ処理中), "succeeded"(サーバ処理成功にて応答), "failed"(サーバ処

			理が失敗またはタイムアウトにて実行終了)のいずれか。

レスポンス：

Property	Type	Required	Description
processStatus	string	Yes	全体の進行状況。 "inProgress"(サーバ処理中), "succeeded"(サーバ処理成功にて応答), "failed"(サーバ処理が失敗またはタイムアウトにて実行終了)のいずれか。
以下は processStatus が"succeeded"の場合のみ使用。Required は使用時の基準となる。			
resourceType	string	Yes	履歴データのリソース種。プロパティリソース "property"（将来的には"action"、"event"についても検討予定）
resourceName	string	Yes	履歴データのリソース名
data	array	Yes	取得対象履歴データサブセット内の履歴データを配列にて列挙。総件数が 0 の場合は空（"data": []）
data[].time	string	No	取得時刻。RFC3339 準拠
data[].value	object	No	取得値。データ欠損している場合は省略される（"time"は省略されない）

７．４ 機器一覧の追加拡張に関する指針

機器一覧取得時の応答内容については、「表 5-2 機器一覧取得時の応答内容」に定義されている内容の変更・削除は行わないことを推奨する。ただし、ベンダ独自に要素を追加することは可能とする。その際の名称は vnd を接頭語とする。

「表 5-2 機器一覧取得時の応答内容」に定義されている内容は、以下のように扱う。

deviceType

独自に定義可能とする。ただし、vnd を接頭語とする。ECHONET 規格の APPENDIX ECHONET 機器オブジェクト詳細規定で定義されている機器でも、「機器仕様部」（別書）で定義されていない機器を独自に定義する場合は vnd を接頭語とする。

deviceType を定義する場合は、７．５．１も参照のこと。

例：vndLightingWithSensor

protocol

独自に定義可能とする。ただし、type、version は記述する。

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

manufacturer

"code": ""（空文字列）とする。descriptions は独自定義で記述する。

７．５ 機器情報（Device Description）の拡張に関する指針

５．７記載の機器情報（Device Description）に関して、「機器仕様部」（別書）に定義されている機器情報は、名称変更せず利用することを推奨する。各機器が対応外のプロパティ・プロパティ値・アクションを削除するのは構わない。

なお、新たな機器を追加したい、または既存の機器情報に対して新たなプロパティを追加したい場合、機器情報をベンダ独自で拡張しても良い。将来、機器仕様部の定義が拡張された場合に名称が重複しないよう、機器情報をベンダ独自で追加拡張する場合の指針について下記にまとめる。

７．５．１ 新たな機器の追加

「機器仕様部」（別書）で定義されていない ECHONET Lite 機器、４．４記載の仮想的な機器、ECHONET Lite 以外の方式の機器など、「機器仕様部」（別書）で定義されていない機器を新規に定義してもよい。

新たな機器を追加する場合、deviceType の名前の先頭に vnd を付ける。プロパティやアクションの名前の先頭に vnd を付ける必要はない。ECHONET 規格の APPENDIX ECHONET 機器オブジェクト詳細規定に対応する定義がない場合は、eoj と epc は省略可とする。

７．５．２ 既存の機器情報に対する拡張

「機器仕様部」（別書）で既に定義されている機器情報に対して変更を行う場合、以下 3 つの条件のいずれかに該当する場合は、ベンダ独自の拡張として扱うことが望ましい。

1. 既存のプロパティやアクションと異なる機能のプロパティやアクションを追加する場合。
   機器仕様部」（別書）で定義済みの機器に対して、新たなプロパティやアクションを追加する場合は、プロパティやアクションの名前の先頭に vnd を付けた名前とする。ECHONET 規格の APPENDIX ECHONET 機器オブジェクト詳細規定に対応する定義がない場合は、epc は省略可とする。
   例：vndYUV（一般照明クラスなら RGB）

2. 既存のプロパティやアクションと同じ名前のプロパティやアクションを定義しているが、共通のプロパティ値が全く存在しないスキーマ定義になっている場合。
   ただし、１つでもプロパティ値が重複している場合は、本条件に該当しない。
   例：既存定義の type が number で、独自定義の type が string
   1. と同様に、プロパティやアクションの名前の先頭に vnd を付けた名前とする。

3. 既存の列挙型（enum）のプロパティ値に列挙子を追加する場合。
「機器仕様部」（別書）で定義済みの列挙型（enum）のプロパティ値に対して、新たな列挙子を追加する場合は、vnd を接頭語とする。
例：運転モードの"enum":["auto","cooling","heating"] に、"vndExtreamCooling" の列挙子を追加

property object の内容については、「表 5-4 Property Object の内容」に定義されている内容の変更は行わないことを推奨する。ただし、ベンダ独自に要素を追加することは可能とする。その際の名称は vnd を接頭語とする。

例：vndOriginalProtoCode

「表 5-4 Property Object の内容」に定義されている内容は、以下のように扱う。

epc

省略可（ECHONET Lite 以外の機器においては不要）

epcAtomic

省略可（ECHONET Lite 以外の機器においては不要）

descriptions

独自定義で記述する。

urlParameters

独自定義で記述可。ただし JSON Schema 形式で記述する。

schema

独自定義で記述可。ただし JSON Schema 形式で記述する。

第８章 おわりに

本書では、宅内の ECHONET Lite モデルをクラウドなどのサーバ上にマッピングし、宅内の ECHONET Lite 対応機器などを Web API 化したサービスを介して外部クライアントから活用可能とするシステムの構築に際し、Web API 設計時に考慮すべき各種指針について示した。

本書は前回ガイドラインで記述した機器操作などの基本ユースケースに加え、特に、応用ユースケース（複数命令の一括指示、機器のグルーピング、履歴データ機器など）にフォーカスをあて、提供可能なサービスの幅を広げ、より高機能なサービスを提供できるよう、ガイドラインの策定をはかった。今後、対応機器の種類を順次増やしつつ、応用ユースケースの拡充やサービス事業者向けに活用しやすい豊富な機能や環境について整備を進めていく。

第９章 謝辞

本書を作成するにあたり、神奈川工科大学の藤田裕之氏には、WG へオブザーバ参加いただき、検討・執筆に際し多大なるご協力を賜りました。深謝いたします。また、議論に参加いただいた関係各位に感謝の気持ちと御礼を申し上げます。