Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 28 August 2026

B. Brinckman
R. Mohan
Cisco Systems
B. Sanford
Philips

24 February 2026

An Application Layer Interface for Non-Internet-Connected Physical Components (NIPC)
draft-ietf-asdf-nipc-18

Abstract

This document describes an API that allows applications to perform operations against a gateway serving one or more devices described by an SDF model. The API consists of a RESTful application layer interface that performs operations on those devices, as well as a CBOR-based publish-subscribe interface for streaming data.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-asdf-nipc/.

Discussion of this document takes place on the A Semantic Definition Format for Data and Interactions of Things Working Group mailing list (mailto:asdf@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/asdf/. Subscribe at https://www.ietf.org/mailman/listinfo/asdf/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-wg-asdf/asdf-nipc.

Status of This Memo

This Internet-Draft will expire on 28 August 2026.

Table of Contents

1. Introduction 5
1.1. Motivation 5
1.2. Non-IP Gateway 6
1.3. Terminology 8
1.4. Glossary 8
2. Architecture 10
2.1. Overview 10
2.1.1. Device instance information 11
2.1.2. Device class information 11
2.2. NIPC Registrations 11
2.2.1. SDF model registrations 12
2.2.2. Data application registrations 12
2.3. NIPC Operations 12
2.3.1. Overview 12
2.3.2. Properties 13
2.3.3. Actions 13
2.3.4. Events 14
2.3.5. Triggers 14
2.3.6. Groups 14
2.3.7. Connection management for NIPC Operations 14
2.3.8. Extensions 15
2.4. Events publish subscribe interface 15
2.5. Paths 15
2.5.1. General 15
2.5.2. NIPC Registrations 17

2.5.3. NIPC Operations 17
2.6. Schema 17
2.6.1. SDF model registrations 17
2.6.2. NIPC Operations 18
2.6.3. Parameters 19
2.6.4. Responses 19
3. NIPC Registration APIs 22
3.1. SDF model registrations APIs 22
3.1.1. Register an SDF model 22
3.1.2. Get all SDF models 23
3.1.3. Get an SDF model 24
3.1.4. Delete an SDF model 24
3.1.5. Update an SDF model 25
3.2. Data application registrations APIs 26
3.2.1. Register a data application 27
3.2.2. Update a data application 30
3.2.3. Get a data application 31
3.2.4. Delete a data application 31
4. NIPC Operation APIs 32
4.1. NIPC Property APIs 32
4.1.1. Update one or multiple values 33
4.1.2. Read one or multiple values 35
4.2. NIPC Event APIs 37
4.2.1. Enable event reporting 38
4.2.2. Disable event reporting 39
4.2.3. Get status of one or more events 39
4.2.4. Enable event reporting on a group of devices 40
4.2.5. Disable event reporting on a group of devices 41
4.2.6. Get event status on a group of devices 43
4.3. NIPC Action APIs 44
4.3.1. Perform an action 44
4.3.2. Check action status 45
4.4. NIPC Trigger APIs 46
4.4.1. Create a trigger on a device 46
4.4.2. Delete a trigger on a device 48
4.4.3. Get installed triggers for a device 48
4.4.4. Create a trigger on a group of devices 50
4.4.5. Delete a trigger on a group of devices 51
4.4.6. Get installed triggers for a group of devices 51
4.5. NIPC explicit connection management APIs 53
4.5.1. Protocol Information Object 53
4.5.2. Connect to a device 55
4.5.3. Update a connection 59
4.5.4. Disconnect from a device 62
4.5.5. Get connection status 62
5. NIPC Extensibility 64
5.1. Protocol mappings 64
5.2. API extensions 64

6. NIPC Error Handling 65
7. Publish/Subscribe Interface 67
7.1. CDDL Definition 67
7.2. CBOR Examples 70
8. Examples 71
8.1. Property Read/Write 71
8.2. Enabling an Event on a Device 73
8.3. Enabling an Event on a Group of Devices 76
9. Implementation Status 78
9.1. TieDie IoT 79
9.2. Cisco Sensor Connect for IoT Services (Catalyst) 79
9.3. Cisco Sensor Connect for IoT Services (Meraki) 80
9.4. NIPC Prototype 80
10. Security Considerations 80
10.1. Payload Encryption Considerations 80
10.2. TLS Support Considerations 81
10.3. HTTP Considerations 81
10.4. Authorization Considerations 81
10.4.1. API authorization Considerations 81
10.4.2. Authorization Token/Bearer Token/Cookie Considerations 81
10.5. Other Security Considerations 82
11. IANA Considerations 82
11.1. Media Type Registration 83
11.2. API extensions 84
11.3. Well-known URIs 85
11.4. Data Subscription Types 86
11.5. NIPC Protocols 87
11.6. Problem Details for NIPC APIs 88
12. References 91
12.1. Normative References 91
12.2. Informative References 94
Appendix A. OpenAPI definition 94
Appendix B. Protocol Mapping 135
Appendix C. Protocol Information 135
C.1. Protocol Information for BLE 136
C.2. Protocol Information for Zigbee 139
Appendix D. NIPC API extensions 141
D.1. NIPC API write binary blob extension 141
D.2. NIPC API bulk operations extension 143
D.3. NIPC API write file extension 153
D.4. NIPC API conditional read extension 159
D.5. NIPC API conditional event extension 164
D.6. NIPC API property extensions 170
Appendix E. NIPC API CDDL Definition 174
Appendix F. Example SDF model with protocol mappings for BLE . . 181
Appendix G. Acknowledgements 185
Authors' Addresses 185

1. Introduction

1.1. Motivation

Low-power sensors, actuators, and other connected devices deployed for building management, healthcare, workplace, manufacturing, logistics, and hospitality use cases are often resource and battery constrained. Many lack native IP connectivity and instead attach via heterogeneous non-IP operational networks. Common non-IP protocols include BLE [BLE53] and Zigbee [Zigbee22]. When IP is available, constrained application protocols such as CoAP [RFC7252] may be used. These devices still need to exchange data with IP-based applications. Accordingly, applications on the IP network obtain telemetry from and issue operations to such devices through an application-layer gateway. This gateway bridges the application network and one or more separate operational networks where devices are connected, allowing applications on the IP network to perform operations on devices connected to these other operational networks.

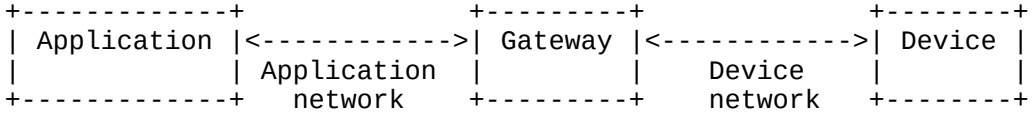

Figure 1: Gateway for non-Internet-Connected Devices

There have been efforts to define Gateway functions for devices that support a particular protocol, such as a BLE GATT REST API for BLE Gateways ([Gatt-REST-API]), however they have been limited to a single protocol or a particular use case. In absence of an open standard describing how applications on an IP network communicate with devices on other operational networks, bespoke and vendor-specific implementations have proliferated. Each deployment then requires: (1) defining or adapting yet another API, and (2) deploying additional gateway functions, increasing operational and integration cost. This specification defines a single, extensible application layer interface for cross-network and cross-protocol device interaction through a network gateway. The intent is to support multiple network and network protocols (and versions) concurrently, allow incremental addition of new protocols via mapping, and reduce redundant infrastructure by enabling multiple applications to share one standardized gateway function. Furthermore, by leveraging interaction models, the application and gateway are able to maintain a protocol-neutral interface, while the gateway handles the protocol-specific interactions with devices.

A standardized Application Layer Gateway interface has the following benefits:

1. Eliminates repeated bespoke integration effort across deployments.

2. Avoids deploying multiple overlapping gateway functions for different networks, protocols or use cases.

3. Reduces time and operational cost to integrate new networks and devices.

4. Allows applications to interact with devices in a protocol-neutral way, leveraging interaction models.

1.2. Non-IP Gateway

A Non-Internet-Connected Physical Components (NIPC) gateway is an application layer gateway (ALG) that implements APIs for applications to communicate with devices on different networks connected to the Gateway. These devices may leverage different protocols, IP based or non-IP based. NIPC APIs allow reading or writing properties of devices, invoking actions on devices, as well as enabling or disabling events on devices, by means of a supporting gateway, in a protocol-neutral way.

In order to perform NIPC operations on a device, 2 prerequisites must be fulfilled:

* The gateway has access to a device object, that contains its identity, in the form of a unique UUID and any credentials & trust material required to communicate with the device. Provisioning this device object is out of scope of this document. It may be performed via SCIM [RFC7644] with [I-D.ietf-scim-device-model].

* An interaction model for the class of devices must be available to the gateway. This allows the gateway to understand how to interact with the device in a protocol-neutral way. The interaction model is provided to the gateway by means of an SDF model, as described in [RFC9880].

Once these prerequisites are met, the gateway can resolve an SDF affordance referenced in the SDF model into the protocol-specific operations required for that device.

A NIPC gateway provides the following functions:

* Authentication and authorization of application clients that will leverage the NIPC APIs.

* Maintain or have access to a repository of device objects, including device identity and trust material.

* Accept and validate SDF interaction model registrations.

* Expose APIs for property, action, and event operations.

* Perform implicit connection management to devices where required; optionally support explicit connection management.

* Stream events (publish/subscribe) to authorized data applications.

* Proxy payloads between networks without interpreting or modifying application data.

* Operate one or more channels to supported wired or wireless networks.

* Optionally provide a bridge between devices on one or more device networks connected to the NIPC-Gateway. This may include translating between different protocols, if multiple protocols are supported on the device network(s).

The gateway' s role is to provide gateway functions between application and device networks; it is not intended to be middleware that inspects, decodes, or transforms device payloads.

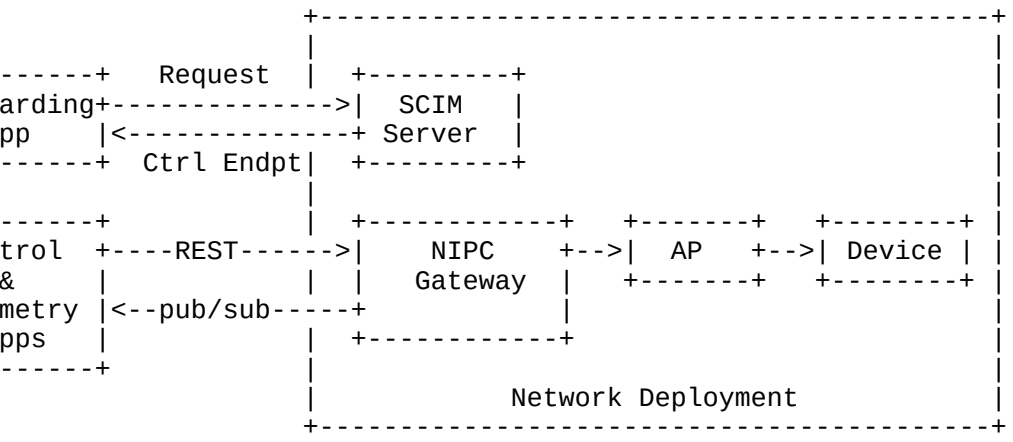

Figure 2: Basic Architecture

Figure 2 illustrates the basic components: applications, the NIPC application-layer gateway (ALG), an access point (AP), and a device (D). The applications, ALG, and AP reside on an IP network; the AP

provides a wireless or wired interface to the device. Applications often operate in a different administrative domain than the ALG and AP, so the ALG will have to support authorization. The ALG bridges the IP application domain and the device network, be it an IP-based or non-IP device network. This enables applications to perform operations on devices attached to those device networks. Applications use a JSON-based [RFC8259] RESTful NIPC APIs for property, action, and event operations, and a CBOR-based [RFC8949] publish/subscribe interface for event streaming.

1.3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.4. Glossary

* NIPC: Non-Internet-Connected Physical Components, the application layer interface defined in this document.

* NIPC Gateway: An application layer gateway that implements the NIPC interface.

* SDF: Semantic Device Format, a standardized format to describe devices and their interaction models, as defined in [RFC9880].

* SDF Affordance: An interaction point on a device as defined in an SDF model. Examples of affordances are properties, actions, and events.

* SDF Global Name: Absolute URI (with fragment) identifying an SDF affordance.

* SCIM: System for Cross-domain Identity Management, a standardized protocol to manage identity information, as defined in [RFC7644].

* ALG: Application Layer Gateway.

* IoT: Internet of Things.

* Protocol Mapping / sdfProtocolMap: Mapping from protocol-neutral SDF affordances to protocol-specific operations.

* BLE: Bluetooth Low Energy protocol.

* Zigbee: Low-power mesh networking protocol.

* GATT: Generic Attribute Profile used in BLE for services/characteristics/descriptors.

* Service (BLE): Top-level GATT grouping of characteristics.

* Characteristic (BLE): GATT data element supporting read/write/notify.

* Descriptor (BLE): Metadata element attached to a characteristic.

* Bonding (BLE): Procedure to establish trusted, reusable security keys.

* Service Discovery (BLE): Procedure to enumerate GATT services/characteristics/descriptors.

* Device ID / Group ID: UUID identifying a device or a group of devices.

* UUID: Universally unique identifier (128-bit).

* Data Application / Data App: Registered application receiving streamed event data.

* MQTT: Publish/subscribe messaging protocol used for streaming.

* Webhook: HTTP callback endpoint for push delivery.

* Websocket: Bidirectional TCP-based message channel over HTTP.

* Publish/Subscribe Interface: Streaming channel for events (CBOR-encoded payloads).

* CBOR: Concise Binary Object Representation; compact binary data format.

* CDDL: Concise Data Definition Language; schema language for CBOR data.

* JSON: JavaScript Object Notation; text encoding used for API payloads.

* Access Point (AP): Network element with a radio interface communicating with devices.

2. Architecture

2.1. Overview

A Non-Internet-Connected Physical Components (NIPC) gateway is an application-layer gateway (ALG) that exposes APIs enabling applications to perform operations on devices attached to networks connected to the gateway. NIPC defines two API categories:

* Registrations: register SDF models for classes of devices and register data applications that receive streaming event data.

* Operations: perform protocol-neutral device interactions (read/write properties, invoke actions, enable/disable events) across heterogeneous networks and protocols.

To execute NIPC operations on a device, both prerequisites MUST be met:

1. The NIPC gateway has access to device instance information: The device object contains its identity, in the form of a unique UUID and any credentials/trust material required to communicate with the device (e.g., via SCIM [RFC7644] with [I-D.ietf-scim-device-model]). This device object is identified by the device ID referenced in NIPC API paths.

2. The NIPC gateway has access to an interaction model (device class information): An SDF model [RFC9880] is registered, providing protocol-neutral affordances and mappings to protocol-specific operations.

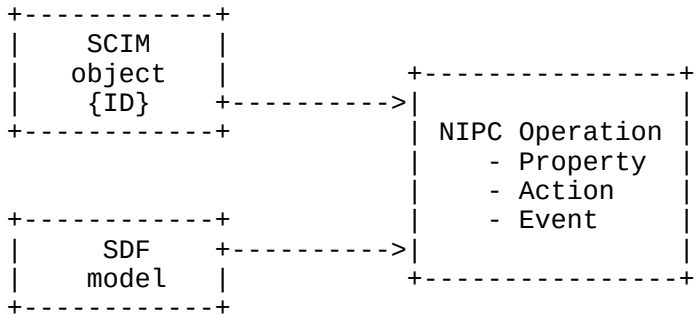

Figure 3: NIPC prerequisites

Once both prerequisites are met, authorized applications can perform NIPC operations on devices identified by their IDs. See Section 10 for authorization details. NIPC operations act on SDF

affordances—properties, actions, and events defined in the registered SDF model. Certain NIPC operations may also be performed on groups of devices identified by a group ID.

2.1.1. Device instance information

In order for the NIPC gateway to perform operations on a device, it must have access to the device's instance information. This includes the device's identity and any credentials or trust material required to communicate with the device. The device object MUST include a unique identity (UUID) and sufficient information to bootstrap trust and establish connectivity, as NIPC operations assume connectivity can be established without separate API calls. While provisioning devices instance information can be performed in various ways, it is RECOMMENDED to use SCIM [RFC7644] with the device schema [I-D.ietf-scim-device-model], which defines the necessary attributes and extensions to support NIPC. As per Section 4.2 of [RFC7643], group objects may also be declared, and leveraged in NIPC operations.

2.1.2. Device class information

Device class information is declared through SDF models, as defined in [RFC9880]. These SDF models define the protocol-neutral affordances of a class of devices, as well as protocol mappings [I-D.ietf-asdf-sdf-protocol-mapping] that relate these affordances to protocol-specific operations. The SDF model for a class of devices can be registered through NIPC registration APIs, as described in Section 2.2.

The SDF model reference and/or data-app registration MAY also be included in a device's SCIM object. See [I-D.ietf-scim-device-model] and [I-D.ietf-asdf-sdf-protocol-mapping] for details.

2.2. NIPC Registrations

NIPC registration APIs allow applications to register objects that are not tied to specific device instances.

NIPC supports two registration types:

1. SDF model registration: Registers an SDF interaction model for a class of devices.

2. Data application authorization: Authorizes an application to receive streaming event data.

2.2.1. SDF model registrations

The SDF model for a class of devices determines how a gateway can interact with these devices in a protocol-neutral way. To enable this, the SDF model must contain protocol mappings, mapping protocol-neutral SDF affordances to protocol-specific operations as defined in [I-D.ietf-asdf-sdf-protocol-mapping]. The SDF affordances supported by the device, as well as its protocol-mappings, are provided to the gateway by means of an SDF model. SDF models are described in [RFC9880].

2.2.2. Data application registrations

NIPC operations can enable or disable event reporting on a device. Events are reported through a publish-subscribe interface. Applications that are authorized to perform NIPC operations on devices can define which applications are permitted to receive streaming event data for that device. The data-app registrations API maps an event to an application that is authorized to receive that data. The registration also defines what protocol will be used to deliver the data (e.g., MQTT, webhook, websocket). This registration basically allows applications to instruct the gateway to direct event data-streams to specific data-applications.

2.3. NIPC Operations

2.3.1. Overview

NIPC APIs are exposed over HTTP [RFC9110]. Requests and responses use JSON [RFC8259] unless another media type is negotiated via Content-Type and Accept. A media type for an SDF affordance can also be stipulated in the SDF ContentFormat data quality, as described in Section 4.7 of [RFC9880]. The default media type is “application/nipc+json” (see Section 11.1). SDF model registrations use “application/sdf+json” . Property APIs MAY use other media types appropriate to the property payload.

Failures use Problem Details [RFC9457] with media type application/problem+json.

NIPC operations are protocol-neutral operations on SDF affordances, more specifically properties, actions & events. NIPC operations can happen against affordances registered in an SDF model. Operations reference affordances by their SDF global name. If the underlying protocol requires a connection, the gateway establishes and tears down the connection implicitly unless an explicit connection is already in place.

NIPC exposes four operation groups:

* Properties APIs: These APIs allow applications to perform operations on properties, such as to read or write values to them.

* Actions APIs: These APIs perform actions on devices, such as enabling or disabling a feature on a device.

* Events APIs: These APIs allow apps to enable or disable event reporting on devices. Events are reported over the events publish/subscribe interface.

* Trigger APIs: These APIs allow an event on one device or group to trigger an action on another device or group. Trigger APIs are not a fundamental operation, but rather tie 2 fundamental operations together; an event triggers an action.

2.3.2. Properties

Property operations allow clients to read and write values for SDF properties.

An example of using a property API is reading the property temperature from a temperature sensor.

Requests and responses use application/nipc+json unless another media type is negotiated via Content-Type and Accept. When using JSON, binary property values are base64-encoded with padding per Section 5 of [RFC4648]. Multiple properties MAY be read or written in a single request. When a single property is addressed via a query parameter, non-JSON media types MAY be used for the payload. On success, the response returns either 200 with per-property status (JSON) or 204 No Content for single, non-JSON writes.

2.3.3. Actions

Action operations invoke SDF actions on devices.

An example of using an action API is to turn on a lightbulb.

A successful action request returns 202 Accepted with a Location header referencing the action instance. Clients poll the instance URI to obtain status (e.g., IN_PROGRESS, COMPLETED). Request bodies are optional and MAY carry action input in a media type appropriate to the underlying protocol (e.g., octet-stream).

2.3.4. Events

Event operations enable and disable device event reporting.

An example of using an event API receiving an event that a button has been pressed.

Enabling an event returns 201 Created with a Location header referencing the event instance. Disabling an event uses the instance identifier and returns 204 No Content on success (for a single device) or 200 Success with a per-device status list (for a group). Event payloads are delivered via the publish/subscribe interface encoded in CBOR [RFC8949].

2.3.5. Triggers

Triggers allow an event on one device or group to trigger an action on another device or group. Triggers are not protocol-specific. As an example, an event on a BLE device can trigger an action on a Zigbee device.

An example of a Trigger is when a button is pressed (event), a lightbulb should turn on (action). Multiple buttons (group) can also turn on a lightbulb (device). A single button (device) could also turn on multiple lights (group).

Creating a trigger returns 201 Created with a Location header referencing the trigger instance. Deleting a trigger uses the instance identifier and returns 204 No Content on success (for a single device) or 200 Success with a per-device trigger list (for a group).

2.3.6. Groups

Where supported by the underlying protocol, operations MAY target a group of devices identified by a group ID. Responses for group operations return per-device results; failures for individual devices are reported using Problem Details entries within the array.

2.3.7. Connection management for NIPC Operations

For protocols that require connection setup, the gateway performs implicit connection management during an operation (establish on demand; tear down on completion). Gateways MAY support explicit connection management; when an explicit connection is active, operations reuse it and do not tear it down. Explicit connection management is described in Section 4.5.

2.3.8. Extensions

NIPC supports API extensions for compound or specialized operations. Extensions can execute a set of NIPC operations in a single request or provide more efficient mechanisms for specific use cases (e.g., a bulk operation).

Extensions MUST use the “/extensions” path element. To ensure interoperability, extensions MUST be registered with IANA as defined in Section 11.2.

2.4. Events publish subscribe interface

Events are delivered via a publish/subscribe interface. NIPC events are encoded in CBOR ([RFC8949]) and can be transported over MQTT, Webhook or Websocket.

CBOR is used for the publish/subscribe interface as Non-IP payloads are typically binary. CBOR encodes binary payloads efficiently, and is more compact than JSON, therefore reducing the amount of data that needs to be transmitted to the application.

Event types include:

* Streaming data from devices: Streaming data is activated/ deactivated with the NIPC events API

* Broadcasts from devices (e.g., advertisements in BLE)

* Connection events: Devices connecting & disconnecting

2.5. Paths

2.5.1. General

The NIPC HTTP protocol is described in terms of a path relative to a Base URI. The Base URI MUST NOT contain a query string, as clients MAY append additional path information and query parameters as part of forming the request. The base URI is a URL that most often consists of the "https" protocol scheme, a domain name, and an initial path [RFC3986]. That initial path for NIPC is recommended to be /nipc. For example:

"https://example.com/nipc/"

Additionally a version number may be added, for example:

"https://example.com/nipc/v1/"

After the base or version number, the path must contain a collection identifier. The collection identifier can be one of the following:

* /registrations: for NIPC registration APIs

* /devices: for NIPC operations on devices

* /groups: for NIPC operations on groups of devices

* /extensions: for NIPC extension APIs

The well-known URI /.well-known/nipc defined in Section 11.3 can be used to discover the base path of the NIPC APIs and the supported versions and extensions. The response to a GET request on this URI MUST be a JSON document that contains the base path, and optionally the supported versions and
extension APIs. The paths MUST be a URI template as defined in [RFC6570]. The following is an example of a template defining the NIPC base path as well as supported extensions on a server.

<CODE BEGINS>
```
{
  "base_path": "/nipc",
  "versions": [
    "/v1"
  ],
  "extensions": [
    "/extensions/{id}/bulk",
    "/extensions/{id}/properties/blob",
    "/extensions/{id}/properties/file",
    "/extensions/{id}/properties/read/conditional",
    "/extensions/{id}/events/conditional",
    "/extensions/{id}/properties/write"
  ]
}
```
<CODE ENDS>

Figure 4: Example response for /.well-known/nipc

A formal CBOR definition of the well-known response is as follows:

<CODE BEGINS> file "nipc_well_known.cddl"
```
NipcWellKnown = {
  base_path: text,
  ? versions: [* uri / text],
  ? extensions: [* uri / text]
}
```
<CODE ENDS>

2.5.2. NIPC Registrations

Registrations leverage the base path + /registrations. NIPC supports SDF model registrations and data-app registrations.

paths:

* /registrations/models

* /registrations/data-apps

2.5.3. NIPC Operations

Every NIPC Operations API pertains to either a device or group of devices, identified by an ID, hence the ID must be reflected as the first parameter in the path. For example:

"https://example.com/nipc/v1/{id}"

The second parameter in the path refers to the NIPC operation that the API will perform on the device. This can be:

* properties

* events

* actions

* triggers

* extensions

These are described in Section 2.3.

2.6. Schema

The NIPC schema leans heavily on the SDF schema, as defined in [RFC9880]. NIPC operations map directly to SDF affordances.

2.6.1. SDF model registrations

To execute NIPC operations, an SDF interaction model for the device class MUST be registered. The model MUST include protocol mappings that relate protocol-neutral SDF affordances to protocol-specific operations.

Registration is performed via POST /registrations/models with the SDF model in the request body. A registered model can be retrieved via GET /registrations/models using the model identifier (sdfName).

2.6.2. NIPC Operations

NIPC operations require two parameters:

1. Device ID: the UUID identifying the target device (or group).

2. sdfName: the SDF global name (absolute URI with fragment) of the affordance (property, action, or event) on which the operation acts.

2.6.2.1. Device ID

All NIPC operations are executed against a device or a group of devices. Devices or groups of devices are identified by a unique UUID, adhering to [RFC9562].

Attribute	Type	Example
id	uuid	1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30

Table 1: Definition of a device or group of devices

ID is a UUID assigned to uniquely identify a device to the NIPC Gateway (e.g., by a SCIM server) and the application calling the APIs. The NIPC Gateway must have access to a device object identified by this device ID and the application must store this UUID in order to use it to identify the device on which it wants to perform NIPC operations.

2.6.2.2. SDF Name

Operations act on SDF affordances and reference them by SDF global name—an absolute URI with fragment that includes the namespace. The affordance referenced MAY be a property, action, or event; the reference is carried as a string.

For example:

NIPC Operation	Example SDF Name used in the API
Property	https://example.com/heartrate#/ sdfObject/thermostat/sdfProperty/temperature
Event	https://example.com/heartrate# /sdfObject/healthsensor/sdfEvent/fallDetected
Action	https://example.com/heartrate#/ sdfObject/thermostat/sdfAction/resetThermostat

Table 2: Example SDF names for NIPC operations

2.6.3. Parameters

To minimize deployment risk arising from inconsistent URI path normalization and handling of percent-encoded reserved characters in path elements (notably “/” encoded as “%2F”) across common HTTP servers and intermediaries (e.g., Jetty, Tomcat, Apache httpd, NGINX), NIPC uses query parameters (except the primary {id} path segment) rather than additional path segments for values that can legitimately contain reserved characters (such as SDF global names). Although [RFC3986] and [RFC9110] require that reserved characters not be normalized or decoded in ways that alter semantics, misinterpretation of these rules has led to security vulnerabilities (path confusion, cache poisoning, authorization bypass) and to conservative default configurations that reject encoded slashes. In proxy deployments, relaxing these defaults often triggers security review friction. Representing such values in the query component avoids reliance on tolerant path normalization behavior, reduces ambiguity for intermediaries, and aligns with prevailing “safe” operational profiles; a malformed or unsupported parameter can be rejected with a 4xx status without exposing downstream services to traversal or normalization discrepancies.

2.6.4. Responses

A NIPC Gateway must respond to a NIPC operation request synchronously, and provide the result of the completed operation in the HTTP response.

Exceptions to this are the following:

1. Extensions: Extension APIs (see Section 5.2) execute compound operations and thus require the gateway to execute multiple NIPC operations. On acceptance, the gateway returns 202 Accepted.

Clients poll the extension URI (GET) for execution status. If a callback URI was supplied in the request, the gateway MAY send the final result upon completion.

2. Actions: Action requests return 202 Accepted with a Location header pointing to the action instance used for status tracking.

A failure response must contain an HTTP status code of 4xx or 5xx, and use [RFC9457] Problem Details with application/problem+json media type.

All NIPC failure responses must include the following attributes:

* type: a URI identifying the error (see Section 6)

* status: the 4xx or 5xx HTTP status code

* title: a brief, human-readable summary

* detail: a human-readable explanation specific to this occurrence Additional attributes MAY be included as permitted by [RFC9457].

<CODE BEGINS> file "failure_response.cddl"
=============== NOTE: '\' line wrapping per RFC 8792 ================

```
FailureResponse = {
  ? type: FailureTypeURI,
  ? status: uint,
  ? title: text,
  ? detail: text
}

; Enumerated problem type URIs registered for NIPC
FailureTypeURI = (
  "https://www.iana.org/assignments/nipc-problem-types#invalid-id" /
  "https://www.iana.org/assignments/nipc-problem-types#invalid-sdf-u\
rl" /
  "https://www.iana.org/assignments/nipc-problem-types#extension-ope\
ration-not-executed" /
  "https://www.iana.org/assignments/nipc-problem-types#sdf-model-alr\
eady-registered" /
  "https://www.iana.org/assignments/nipc-problem-types#sdf-model-in-\
use" /
  "https://www.iana.org/assignments/nipc-problem-types#property-not-\
readable" /
  "https://www.iana.org/assignments/nipc-problem-types#property-read\
-failed" /
  "https://www.iana.org/assignments/nipc-problem-types#property-not-\
```

```
   writable" /
     "https://www.iana.org/assignments/nipc-problem-types#property-writ\
   e-failed" /
     "https://www.iana.org/assignments/nipc-problem-types#event-already\
   -enabled" /
     "https://www.iana.org/assignments/nipc-problem-types#event-not-ena\
   bled" /
     "https://www.iana.org/assignments/nipc-problem-types#event-not-reg\
   istered" /
     "https://www.iana.org/assignments/nipc-problem-types#protocolmap-b\
   le-already-connected" /
     "https://www.iana.org/assignments/nipc-problem-types#protocolmap-b\
   le-no-connection" /
     "https://www.iana.org/assignments/nipc-problem-types#protocolmap-b\
   le-connection-timeout" /
     "https://www.iana.org/assignments/nipc-problem-types#protocolmap-b\
   le-bonding-failed" /
     "https://www.iana.org/assignments/nipc-problem-types#protocolmap-b\
   le-connection-failed" /
     "https://www.iana.org/assignments/nipc-problem-types#protocolmap-b\
   le-service-discovery-failed" /
     "https://www.iana.org/assignments/nipc-problem-types#protocolmap-b\
   le-invalid-service-or-characteristic" /
     "https://www.iana.org/assignments/nipc-problem-types#protocolmap-z\
   igbee-connection-timeout" /
     "https://www.iana.org/assignments/nipc-problem-types#protocolmap-z\
   igbee-invalid-endpoint-or-cluster" /
     "https://www.iana.org/assignments/nipc-problem-types#extension-tra\
   nsmit-invalid-data" /
     "https://www.iana.org/assignments/nipc-problem-types#extension-fir\
   mware-rollback" /
     "https://www.iana.org/assignments/nipc-problem-types#extension-fir\
   mware-update-failed" /
     "about:blank"
   )
   <CODE ENDS>
```

Example of a failure response:

```
<CODE BEGINS>
=============== NOTE: '\\' line wrapping per RFC 8792 ===============

{
  "type": "https://www.iana.org/assignments/nipc-problem-types#inval\
\id-id",
  "status": 400,
  "title": "Invalid Device ID",
  "detail": "Device ID 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30 does not\
\ exist or is not a device"
}
<CODE ENDS>
```

Figure 5: Example failure response

3. NIPC Registration APIs

3.1. SDF model registrations APIs

SDF model registration APIs allow applications to register an SDF model for a class of devices. These APIs use the application/sdf+json media type, as described in Section 7.1 of [RFC9880].

3.1.1. Register an SDF model

Method: POST /registrations/models

Description: Registers one or more SDF models for a class of devices.

Request Body:

* The SDF document in JSON format containing one or more sdfThings or sdfObjects, similar to the example in Appendix F.

* The SDF document MUST contain protocol mappings, as described in [I-D.ietf-asdf-sdf-protocol-mapping].

Response:

A list containing objects where each object has an "sdfName" which is the global name of the top-level sdfThing or sdfObject in the SDF model.

```
<CODE BEGINS> file "sdf_reference.cddl"
=============== NOTE: '\' line wrapping per RFC 8792 ================

SdfGlobalName = text    ; absolute URI with fragment referencing an \
sdfThing or sdfObject

SdfReference = {
  sdfName: SdfGlobalName
}

SdfReferenceArray = [* SdfReference]
<CODE ENDS>
```

Example of a response:

```
<CODE BEGINS>
[
 {
  "sdfName": "https://example.com/heartrate#/sdfObject/healthsensor"
 }
]
<CODE ENDS>
```

Figure 6: Example of a response to an SDF model registration

3.1.2. Get all SDF models

Method: GET /registrations/models

Description: Gets all SDF models registered with the gateway.

Response:

A list containing objects where each object has an "sdfName" which is the global name of the top-level sdfThing or sdfObject in the SDF model.

```
<CODE BEGINS> file "sdf_reference.cddl"
=============== NOTE: '\' line wrapping per RFC 8792 ================

SdfGlobalName = text    ; absolute URI with fragment referencing an \
sdfThing or sdfObject

SdfReference = {
  sdfName: SdfGlobalName
}

SdfReferenceArray = [* SdfReference]
<CODE ENDS>
```

Example of a response:

```
<CODE BEGINS>
[
 {
  "sdfName": "https://example.com/heartrate#/sdfObject/healthsensor"
 },
 {
  "sdfName": "https://example.com/thermometer#/sdfObject/thermometer"
 }
]
<CODE ENDS>
```

Figure 7: Example of a response to get all SDF models

3.1.3. Get an SDF model

Method: GET /registrations/models{?sdfName}

Description: Gets an SDF model registered with the gateway.

Query Parameters:

* sdfName: the name of the top-level sdfThing or sdfObject in the SDF model.

Response:

* The SDF model is returned in JSON format, similar to the example in Appendix F.

3.1.4. Delete an SDF model

Method: DELETE /registrations/models{?sdfName}

Description: Deletes an SDF model registered with the gateway.

Query Parameters:

* sdfName: the name of the top-level sdfThing or sdfObject in the SDF model.

Response:

* A list containing objects where each object has an "sdfName" which is the global name of the top-level sdfThing or sdfObject in the SDF model

```
<CODE BEGINS> file "sdf_reference.cddl"
=============== NOTE: '\' line wrapping per RFC 8792 ================

SdfGlobalName = text    ; absolute URI with fragment referencing an \
sdfThing or sdfObject

SdfReference = {
  sdfName: SdfGlobalName
}

SdfReferenceArray = [* SdfReference]
<CODE ENDS>
```

Example of a response:

```
<CODE BEGINS>
{
  "sdfName": "https://example.com/heartrate#/sdfObject/healthsensor"
}
<CODE ENDS>
```

Figure 8: Example of a response to an SDF model registration

3.1.5. Update an SDF model

Method: PUT /registrations/models{?sdfName}

Description: Updates an SDF model registered with the gateway.

Query Parameters:

* sdfName: the name of the top-level sdfThing or sdfObject in the SDF model.

Request Body:

* An SDF model in JSON format, similar to the example in Appendix F.

Response:

* A list containing objects where each object has an "sdfName" which is the global name of the top-level sdfThing or sdfObject in the SDF model

```
<CODE BEGINS> file "sdf_reference.cddl"
=============== NOTE: '\' line wrapping per RFC 8792 ================

SdfGlobalName = text    ; absolute URI with fragment referencing an \
sdfThing or sdfObject

SdfReference = {
  sdfName: SdfGlobalName
}

SdfReferenceArray = [* SdfReference]
<CODE ENDS>
```

Example of a response:

```
<CODE BEGINS>
{
  "sdfName": "https://example.com/heartrate#/sdfObject/healthsensor"
}
<CODE ENDS>
```

Figure 9: Example of a response to an SDF model registration

3.2. Data application registrations APIs

Data-app registration APIs allow applications to register a data application that will receive streaming data from the gateway. These APIs operate on a data app ID. This ID corresponds to the endpoint app ID of the telemetry endpoint app that is registered with the SCIM server as an authorized endpoint that can receive data for a device. The endpoint app is defined in Section 6 of [I-D.ietf-scim-device-model].

Apart from the endpoint app ID, the API also configures the protocol the data-app supports. These should be one of the following:

* mqttClient: The data-app is an MQTT client, which means that the NIPC gateway must act as an MQTT broker.

* mqttBroker: The data-app is an MQTT broker, which means that the NIPC gateway must act as an MQTT client. The MQTT broker configuration data, such as the URI, credentials and MQTT topic must also be provided in the registration API.

* webhook: The data-app supports a webhook the gateway can publish to. The registration API has to include the webhook URI & credentials.

* websocket: The data-app supports a websocket interface. The registration API has to include the websocket URI & credentials.

3.2.1. Register a data application

Method: POST /registrations/data-apps{?dataAppId}

Description: Registers a data application with the gateway.

Query Parameters:

* dataAppId: the ID of the data application

Request Body:

* events: a list of events that the data application is authorized for.

* mqttClient: a boolean that denotes that the data application is an MQTT client.

* mqttBroker: The data app is an MQTT broker. This object contains the MQTT broker information:

  - URI: the URI of the MQTT broker.

  - username: the username to authenticate with the MQTT broker.

  - password: the password to authenticate with the MQTT broker.

  - brokerCACert: the base64-encoded CA certificate of the MQTT broker (optional).

  - customTopic: By default, the topic will be composed of data-app/<dataAppId>/<namespace>/<json_pointer_to_sdf_event>, as described in Section 4.2. In this attribute, a custom topic can be configured (optional).

* webhook: The data app supports a webhook. This object contains a webhook URL along with any credentials that are required to authenticate the webhook.

  - URI: the webhook URI. The URI MUST include the scheme used by the webhook server (e.g., "https://"). It is up to the implementation to support different schemes. If a scheme is not supported, the NIPC gateway MUST return an error response with type "https://www.iana.org/assignments/nipc-problem-types#unsupported-uri-scheme".

  - headers: An object that contains the headers to be sent with the webhook request. The headers can contain any authentication information required by the webhook server. Each header is represented as a key-value pair in the object.

  - serverCACert: the CA certificate of the webhook server, encoded as per [RFC7468] and newlines encoded as '\n' (optional)

* websocket: The data app supports a websocket. This object contains a websocket URL along with any credentials that are required to authenticate the websocket. The websocket URL is the endpoint where the streaming data will be sent.

  - URI: the websocket URI. The URI MUST include the scheme used by the websocket server (e.g., "wss://"). It is up to the implementation to support different schemes. If a scheme is not supported, the NIPC gateway MUST return an error response with type "https://www.iana.org/assignments/nipc-problem-types#unsupported-uri-scheme".

  - headers: An object that contains the headers to be sent with the websocket request. The headers can contain any authentication information required by the websocket server. Each header is represented as a key-value pair in the object.

  - serverCACert: the CA certificate of the websocket server, encoded as per [RFC7468] and newlines encoded as '\n' (optional)

```
<CODE BEGINS> file "data_app.cddl"
DataApp = {
  events: [* EventRef],
  ( DataAppMqttClient //
    DataAppMqttBroker //
    DataAppWebhook //
    DataAppWebsocket )
}

EventRef = {
  event: text    ; SDF global name (absolute URI with fragment)
}

DataAppMqttClient = {
  mqttClient: bool
}

DataAppMqttBroker = {
  mqttBroker: {
    URI: text,
    username: text,
    password: text,
    ? brokerCACert: text,   ; PEM-encoded CA certificate
    ? customTopic: text     ; optional custom MQTT topic
  }
}

DataAppWebhook = {
  webhook: {
    URI: text,
    ? headers: { * text => text },  ; key/value headers
    ? serverCACert: text
  }
}

DataAppWebsocket = {
  websocket: {
    URI: text,
    ? headers: { * text => text },  ; key/value headers
    ? serverCACert: text
  }
}
<CODE ENDS>
```

Example of a request body:

```
<CODE BEGINS>
================ NOTE: '\' line wrapping per RFC 8792 ================

{
  "events": [
    {
      "event": "https://example.com/heartrate#/sdfObject/healthsenso\
r/sdfEvent/fallDetected"
    }
  ],
  "mqttClient": true
}
<CODE ENDS>
```

Figure 10: Example with mqttClient

Example of a request body for a data application that is an MQTT broker:

```
<CODE BEGINS>
================ NOTE: '\' line wrapping per RFC 8792 ================

{
  "events": [
    {
      "event": "https://example.com/heartrate#/sdfObject/healthsenso\
r/sdfEvent/fallDetected"
    }
  ],
  "mqttBroker": {
    "URI": "mqtt.example.com:1883",
    "username": "user",
    "password": "password",
    "customTopic": "custom/topic"
  }
}
<CODE ENDS>
```

Figure 11: Example with mqttBroker

Response:

* If successful, the response will be identical to the request body.

3.2.2. Update a data application

Method: PUT /registrations/data-apps{?dataAppId}

Description: Updates a data application registration.

Query Parameters:

* dataAppId: the ID of the data application

Request Body:

* The request body is identical to the request body for the register data application API Section 3.2.1.

Response:

* If successful, the response will be identical to the request body.

3.2.3. Get a data application

Method: GET /registrations/data-apps{?dataAppId}

Description: Gets a data application object registered with the gateway.

Query Parameters:

* dataAppId: the ID of the data application

Response:

The response will be identical to the request body for the register data application API Section 3.2.1.

3.2.4. Delete a data application

Method: DELETE /registrations/data-apps{?dataAppId}

Description: Deletes a data application registered with the gateway.

Query Parameters:

* dataAppId: the ID of the data application

Response:

* If successful, the response will be identical to the request body for the register data application API Section 3.2.1.

4. NIPC Operation APIs

The NIPC operation APIs perform protocol-neutral interactions on SDF affordances, properties, events, and actions. This allows applications to read and update device properties, invoke actions, and consume events.

NIPC defines three API collections aligned with the SDF Affordances defined in Section 1.2 of [RFC9880]:

* Properties: read and write device properties.

* Events: enable and disable device event reporting.

* Actions: invoke device actions.

Additionally, NIPC defines one more API collection that allows applications to install triggers on events. Triggers will trigger an action if the event is executed. Trigger collection:

* Triggers: Install a trigger on an event (invokes an action)

To invoke NIPC operations APIs on a device, one or more SDF models MUST be registered for that device. The SDF model MAY have a top-level sdfThing (with multiple sdfObjects) or a top-level sdfObject. Operations depend on affordances (sdfProperty, sdfEvent, sdfAction) defined in the registered SDF model and on a device ID (see [I-D.ietf-scim-device-model]). Affordances are referenced by their SDF global name (absolute URI with fragment) as described in Section 4 of [RFC9880].

The NIPC Gateway must match the SDF global name against the registered SDF model to resolve the protocol mapping (protocolmap) the gateway will execute. When carried in a URI, the SDF global name MUST be percent-encoded per Section 2.1 of [RFC3986].

4.1. NIPC Property APIs

These APIs allow applications to read and update device properties. If the underlying protocol requires a connection, the gateway establishes it implicitly for the operation; when an explicit connection is already active, operations reuse it without modification.

Requests and responses support content negotiation via Content-Type and Accept. When using "application/nipc+json", payloads must follow the examples above. Binary property values must be base64-encoded with padding per Section 5 of [RFC4648] in the "value" field. For other media types, payload semantics must follow the selected media type.

4.1.1. Update one or multiple values

Method: PUT /devices/{id}/properties{?propertyName}

Description: Write values to one or more properties on a device

Parameters:

* id: the ID of the device

Query Parameters:

* propertyName: Identifies a single property to update. If present, the request body MAY use any media type appropriate to the property payload.

-or-

* If absent, the request body MUST be application/nipc+json and contain an array of update items, each with a property and a value.

Request Body:

* If the query parameter propertyName is provided, the request body MAY use any media type appropriate to the property payload. The value is encoded as per the content type of the payload.

-or-

* If the query parameter propertyName is NOT provided, the request body must be an array of properties to update, each containing a property and a value. The value attribute contains the raw binary data, which must be encoded in base64 with padding as per Section 5 of [RFC4648].

```
<CODE BEGINS> file "property_value_array.cddl"
PropertyValueArray = [* PropertyValue]

; Minimal PropertyValue shape (matches allOf of Property + Value)
PropertyValue = {
  property: text,        ; SDF global name of the property
  value: b64text         ; base64-encoded bytes (RFC 4648 Section 5)
}

; Helper type for base64-with-padding encoded text
b64text = text
<CODE ENDS>
```

Example body for updating multiple properties:

```
<CODE BEGINS>
=============== NOTE: '\' line wrapping per RFC 8792 ================

[
  {
    "property": "https://example.com/heartrate#/sdfObject/thermostat\
/sdfProperty/temperature",
    "value": "dGVzdA=="
  },
  {
    "property": "https://example.com/heartrate#/sdfObject/thermostat\
/sdfProperty/humidity",
    "value": "eGVzdB=="
  }
]
<CODE ENDS>
```

Figure 12: Example updating multiple properties

Response:

* If the Accept header is set to application/nipc+json, the response must be an array with a status field set to 200 for each property that was updated, or a problem type object for each property that failed to update. The "properties" array must be an array of properties that were updated, each containing a property and a value.

-or-

* If the Accept header is set to any other media type and the propertyName query parameter is provided, the response must be 204 No Content with no body.

```
<CODE BEGINS> file "property_value_response_array.cddl"
=============== NOTE: '\' line wrapping per RFC 8792 ================

PropertyValueResponseArray = [* PropertyValueResponseArrayItem]

PropertyValueResponseArrayItem = ( SuccessResponse // FailureRespons\
e )

; Minimal success shape (may be extended)
SuccessResponse = {
  ? status: uint
}
<CODE ENDS>
```

Example of a response:

```
<CODE BEGINS>
=============== NOTE: '\' line wrapping per RFC 8792 ================

[
  {
    "status": 200
  },
  {
    "type": "https://www.iana.org/assignments/nipc-problem-types#inv\
alid-property",
    "status": 400,
    "title": "Invalid Property",
    "detail": "Property https://example.com/heartrate#/sdfObject/the\
rmostat/sdfProperty/temperature does not exist or is not writable"
  }
]
<CODE ENDS>
```

Figure 13: Example update multiple properties response

4.1.2. Read one or multiple values

Method: GET /devices/{id}/properties{?propertyName*}

Description: Read values from one or more properties on a device

Parameters:

* id: the ID of the device

Query Parameters:

* propertyName: The property to read. This can be a single property or multiple properties. If multiple properties are provided, the request body MUST contain an application/nipc+json payload with an array of properties to read.

Response:

* If the Accept header is set to application/nipc+json, the response must be an array of properties, each containing a property and a value. The value must be the raw binary data read from the property, encoded in base64 with padding as per Section 5 of [RFC4648]. The array must contain objects with 2 attributes: - property: The property that was read. - value: The bytes that were read in base64 encoding

-or-

* If the Accept header is set to any other media type and a single propertyName query parameter is provided, the request body MAY use any media type appropriate to the property payload. The value is encoded as per the content type of the payload.

<CODE BEGINS> file "property_value_read_response_array.cddl"
=============== NOTE: '\' line wrapping per RFC 8792 ================

```
PropertyValueReadResponseArray = [* PropertyValueReadResponseArrayIt\
em]

PropertyValueReadResponseArrayItem = ( PropertyValue // FailureRespo\
nse )
```

<CODE ENDS>

Example of a response:

```
<CODE BEGINS>
================ NOTE: '\' line wrapping per RFC 8792 ================

[
  {
    "property": "https://example.com/heartrate#/sdfObject/thermostat\
/sdfProperty/temperature",
    "value": "dGVzdA=="
  },
  {
    "property": "https://example.com/heartrate#/sdfObject/thermostat\
/sdfProperty/humidity",
    "value": "eGVzdB=="
  }
]
<CODE ENDS>
```

Figure 14: Example read multiple properties response

4.2. NIPC Event APIs

Event APIs enable or disable reporting of device events. For certain protocols, a connection may be required. If the underlying protocol requires a connection, the gateway establishes it implicitly for the operation. If an explicitly created connection is already active, it is reused without modification.

Events are referenced by the SDF global name of an sdfEvent. The {id} path segment identifies a device or a group of devices. A group event MAY be enabled only if the underlying protocol supports group activation (e.g., BLE advertisement or connection status events).

Events are delivered to registered data-apps over a publish/subscribe interface, as defined in Section 7. If the data application registered for this event is an MQTT broker or client, the event SDF global name may be used to construct the MQTT topic for the event. The topic is constructed using the data application ID, the default namespace for the event, and the event itself. For example, if the data application ID is "0927ce7c-b258-4bfa-a345-bcc9f74385b4" and the event is "https://example.com/thermometer#/sdfThing/thermometer/sdfEvent/isPresent", the topic will be:

```
================ NOTE: '\' line wrapping per RFC 8792 ================

data-app/<dataAppId>/<namespace>/<json_pointer_to_sdf_event>

data-app/0927ce7c-b258-4bfa-a345-bcc9f74385b4/thermometer/sdfThing/\
thermometer/sdfEvent/isPresent
```

A data application may subscribe to this topic using the topic or it may use MQTT wildcards to subscribe to data-app/+/temperature/# to receive all events for the temperature namespace.

If a customTopic was supplied in the data-app registration (mqttBroker case), that topic MUST be used instead of the constructed default.

4.2.1. Enable event reporting

Method: POST /devices/{id}/events{?eventName}

Description: Enables an event on a device

Parameters:

* id: the ID of the device

Query Parameters:

* eventName: the event to enable. The eventName must be a URL encoded string that is the absolute URI that is the global name of an sdfEvent.

Response:

* Returns HTTP status code 201 Created with a Location header pointing to the created event instance.

Example of a successful response:

```
================ NOTE: '\' line wrapping per RFC 8792 ================

HTTP/1.1 201 Created
Location: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/events?insta\
nceId=02ee282c-8915-4b2e-bbd2-88966773134a
```

The Location header must contain the URI for the created event instance, which may be used to check status or disable the event.

4.2.2. Disable event reporting

Method: DELETE /devices/{id}/events{?instanceId}

Description: Disables an event on a device

Parameters:

* id: the ID of the device or group of devices

Query Parameters:

* instanceId: the instance ID of the event to disable (obtained from the Location header when the event was enabled)

Response:

* Returns HTTP status code 204 No Content on successful disable.

HTTP/1.1 204 No Content

4.2.3. Get status of one or more events

Method: GET /devices/{id}/events{?instanceId*}

Description: Get the status of one or more events on a specific device

Parameters:

* id: the ID of the device or group of devices

Query Parameters:

* instanceId: a comma separated list of event instance IDs to filter by (optional)

Response: The response must be an array of events, each containing an instanceID and an event. - instanceId: must be the unique instance ID for each enabled event. - event: must be the event URI for each enabled event.

```
<CODE BEGINS> file "event_status_array.cddl"
=============== NOTE: '\' line wrapping per RFC 8792 ================

EventStatusResponseArray = [* EventStatusResponseArrayItem]

EventStatusResponseArrayItem = ( EventInstanceSuccess // FailureResp\
onse )

; Success item = { event, instanceId }
EventInstanceSuccess = {
  event: text,        ; SDF global name of the event (absolute URI w\
ith fragment)
  instanceId: text    ; UUID (as text)
}
<CODE ENDS>
```

Example of a response:

```
<CODE BEGINS>
=============== NOTE: '\' line wrapping per RFC 8792 ================

[
  {
    "instanceId": "02ee282c-8915-4b2e-bbd2-88966773134a",
    "event": "https://example.com/heartrate#/sdfObject/healthsensor/\
sdfEvent/fallDetected"
  }
]
<CODE ENDS>
```

Figure 15: Example get multiple events status response

4.2.4. Enable event reporting on a group of devices

Method: POST /groups/{id}/events

Description: Enables an event on a group of devices

Parameters:

* id: the ID of the group of devices

Query Parameters:

* eventName: the event to enable. The eventName is a URL encoded string that is the absolute URI that is the global name of an sdfEvent.

Response:

* The response must return HTTP status code 201 Created with a Location header pointing to the created event instance. The Location header must contain the URI for the created event instance, which can be used to check status or disable the event.

Example of a successful response:

=============== NOTE: '\' line wrapping per RFC 8792 ================

```
HTTP/1.1 201 Created
Location: /groups/0dc729d7-f6c3-491d-9b9d-e7176d2be243/events?instan\
ceId=f1b9f26b-21ce-4deb-9d57-289ba7e17cce
```

4.2.5. Disable event reporting on a group of devices

Method: DELETE /groups/{id}/events{?instanceId}

Description: Disables an event on a group of devices

Parameters:

* id: the ID of the group of devices

Query Parameters:

* instanceId: the instance ID of the event to disable (obtained from the Location header when the event was enabled)

Response:

MUST return 200 OK with an array of per-device event status entries. For each device where the event was successfully disabled, the entry MUST include deviceId and event (SDF global name). For each device where disabling failed, the entry MUST be a Problem Details error object for that device.

```
<CODE BEGINS> file "group_event_status_response_array.cddl"
=============== NOTE: '\' line wrapping per RFC 8792 ================

GroupEventStatusResponseArray = [* GroupEventStatusResponse]

GroupEventSuccessResponse = { event: text, deviceId: text }

; Each item is either an event+deviceId success or a GroupFailureRes\
ponse
GroupEventStatusResponse = (GroupEventSuccessResponse // GroupFailur\
eResponse)

GroupFailureResponse = {
  FailureResponse,
  ? deviceId: text
}
<CODE ENDS>
```

Example of a response:

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

[
  {
    "event": "https://example.com/heartrate#/sdfObject/healthsensor/\
sdfEvent/fallDetected",
    "deviceId": "1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30"
  },
  {
    "event": "https://example.com/heartrate#/sdfObject/healthsensor/\
sdfEvent/fallDetected",
    "deviceId": "d62c7fb2-a216-4811-a388-053b17fdbedc"
  },
  {
    "event": "https://example.com/heartrate#/sdfObject/healthsensor/\
sdfEvent/fallDetected",
    "deviceId": "01b52a23-b98c-454c-ba9e-086a43bdfd79"
  },
  {
    "type": "https://www.iana.org/assignments/nipc-problem-types#eve\
nt-not-enabled",
    "status": 400,
    "title": "Event Not Enabled",
    "deviceId": "9171ec16-e3c1-4ccf-ad23-b92a1a3f069d",
    "detail": "Failed to disable the event for device 9171ec16-e3c1-\
    4ccf-ad23-b92a1a3f069d"
  }
]
```

4.2.6. Get event status on a group of devices

Method: GET /groups/{id}/events{?instanceId}

Description: Get the status of one or more events for a group of devices

Parameters:

* id: the ID of the group of devices

Query Parameters:

* instanceId: the instance ID of the event (obtained from the Location header when the event was enabled).

Response:

MUST return 200 OK with an array of per-device event status entries. For each device where the event was successfully enabled, the entry MUST include deviceId and event (SDF global name). For each device where enabling failed, the entry MUST be a Problem Details error object for that device.

<CODE BEGINS> file "group_event_status_response_array.cddl"

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

GroupEventStatusResponseArray = [* GroupEventStatusResponse]

GroupEventSuccessResponse = { event: text, deviceId: text }

; Each item is either an event+deviceId success or a GroupFailureRes\
ponse
GroupEventStatusResponse = (GroupEventSuccessResponse // GroupFailur\
eResponse)

GroupFailureResponse = {
  FailureResponse,
  ? deviceId: text
}
```

<CODE ENDS>

Example of a response:

```
<CODE BEGINS>
=============== NOTE: '\' line wrapping per RFC 8792 ================

[
  {
    "event": "https://example.com/heartrate#/sdfObject/healthsensor/\
sdfEvent/fallDetected",
    "deviceId": "1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30"
  },
  {
    "event": "https://example.com/heartrate#/sdfObject/healthsensor/\
sdfEvent/fallDetected",
    "deviceId": "d62c7fb2-a216-4811-a388-053b17fdbedc"
  },
  {
    "event": "https://example.com/heartrate#/sdfObject/healthsensor/\
sdfEvent/fallDetected",
    "deviceId": "01b52a23-b98c-454c-ba9e-086a43bdfd79"
  },
  {
    "type": "https://www.iana.org/assignments/nipc-problem-types#eve\
nt-not-enabled",
    "status": 400,
    "title": "Event Not Enabled",
    "deviceId": "9171ec16-e3c1-4ccf-ad23-b92a1a3f069d",
    "detail": "Failed to disable the event for device 9171ec16-e3c1-\
4ccf-ad23-b92a1a3f069d"
  }
]
<CODE ENDS>
```

Figure 16: Example get multiple group events status response

4.3. NIPC Action APIs

NIPC Action APIs invoke device actions. If the underlying protocol requires a connection, the gateway establishes it implicitly for the operation. If an explicitly established connection is already active, the operation MUST reuse it without modification.

4.3.1. Perform an action

Method: POST /devices/{id}/actions{?actionName}

Description: Perform an action on a specific device

Parameters:

* id: the ID of the device

Query Parameters:

* actionName: the action to perform

Request Body:

The request body is optional and may contain a value. The media type of the value can be defined by the underlying protocol, for example it could be octet-stream for binary data.

Response:

Actions are performed asynchronously. A successful request returns HTTP status code 202 Accepted with a Location header pointing to the action instance for status checking. The Location header contains the URI for the action instance, which can be used to check the action status.

Example of a successful response:

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

HTTP/1.1 202 Accepted
Location: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/actions?inst\
anceId=02ee282c-8915-4b2e-bbd2-88966773134a
```

4.3.2. Check action status

Method: GET /devices/{id}/actions{?instanceId}

Description: Check the status of an action on a specific device

Parameters:

* id: the ID of the device

Query Parameters:

* instanceId: the instance ID of the action (obtained from the Location header)

Response: MUST return 200 OK with an action status, which may be "in progress" or "completed".

```
<CODE BEGINS> file "action_response.cddl"
ActionResponse = {
  status: ActionStatus
}

ActionStatus = "IN_PROGRESS" / "COMPLETED"
<CODE ENDS>
```

Example of a response:

```
<CODE BEGINS>
{
  "status": "COMPLETED"
}
<CODE ENDS>
```

Figure 17: Example action status response

4.4. NIPC Trigger APIs

Triggers APIs do not actually execute an operation on a device or group of devices, but install a trigger that registers an operation. When triggered the registered operation gets executed.

Triggers can be installed for devices or groups of devices, represented by their id as a path parameter.

The trigger must always be triggered by an event. the global name of the event must be passed as a path parameter.

The triggered operation must always be an action. Since an action can be executed against both a device and a group, the trigger API also supports actions on both devices and groups of devices. The action is defined by its full NIPC URI to be executed, for example: "/devices/3171ec43-42a5-4415-ab4b-afd0dfbe9615/actions?actionName=https://example.com/AlarmSystem#/sdfObject/bell/sdfAction/ring"

If a NIPC Gateway supports multiple protocols, then a trigger can be defined on a devices that supports one protocol and triggers an action on a device that supports a different protocol. As an example, an event on a BLE device can trigger an action on a Zigbee device.

4.4.1. Create a trigger on a device

Method: POST /devices/{id}/triggers{?sdfName}

Description: Creates a trigger on an affordance of a device. A trigger will trigger an action on another device or a group of devices.

Parameters:

* id: the ID of the device

Query Parameters:

* sdfName: the sdfName of the affordance that is associated with the trigger. This must be an event. The sdfName must be a URL encoded string that is the absolute URI that is the global name of an sdfEvent.

Request Body:

* The request body must be of type 'application/nipc+json', and contain an action object. The action object contains a NIPC URI to be executed, including the device or group id and actionname percent-encoded query parameter.

```
<CODE BEGINS> file "action.cddl"
Action = {
  ? action: text  ; NIPC action operation to execute
}
<CODE ENDS>
```

Example body for trigger creation:

```
<CODE BEGINS>
=============== NOTE: '\' line wrapping per RFC 8792 ================

{
  "action": "/devices/3171ec43-42a5-4415-ab4b-afd0dfbe9615/actions?a\
ctionName=https%3A%2F%2Fexample.com%2FAlarmSystem%23%2FsdfObject%2Fb\
ell%2FsdfAction%2Fring"
}
<CODE ENDS>
```

Figure 18: Example trigger creation

Response:

* Returns HTTP status code 201 Created with a Location header pointing to the created trigger instance.

Example of a successful response:

```
================ NOTE: '\' line wrapping per RFC 8792 ================

HTTP/1.1 201 Created
Location: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/triggers?ins\
tanceId=02ee282c-8915-4b2e-bbd2-88966773134a
```

The Location header must contain the URI for the created trigger instance, which may be used to check status or disable the trigger.

4.4.2. Delete a trigger on a device

Method: DELETE /devices/{id}/triggers{?instanceId}

Description: Deletes an installed trigger

Parameters:

* id: the ID of the device

Query Parameters:

* instanceId: the instance ID of the trigger to disable (obtained from the Location header when the trigger was installed)

Response:

* Returns HTTP status code 204 No Content on successful delete.

```
HTTP/1.1 204 No Content
```

4.4.3. Get installed triggers for a device

Method: GET /devices/{id}/triggers{?instanceId}

Description: Get the installed triggers for a devices, or a specific trigger when specifying an instanceId.

Parameters:

* id: the ID of the device

Query Parameters:

* instanceId: an InstanceID, or comma separated list of event instance IDs to filter by (optional)

* If no query parameter is supplied, then all triggers for the device must be retrieved

Response:

The response must be an array of triggers, each containing an instanceId, the sdfName of an event and an action.

* instanceId: must be the instance ID for each installed trigger, as returned by the NIPC GW when the trigger was created.

* sdfName: must be the sdfName of the event that is associated with the trigger.

* action: must be the NIPC URI of the action to be executed when the trigger is activated.

<CODE BEGINS> file "trigger_status_array.cddl"

```
; Trigger status response array and item shape

TriggerStatusResponseArray = [* TriggerResponse]

TriggerResponse = {
  SdfReference,
  ? action: text,      ; NIPC action operation to execute
  instanceId: text     ; UUID (as text)
}
```

<CODE ENDS>

Example of a response:

<CODE BEGINS>

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

[
  {
    "instanceId": "02ee282c-8915-4b2e-bbd2-88966773134a",
    "sdfName": "https://example.com/heartrate#/sdfObject/healthsenso\
r",
    "action": "/devices/3171ec43-42a5-4415-ab4b-afd0dfbe9615/actions\
?actionName=https%3A%2F%2Fexample.com%2FAlarmSystem%23%2FsdfObject%2\
Fbell%2FsdfAction%2Fring"
  }
]
```

<CODE ENDS>

Figure 19: Example get multiple triggers response

4.4.4. Create a trigger on a group of devices

Method: POST /groups/{id}/triggers{?sdfName}

Description: Creates a trigger on an affordance of a group of devices. A trigger will trigger an action on another device or a group of devices.

Parameters:

* id: the ID of the group of devices

Query Parameters:

* sdfName: the sdfName of the affordance that is associated with the trigger. This must be an event. The sdfName must be a URL encoded string that is the absolute URI that is the global name of an sdfEvent.

Request Body:

* The request body must be of type 'application/nipc+json', and contain an action object. The action object contains a NIPC URI to be executed, including the device or group id and actionname percent-encoded query parameter.

```
<CODE BEGINS> file "action.cddl"
Action = {
  ? action: text  ; NIPC action operation to execute
}
<CODE ENDS>
```

Example body for trigger creation:

```
<CODE BEGINS>
=============== NOTE: '\' line wrapping per RFC 8792 ================

{
  "action": "/groups/3171ec43-42a5-4415-ab4b-afd0dfbe9615/actions?ac\
tionName=https%3A%2F%2Fexample.com%2FAlarmSystem%23%2FsdfObject%2Fbe\
ll%2FsdfAction%2Fring"
}
<CODE ENDS>
```

Figure 20: Example trigger creation

Response:

* Returns HTTP status code 201 Created with a Location header pointing to the created trigger instance.

Example of a successful response:

=============== NOTE: '\' line wrapping per RFC 8792 ================

```
HTTP/1.1 201 Created
Location: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/triggers?ins\
tanceId=02ee282c-8915-4b2e-bbd2-88966773134a
```

The Location header must contain the URI for the created trigger instance, which may be used to check status or disable the trigger.

4.4.5. Delete a trigger on a group of devices

Method: DELETE /groups/{id}/triggers{?instanceId}

Description: Deletes an installed trigger

Parameters:

* id: the ID of the group of devices

Query Parameters:

* instanceId: the instance ID of the trigger to disable (obtained from the Location header when the trigger was installed)

Response:

* Returns HTTP status code 204 No Content on successful delete.

```
HTTP/1.1 204 No Content
```

4.4.6. Get installed triggers for a group of devices

Method: GET /groups/{id}/triggers{?instanceId}

Description: Get the installed triggers for a group of devices, or a specific trigger when specifying an instanceId.

Parameters:

* id: the ID of the group of devices

Query Parameters:

* instanceId: an InstanceID, or comma separated list of event instance IDs to filter by (optional)

* If no query parameter is supplied, then all triggers for the group of devices must be retrieved

Response:

The response must be an array of triggers, each containing an instanceId, the sdfName of an affordance and an action. - deviceId: must be the device ID for each installed trigger. - sdfName: must be the sdfName of the affordance (event or action) that is associated with the trigger. - action: must be the NIPC URI of the action to be executed when the trigger is activated.

<CODE BEGINS> file "group_trigger_status_array.cddl"
=============== NOTE: '\' line wrapping per RFC 8792 ================

```
; Group Trigger status response array and item shape

GroupTriggerStatusResponseArray = [* GroupTriggerResponse]

GroupTriggerResponse = (GroupTriggerSuccessResponse // GroupTriggerF\
ailureResponse)

GroupTriggerSuccessResponse = {
  SdfReference,
  ? action: text,      ; NIPC action operation to execute
  deviceId: text     ; UUID (as text)
}

GroupTriggerFailureResponse = {
  FailureResponse,
  ? deviceId: text
}
```

<CODE ENDS>

Example of a response:

```
<CODE BEGINS>
================ NOTE: '\' line wrapping per RFC 8792 ================

[
  {
    "deviceId": "02ee282c-8915-4b2e-bbd2-88966773134a",
    "sdfName": "https://example.com/heartrate#/sdfObject/healthsenso\
r",
    "action": "/devices/3171ec43-42a5-4415-ab4b-afd0dfbe9615/actions\
?actionName=https%3A%2F%2Fexample.com%2FAlarmSystem%23%2FsdfObject%2\
Fbell%2FsdfAction%2Fring"
  }
]
<CODE ENDS>
```

Figure 21: Example get multiple group triggers response

4.5. NIPC explicit connection management APIs

Some protocols do not require explicit connection setup; for those protocols, the APIs in this section do not apply. For protocols that do require a connection (e.g., BLE), an NIPC gateway performs implicit connection management for individual operations (establish on demand; release on completion), so clients ordinarily need not manage connections. Clients MAY choose to explicitly establish and retain a connection to perform a sequence of operations that depends on intermediate results. This section specifies APIs for explicit connection lifecycle control. Examples use BLE.

4.5.1. Protocol Information Object

The protocol information object is used to define protocol specific parameters for connections. The protocol information object is protocol specific and defined in the protocol extensions. An example of where a protocol info object would be used is to return protocol specific connection parameters when making connections, for example a BLE service map. An example of a BLE protocol information object is shown below.

This specification defines both the BLE and the Zigbee protocol information objects for connections and broadcast messages.

```
<CODE BEGINS> file "protocolinfo.cddl"
================ NOTE: '\' line wrapping per RFC 8792 ================

; Top-level wrappers
protocol-info-service-map = (
  ? protocolInformation: ble-service-map / zigbee-service-map
```

```
)

protocol-info-broadcast = (
  ? protocolInformation: ble-broadcast / zigbee-broadcast
)

; BLE protocol information
ble-service-map = {
  ble: {
    ? services: [* ble-service],
    ? cached: bool,
    ? cacheExpiryDuration: int,
    ? autoUpdate: bool,
    ? bonding: bonding-type,
  }
}

bonding-type = "default" / "none" / "justworks" / "passkey" / "oob"

ble-service = {
  serviceID: uuid,
  ? characteristics: [* ble-characteristic],
}

ble-characteristic = {
  characteristicID: uuid,
  ? flags: [* ble-flag],
  ? descriptors: [* ble-descriptor],
}

ble-flag = "read" / "write" / "notify"

ble-descriptor = {
  descriptorID: uuid,
}

ble-broadcast = {
  ble: {
    ? connectable: bool,
  },
}

; Zigbee protocol information
zigbee-service-map = {
  zigbee: {
    ? endpoints: [* zigbee-endpoint],
  },
}
```

```
   zigbee-endpoint = {
     endpointID: uint,
     ? clusters: [* zigbee-cluster],
   }

   zigbee-cluster = {
     clusterID: uint,
     ? properties: [* zigbee-property],
   }

   zigbee-property = {
     attributeID: uint,
     propertyType: uint,
   }

   zigbee-broadcast = {
     zigbee: {
     },
   }

   ; Basic types
   uuid = tstr .regexp "(?i)^[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-\
   f]{4}-[0-9a-f]{12}$"
   <CODE ENDS>
```

4.5.2. Connect to a device

Method: POST /devices/{id}/connections

Description: Connect to a device

Parameters:

* id: the ID of the device

Request Body:

* Connection retry parameters

* A protocol info object representing the BLE service map to be discovered. In the case of BLE, if no protocol info object is included, service discovery is performed to discover all supported properties when connecting to a device. Optionally, service discovery may be limited to properties defined in the "ble" protocol extension. The services to be discovered can be added in an array. Property discovery can be buffered across connections, so the API also supports caching parameters.

```
<CODE BEGINS> file "connection.cddl"
ConnectionRequest = {
  ? retries: uint,
  ? protocol-info-service-map
}

ConnectionResponse = {
  id: text,  ; UUID of the connection
  ? protocol-info-service-map
}
<CODE ENDS>
```

Example body of a connection without specific discovery of properties:

```
<CODE BEGINS>
{
  "retries": 3
}
<CODE ENDS>
```

Figure 22: Example connection

where-

* "retries" defines the number of retries in case the operation does not succeed

In case the application would like to discover specific properties of a device, an additional protocol information object can be provided that defines what properties should be discovered.

Example body of a BLE connection with specific discovery of properties:

```
<CODE BEGINS>
{
  "retries": 3,
  "protocolInformation": {
    "ble": {
      "services": [
        {
          "serviceID": "00001809-0000-1000-8000-00805f9b34fb"
        }
      ],
      "cached": false,
      "cacheExpiryDuration": 3600,
      "autoUpdate": true,
      "bonding": "default"
    }
  }
}
<CODE ENDS>
```

Figure 23: Example connection with explicit discovery of connections

where in the BLE protocol object:

* "services" is an array of services defined by their serviceIDs.

* "cached" refers to whether the services need to be discovered for this connection. If cached is true, the services will be discovered for this connection only if it is not present in the cache. If cached is false, the services will be discovered for this connection. The services will be cached once it is discovered.

* "cacheExpiryDuration" defines how long (in seconds) the cache should be maintained before purging.

* some devices support notifications on changes in services, "autoUpdate" allows the network to update services based on notification (on by default)

* "bonding" allows you to override the bonding method configured in the device object. Possible values are default, none, justworks, passkey, oob. Default behavior is to use the bonding method defined in the device object.

Response:

Success responses includes an optional protocol information object with an array of discovered properties, as defined in the BLE protocol info object in section Section 4.5.1. This is an array of supported services, which in turn contains an array of characteristics, which in turn contains an array of descriptors, as shown in Figure 24.

```
    services
     - serviceID
        |
        |> characteristics
            - characteristicID
            - flags
               |
               |> Descriptors
                   - descriptorID
```

Figure 24: BLE Services

Example of a response:

```
<CODE BEGINS>
{
  "id": "1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30",
  "protocolInformation": {
    "ble": {
      "services": [
        {
          "serviceID": "00001809-0000-1000-8000-00805f9b34fb",
          "characteristics": [
            {
              "characteristicID":
                  "00002a1c-0000-1000-8000-00805f9b34fb",
              "flags": [
                "read",
                "write"
              ],
              "descriptors": [
                {
                  "descriptorID":
                      "00002902-0000-1000-8000-00805f9b34fb"
                }
              ]
            }
          ]
        }
      ]
    }
  }
}
<CODE ENDS>
```

Figure 25: Example connection response

where-

* "id" is the ID of the device

* "protocolInformation" contains an Array of BLE services as shown in Figure 24

A failure will generate a standard failed response. Please refer to Figure 5 definition of failed response.

4.5.3. Update a connection

Method: PUT /devices/{id}/connections

Description: Update a connection, for BLE this will update the cached ServiceMap for a device. Full service discovery will be performed, unless specific services are described in the API body.

Parameters:

* id: the ID of the device

Request Body:

* A protocol information object. In the case of BLE, if no protocol information is included, service discovery is performed to discover all supported properties when connecting to a device. Optionally, service discovery may be limited to properties defined in the "ble" protocol extension. The services to be discovered can be added in an array. Property discovery can be buffered across connections, so the API also supports caching parameters.

Example body of an update connection:

<CODE BEGINS>
```
{
  "protocolInformation": {
    "ble": {
      "services": [
        {
          "serviceID": "00001809-0000-1000-8000-00805f9b34fb"
        }
      ],
      "cached": false,
      "cacheExpiryDuration": 3600,
      "autoUpdate": true
    }
  }
}
```
<CODE ENDS>

Figure 26: Example service discovery response

where in the BLE protocol object:

* "services" is an array of services defined by their serviceIDs

* "cached" refers to whether the services need to be cached for subsequent connects, in order not to perform service discovery on each request

* "cacheExpiryDuration" defines how long the cache should be maintained before purging

* some devices support notifications on changes in services, "autoUpdate" allows the network to update services based on notification (on by default)

Response:

Success responses include a protocol information object with an array of discovered properties, as defined in the specific protocol. For example, for BLE, this is an array of supported services, which in turn contains an array of characteristics, which in turn contains an array of descriptors, as shown in Figure 24.

Example of a response:

<CODE BEGINS>

```
{
  "id": "1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30",
  "protocolInformation": {
    "ble": {
      "services": [
        {
          "serviceID": "00001809-0000-1000-8000-00805f9b34fb",
          "characteristics": [
            {
              "characteristicID":
                "00002a1c-0000-1000-8000-00805f9b34fb",
              "flags": [
                "read",
                "write"
              ],
              "descriptors": [
                {
                  "descriptorID":
                    "00002902-0000-1000-8000-00805f9b34fb"
                }
              ]
            }
          ]
        }
      ]
    }
  }
}
```

<CODE ENDS>

Figure 27: Example connection response

where-

* "id" is the ID of the device

* "protocolInformation" contains an Array of BLE services as shown in Figure 24 and described in Section 4.5.1.

A failure will generate a standard failed response. Please refer to Figure 5 definition of failed response.

4.5.4. Disconnect from a device

Method: DELETE /devices/{id}/connections

Description: Disconnect from a device

Parameters:

* id: the ID of the device

Response:

Returns HTTP status code 200 OK with device ID on successful disconnect.

Example of a response:

<CODE BEGINS>
```
{
  "id": "1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30"
}
```
<CODE ENDS>

Figure 28: Example disconnect response

where "id" is the ID of the device.

A failure will generate a standard failed response. Please refer to Figure 5 definition of failed response.

4.5.5. Get connection status

Method: GET /devices/{id}/connections

Description: Get connection status for a device. Success when device(s) is/are connected, includes service map for the device if available. Failure when a device is not connected.

Parameters:

* id: the ID of the device

Response:

Example of a response:

<CODE BEGINS>

```
{
  "id": "1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30",
  "protocolInformation": {
    "ble": {
      "services": [
        {
          "serviceID": "00001809-0000-1000-8000-00805f9b34fb",
          "characteristics": [
            {
              "characteristicID":
                "00002a1c-0000-1000-8000-00805f9b34fb",
              "flags": [
                "read",
                "write"
              ],
              "descriptors": [
                {
                  "descriptorID":
                    "00002902-0000-1000-8000-00805f9b34fb"
                }
              ]
            }
          ]
        }
      ]
    }
  }
}
```

<CODE ENDS>

Figure 29: Example connection status response

where-

* "id" is the ID of the device

* "protocolInformation" contains an Array of BLE services as shown in Figure 24

A failure will generate a standard failed response. Please refer to Figure 5 definition of failed response.

5. NIPC Extensibility

NIPC is designed to be extensible in two complementary ways:

* Protocol mappings: NIPC relies on SDF protocol mappings to relate protocol-neutral affordances to protocol-specific operations. Adding a new mapping enables support for additional device protocols (or versions) without the need to change the NIPC API itself. This allows deployments to evolve as device ecosystems change, while preserving application portability and gateway interoperability. Protocol mapping is an extension to SDF and described in [I-D.ietf-asdf-sdf-protocol-mapping].

* API extensions: Extensions compose multiple NIPC operations into a single request or introduce specialized flows optimized for common scenarios (e.g., bulk property updates, conditional reads, firmware operations). Extensions MUST use the “/extensions” path element and SHOULD focus on efficiency, latency reduction, and operational simplicity (fewer round trips, reduced state management). To ensure interoperability and consistent security review, extensions MUST be registered with IANA as defined in Section 11.2.

5.1. Protocol mappings

NIPC relies on SDF protocol mappings [I-D.ietf-asdf-sdf-protocol-mapping] to relate protocol-neutral affordances to protocol-specific operations. In [I-D.ietf-asdf-sdf-protocol-mapping] BLE and Zigbee are used as examples, however the mapping mechanism is extensible; so support for additional protocols (or versions) can be added without changing the NIPC schema or API.

5.2. API extensions

The extension APIs allow for extensibility of the APIs. Extension APIs may leverage the basic NIPC defined APIs and combine them in compound statements in order to streamline application operation against devices, make operations more expedient and convenient in one API call. An example of this is the bulk API extension. They may also introduce new functionality that is specific to a use case or protocol, such as the BLE transmit API.

Extensions must be defined under the /extensions path element. The extension name is defined as a path parameter after the /extensions path element. Extensions may define their own request and response payloads, as well as their own query parameters. Extensions must be IANA registered as defined in Section 11.2.

Extensions MAY implement long-running operations (e.g., firmware updates, bulk actions). For such operations, the server SHOULD respond with 202 Accepted and a Location header referencing a status URI. Clients MAY poll the status URI (GET) to obtain progress. While the operation is in progress, the status endpoint SHOULD return 200 OK with status information; upon completion, the server MAY redirect with 303 See Other to the final result resource, or return 200 OK with the completed result from the status endpoint. This pattern minimizes client state, supports retry, and provides a uniform mechanism for tracking asynchronous execution.

In the appendix Appendix D, we have defined a few example extensions.

6. NIPC Error Handling

Error types in NIPC APIs must use URI-based error type identifiers as defined in Section 11.6. The error types can be generic or specific to the API category. The error types are organized into the following categories:

* Generic: Broadly applicable errors, including authorization, invalid identifiers, and generic failures.

* Property APIs: Errors related to property APIs (read/write).

* Event APIs: Errors related to event APIs (enable/disable).

* Protocol specific: Errors related to protocol-specific operations.

* Extension APIs: Errors related to extension APIs.

NIPC error types are defined in the table below:

Error Type	Description	Category
invalid-id	Invalid device ID or gateway doesn’t recognize the ID	Generic
invalid-sdf-url	Invalid SDF URL or SDF affordance not found	Generic

extension-operation-not-executed	Operation was not executed since the previous operation failed	Generic
sdf-model-already-registered	SDF model already registered	Generic
sdf-model-in-use	SDF model in use	Generic
unsupported-uri-scheme	Unsupported URI scheme	Generic
property-not-readable	Property not readable	Property APIs
property-not-writable	Property not writable	Property APIs
event-already-enabled	Event already enabled	Event APIs
event-not-enabled	Event not enabled	Event APIs
event-not-registered	Event not registered for any data application	Event APIs
protocolmap-ble-already-connected	Device already connected	Protocol specific
protocolmap-ble-no-connection	No connection found for device	Protocol specific
protocolmap-ble-connection-timeout	BLE connection timeout	Protocol specific
protocolmap-ble-bonding-failed	BLE bonding failed	Protocol specific
protocolmap-ble-connection-failed	BLE connection failed	Protocol specific
protocolmap-ble-service-discovery-failed	BLE service discovery failed	Protocol specific
protocolmap-ble-invalid-service-or-	Invalid BLE service or characteristic ID	Protocol specific

characteristic		
protocolmap-zigbee-connection-timeout	Zigbee connection timeout	Protocol specific
protocolmap-zigbee-invalid-endpoint-or-cluster	Invalid Zigbee endpoint or cluster ID	Protocol specific
extension-transmit-invalid-data	Invalid transmit data	Transmit APIs
extension-firmware-rollback	Firmware rollback	Extension APIs
extension-firmware-update-failed	Firmware update failed	Extension APIs

Table 3: Error Codes

The appropriate HTTP status code is returned in the response.

7. Publish/Subscribe Interface

Events are delivered via a publish/subscribe interface. Event types include: (1) streaming data (enabled/disabled via the NIPC Events API), (2) broadcasts (e.g., advertisements), and (3) connection status (device link up/down). Event payloads are encoded in CBOR [RFC8949] and MAY be transported over MQTT, webhook, or websocket. CBOR is used because non-IP device payloads are typically binary; it encodes such data efficiently and is more compact than JSON, reducing bandwidth.

7.1. CDDL Definition

The event streaming format is defined here in CDDL [RFC8610]. A DataSubscription is a CBOR map containing the raw payload (bytes) and a timestamp (epoch seconds). It MAY include deviceID (the SCIM device identifier) when the payload is associated with a known device. Optional members (e.g., apMacAddress, rssi) MAY be present but can reveal deployment topology and SHOULD be omitted unless required. A choice group within DataSubscription indicates the event type (e.g., advertisement, subscription notification, connection status). An event publication MAY carry one or more DataSubscription entries in an array. Such an array is represented as DataBatch.

The subscription type choice uses a CDDL socket ($$subscription-extension) to allow extensibility. New subscription types registered with IANA (see Section 11.4) can be added using the CDDL plug syntax (//=) without modifying the base schema.

```
<CODE BEGINS> file "data_subscription.cddl"
start = DataBatch

DataBatch = [* DataSubscription]

DataSubscription = {
  ? data: bytes,
  timestamp: float, ; epoch in seconds
  ? deviceID: text,
  ? apMacAddress: text,
  subscription
}

; Subscription type - IANA registered types
subscription = (
  bleSubscription: BleSubscription //
  bleAdvertisement: BleAdvertisement //
  bleConnectionStatus: BleConnectionStatus //
  zigbeeSubscription: ZigbeeSubscription //
  $$subscription-extension
)

BleSubscription = {
  serviceID: text,
  characteristicID: text
}

BleAdvertisement = {
  macAddress: text,
  ? rssi: nint,
}

BleConnectionStatus = {
  macAddress: text,
  connected: bool,
  ? reason: int
}

ZigbeeSubscription = {
  endpointID: int,
  clusterID: int,
  attributeID: int,
  attributeType: int
}
<CODE ENDS>
```

7.2. CBOR Examples

This section contains a few examples of the DataSubscription struct in CBOR diagnostic notation.

```
[
  {
    "data": h'02011A020A0C16FF4C001007721F41B0392078',
    "deviceID": "1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30",
    "timestamp": 1727484393,
    "bleAdvertisement": {
        "macAddress": "C1:5C:00:00:00:01",
        "rssi": -25
    }
  }
]
```

Figure 30: Onboarded BLE Device Advertisement

```
[
  {
    "data": h'02011A020A0C16FF4C001007721F41B0392078',
    "timestamp": 1727484393,
    "bleAdvertisement": {
        "macAddress": "C1:5C:00:00:00:01",
        "rssi": -25
    }
  }
]
```

Figure 31: Non-Onboarded BLE Device Advertisement

```
[
  {
    "data": h'43463037434630373946303036',
    "deviceID": "1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30",
    "timestamp": 1727484393,
    "bleSubscription": {
        "serviceID": "a4e649f4-4be5-11e5-885d-feff819cdc9f",
        "characteristicID": "c4c1f6e2-4be5-11e5-885d-feff819cdc9f"
    }
  }
]
```

Figure 32: BLE GATT Notification

```
[
  {
    "deviceID": "1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30",
    "timestamp": 1727484393,
    "bleConnectionStatus": {
        "macAddress": "C1:5C:00:00:00:01",
        "connected": true
    }
  }
]
```

Figure 33: BLE Connection status event

```
[
  {
    "data": h'43463037434630373945303036',
    "deviceID": "1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30",
    "timestamp": 1727484393,
    "zigbeeSubscription": {
        "endpointID": 1,
        "clusterID": 6,
        "attributeID": 12,
        "attributeType": 1
    }
  }
]
```

Figure 34: Zigbee Attribute Notification

8. Examples

This section contains a few examples on how applications can leverage NIPC operations to communicate with BLE and Zigbee devices.

8.1. Property Read/Write

In this example, we will connect to a device and read and write from a property.

The sequence of operations for this are:

* Declare a device instance using the SCIM Interface (out of scope of this memo)

* Register an SDF model for the device

```
============== NOTE: '\' line wrapping per RFC 8792 ==============

POST /registrations/models
Content-Type: application/sdf+json
Accept: application/nipc+json
Host: localhost

{ ... }

HTTP/1.1 200 OK
content-type: application/nipc+json

[
  {
    "sdfName": "https://example.com/thermometer#/sdfThing/thermom\
eter"
  }
]
```

Request Body: JSON object with the SDF model, from Appendix F

* Read a property from the BLE device

```
============== NOTE: '\' line wrapping per RFC 8792 ==============

GET /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/properties?prop\
ertyName=https%3A%2F%2Fexample.com%2Fthermometer%23%2FsdfThing%2F\
thermometer%2FsdfProperty%2Fdevice_name
Accept: application/nipc+json
Host: localhost

HTTP/1.1 200 OK
content-type: application/nipc+json

[
  {
    "property": "https://example.com/thermometer#/sdfThing/thermo\
meter/sdfProperty/device_name",
    "value": "dGVzdA=="
  }
]
```

* Write to a property on the BLE device

```
============== NOTE: '\' line wrapping per RFC 8792 ==============

PUT /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/properties
Content-Type: application/nipc+json
Host: localhost

[
  {
    "property": "https://example.com/thermometer#/sdfThing/thermo\
meter/sdfProperty/device_name",
    "value": "dGVzdA=="
  }
]

HTTP/1.1 200 OK
content-type: application/nipc+json

[
  {
    "property": "https://example.com/thermometer#/sdfThing/thermo\
meter/sdfProperty/device_name",
    "value": "dGVzdA=="
  }
]
```

8.2. Enabling an Event on a Device

In this example, we will declare a device instance, and setup an advertisement subscription event for that device.

The sequence of operations for this are:

* Declaring a device instance and endpoint app using the SCIM Interface (out of scope of this memo)

* Register an SDF model for the device

```
============== NOTE: '\' line wrapping per RFC 8792 ==============

POST /registrations/models
Content-Type: application/sdf+json
Accept: application/nipc+json
Host: localhost

{ ... }

HTTP/1.1 200 OK
content-type: application/nipc+json

[
  {
    "sdfName": "https://example.com/thermometer#/sdfThing/thermom\
eter"
  }
]
```

Request Body: JSON object with the SDF model, from Appendix F

* Register the data app with the event

```
============== NOTE: '\' line wrapping per RFC 8792 ==============

POST /registrations/data-apps?dataAppId=0927ce7c-b258-4bfa-a345-\
bcc9f74385b4
Content-Type: application/nipc+json
Accept: application/nipc+json
Host: localhost

{
  "events": [
    "https://example.com/thermometer#/sdfThing/thermometer/sdfEve\
nt/isPresent"
  ],
  "mqttClient": true
}

HTTP/1.1 200 OK
content-type: application/nipc+json

{
  "events": [
    "https://example.com/thermometer#/sdfThing/thermometer/sdfEve\
nt/isPresent"
  ],
  "mqttClient": true
}
```

* Enable the advertisement event

```
============== NOTE: '\' line wrapping per RFC 8792 ==============

POST /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/events?eventNa\
me=https%3A%2F%2Fexample.com%2Fthermometer%23%2FsdfThing%2Fthermo\
meter%2FsdfEvent%2FisPresent
Host: localhost
Content-Length: 0

HTTP/1.1 201 Created
Location: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/events?in\
stanceId=02ee282c-8915-4b2e-bbd2-88966773134a
```

* Check the status of the event

```
============== NOTE: '\' line wrapping per RFC 8792 ==============

GET /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/events?instance\
Id=02ee282c-8915-4b2e-bbd2-88966773134a
Host: localhost

HTTP/1.1 200 OK
content-type: application/nipc+json

{
  "event": "https://example.com/thermometer#/sdfThing/thermometer\
/sdfEvent/isPresent"
}
```

8.3. Enabling an Event on a Group of Devices

In this example, we will enable an advertisement subscription event for a group of devices.

The sequence of operations for this are:

* Provision a device and endpoint app using the SCIM Interface (out of scope of this memo)

* Register an SDF model for the devices

```
============== NOTE: '\' line wrapping per RFC 8792 ==============

POST /registrations/models
Content-Type: application/sdf+json
Accept: application/nipc+json
Host: localhost

{ ... }

HTTP/1.1 200 OK
content-type: application/nipc+json

[
  {
    "sdfName": "https://example.com/thermometer#/sdfThing/thermom\
eter"
  }
]
```

Request Body: JSON object with the SDF model, from Appendix F

* Register the data app with the event

```
      ============== NOTE: '\' line wrapping per RFC 8792 ==============

      POST /registrations/data-apps?dataAppId=0927ce7c-b258-4bfa-a345-\
      bcc9f74385b4
      Content-Type: application/nipc+json
      Accept: application/nipc+json
      Host: localhost

      {
        "events": [
          "https://example.com/thermometer#/sdfThing/thermometer/sdfEve\
      nt/isPresent"
        ],
        "mqttClient": true
      }

      HTTP/1.1 200 OK
      content-type: application/nipc+json

      {
        "events": [
          "https://example.com/thermometer#/sdfThing/thermometer/sdfEve\
      nt/isPresent"
        ],
        "mqttClient": true
      }
```

* Enable the advertisement event on a group of devices

```
      ============== NOTE: '\' line wrapping per RFC 8792 ==============

      POST /groups/0dc729d7-f6c3-491d-9b9d-e7176d2be243/events?eventNa\
      me=https%3A%2F%2Fexample.com%2Fthermometer%23%2FsdfThing%2Fthermo\
      meter%2FsdfEvent%2FisPresent
      Host: localhost
      Content-Length: 0

      HTTP/1.1 201 Created
      Location: /groups/0dc729d7-f6c3-491d-9b9d-e7176d2be243/events?in\
      stanceId=f1b9f26b-21ce-4deb-9d57-289ba7e17cce
```

* Check the status of the event

```
============== NOTE: '\' line wrapping per RFC 8792 ==============

GET /groups/0dc729d7-f6c3-491d-9b9d-e7176d2be243/events?instance\
Id=f1b9f26b-21ce-4deb-9d57-289ba7e17cce
Host: localhost

HTTP/1.1 200 OK
content-type: application/nipc+json

[
  {
    "event": "https://example.com/heartrate#/sdfObject/healthsen\
sor/sdfEvent/fallDetected",
    "deviceId": "1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30"
  },
  {
    "event": "https://example.com/heartrate#/sdfObject/healthsen\
sor/sdfEvent/fallDetected",
    "deviceId": "d62c7fb2-a216-4811-a388-053b17fdbedc"
  },
  {
    "event": "https://example.com/heartrate#/sdfObject/healthsen\
sor/sdfEvent/fallDetected",
    "deviceId": "01b52a23-b98c-454c-ba9e-086a43bdfd79"
  },
  {
    "type": "https://www.iana.org/assignments/nipc-problem-types\
#event-not-enabled",
    "status": 400,
    "title": "Event Not Enabled",
    "deviceId": "9171ec16-e3c1-4ccf-ad23-b92a1a3f069d",
    "detail": "Failed to disable the event for device 9171ec16-e\
3c1-4ccf-ad23-b92a1a3f069d"
  }
]
```

9. Implementation Status

This section is to be removed before publishing as an RFC.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was

supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

9.1. TieDie IoT

Organization: Cisco Systems, North Carolina State University

Description: Open-source implementation of the NIPC APIs and gateway functionality for BLE. Client libraries and sample application functions for Python and Java are also provided.

Level of maturity: Open-source prototype

Coverage: All NIPC APIs for BLE protocol mapping

Version compatibility: All versions up to draft-17

Licensing: Apache License, Version 2.0

URL: https://github.com/iot-onboarding/tiedie

9.2. Cisco Sensor Connect for IoT Services (Catalyst)

Organization: Cisco Systems

Description: Commercial solution that delivers advanced BLE capabilities over Cisco Wireless infrastructure.

Level of maturity: Production

Coverage: All NIPC APIs for BLE protocol mapping

Version compatibility: draft-00

Licensing: Proprietary

URL: https://developer.cisco.com/docs/spaces-connect-for-iot-services/

9.3. Cisco Sensor Connect for IoT Services (Meraki)

Organization: Cisco Systems

Description: Commercial solution that delivers advanced BLE capabilities over Cisco Wireless infrastructure.

Level of maturity: Beta

Coverage: All NIPC APIs for BLE protocol mapping

Version compatibility: draft-17

Licensing: Proprietary

9.4. NIPC Prototype

Organization: Ericsson Research

Description: Prototype implementation in C of the NIPC API.

Level of maturity: Research prototype

Coverage: Registration and NIPC operations

Version compatibility: Draft-04

Licensing: Proprietary

Contact: Lorenzo Corneo lorenzo.corneo@ericsson.com (mailto:lorenzo.corneo@ericsson.com)

10. Security Considerations

10.1. Payload Encryption Considerations

Responses to NIPC operations requests may contain sensitive or confidential information. Therefore, application and device implementations should consider payload encryption. NIPC does not provide any payload encryption mechanism. If payload encryption is required, it MUST be provided by the underlying device protocol (e.g., BLE security modes) or by the transport-layer security mechanism (e.g., TLS).

10.2. TLS Support Considerations

NIPC MUST run on top of a transport-layer security mechanism such as TLS. When leveraging TLS, the NIPC gateway MUST support TLS 1.2 [RFC5246] and TLS 1.3 [RFC8446] and MAY support additional transport-layer mechanisms. When using TLS, the client MUST perform a TLS/SSL server identity check, per [RFC6125]. Implementation security considerations for TLS can be found in [RFC7525].

10.3. HTTP Considerations

NIPC runs on top of HTTP and is thus subject to the security considerations of HTTP Section 9 of [RFC7230].

10.4. Authorization Considerations

10.4.1. API authorization Considerations

To enable NIPC gateway functions, the network administrator MUST authorize applications (e.g., via exchange of tokens or public keys). Authorization MAY be role-based. The following baseline roles are RECOMMENDED:

* Provisioning: permitted to create and manage device and endpoint-app identities via SCIM (typically co-located with the gateway).

* Control: permitted to invoke NIPC property, action, and event APIs.

* Data: permitted to receive streamed event data. Deployments MAY further refine authorization at per-API or per-affordance granularity.

10.4.2. Authorization Token/Bearer Token/Cookie Considerations

When using authorization tokens such as those issued by OAuth 2.0 [RFC6749], implementers MUST take into account threats and countermeasures as documented in Section 8 of [RFC7521].

Since the possession of a bearer token, Authorization token, or cookie MAY authorize the holder to perform NIPC Operations on devices, tokens and cookies MUST contain sufficient entropy to prevent random guessing attack; for example, see Section 5.2 of [RFC6750] and Section 5.1.4.2.2 of [RFC6819].

As with all NIPC communications, bearer tokens and HTTP cookies MUST be exchanged using transport-layer security mechanism such as TLS.

Bearer tokens MUST have a limited lifetime that can be determined directly or indirectly (e.g., by checking with a validation service) by the application. By expiring tokens, applications are forced to obtain a new token (which usually involves re-authentication) for continued authorized access. For example, in OAuth 2.0, an application MAY use OAuth token refresh to obtain a new bearer token after authenticating to an authorization server. See Section 6 of [RFC6749]. As with bearer tokens, an HTTP cookie SHOULD last no longer than the lifetime of a browser session. An expiry time should be set that limits session cookie lifetime as per Section 5.2.1 of [RFC6265].

Implementations supporting OAuth bearer tokens need to factor in security considerations of this authorization method [RFC7521]. Implementers also need to consider authentication choices coupled with OAuth bearer tokens. For example, when using OAuth bearer tokens with client authentication via client credentials Section 4.4 of [RFC6749], implementers need to consider the security considerations of client authentication via client credentials as described in Section 3.2 of [RFC6819].

10.5. Other Security Considerations

* Preventing automated attacks: It is recommended to limit the number of requests that any particular application MAY make in a period of time.

* Logging and monitoring: It is recommended to log and monitor API usage to detect potential abuse or attacks.

* Input validation: It is recommended to validate all inputs to prevent injection attacks.

* Error handling: It is recommended to handle errors gracefully without exposing sensitive information.

* Least privilege: It is recommended to follow the principle of least privilege when granting access to resources.

* Storage and handling of sensitive data: Credentials MUST NOT be stored in clear-text, but MUST be stored using an encrypted protection mechanism (e.g., hashing).

11. IANA Considerations

This section provides guidance to the Internet Assigned Numbers Authority (IANA) regarding registration of values related to NIPC, in accordance with [RFC8126].

11.1. Media Type Registration

This document registers the "application/nipc+json" media type for messages of the NIPC APIs defined in this document carrying parameters encoded in JSON.

Type name: application

Subtype name: nipc+json

Required parameters: none

Optional parameters: none

Encoding considerations: Encoding considerations are identical to those specified for the "application/json" media type.

Security considerations: See the Section 10 section of this document.

Interoperability considerations: none

Published specification: This document, the NIPC API specification.

Applications that use this media type: Applications implementing NIPC APIs for IoT device management.

Fragment identifier considerations: none

Additional information:

* Deprecated alias names for this type: none
* Magic number(s): none
* File extension(s): none
* Macintosh file type code(s): none

Person & email address to contact for further information:

Intended usage: LIMITED USE

Restrictions on usage: To be used for NIPC APIs as defined in this document.

Author:

Change controller: IETF

11.2. API extensions

IANA is requested to create a new registry called "NIPC API extensions".

The registration policy for this registry is Specification Required [RFC8126].

The registry must contain following attributes:

* Extension URI
* Extension name
* Description
* Openapi model describing the extension. This model must be reviewed by an expert.

Following API extensions are described in this document:

Extension URI	Extension name	Description	Model reference
/extensions/{id}/bulk	Bulk API	Call multiple NIPC’s in a single request	Appendix D
/extensions/{id}/properties/file	File write API	Write a file with multiple property ops	Appendix D
/extensions/{id}/properties/blob	Binary write API	Write a binary blob with multiple property ops	Appendix D
/extensions/{id}/properties/read/conditional	Read conditional API	Read a property until a condition is fulfilled	Appendix D
/extensions/{id}/events/conditional	Event conditional API	Enable an event until a condition is fulfilled	Appendix D
/extensions/{id}/transmit	Transmit API	Transmits a payload to a device	Appendix D

Table 4

11.3. Well-known URIs

IANA is requested to register the following well-known URI in the "Well-Known URIs" registry as defined by [RFC8615]:

URI Suffix	Change Controller	Specification Document
nipc	IETF	This document, Section 2.5.1

Table 5

The well-known URI for NIPC is:

/.well-known/nipc

11.4. Data Subscription Types

IANA is requested to create a new registry called "NIPC Data Subscription Types".

This registry tracks the subscription types used in the NIPC publish/subscribe interface for streaming event data from devices. Each subscription type defines a specific data format for a particular protocol or use case.

The registration policy for this registry is Specification Required [RFC8126].

The registry must contain the following attributes:

* Type Name: The CDDL key name used in the subscription choice
* Description: A brief description of the subscription type
* CDDL Definition: Reference to the CDDL structure defining the subscription data
* Reference: Document defining the subscription type

IANA is requested to register the following initial entries:

Type Name	Description	CDDL Definition	Reference
bleSubscription	BLE GATT notification/ indication	BleSubscription	This document
bleAdvertisement	BLE advertisement data	BleAdvertisement	This document
bleConnectionStatus	BLE device connection status changes	BleConnectionStatus	This document
zigbeeSubscription	Zigbee attribute report subscription	ZigbeeSubscription	This document

Table 6

The CDDL definitions for these subscription types are provided in Section 7.1.

11.5. NIPC Protocols

IANA is requested to create a new registry called "NIPC Protocols".

This registry tracks the protocols used in the protocolInformation object for NIPC connection management and service discovery operations. Each protocol type defines a specific schema for protocol-specific information, as defined in Section 4.5.1.

The registration policy for this registry is Specification Required [RFC8126].

The registry must contain the following attributes:

* Protocol Name: The key name used in the protocolInformation object

* Description: A brief description of the protocol type

* Reference: Document defining the protocol type

IANA is requested to register the following initial entries:

Protocol Name	Description	Reference
ble	Bluetooth Low Energy	This document
zigbee	Zigbee	This document

Table 7

11.6. Problem Details for NIPC APIs

IANA is requested to create a new registry, the "NIPC Problem Type" registry, with following URL: https://www.iana.org/assignments/nipc-problem-types.

The registration policy for this registry is Specification Required [RFC8126].

Registrations MUST use the prefix "https://www.iana.org/assignments/nipc-problem-types#" for the type URI.

The registration requests MUST use the template defined in Section 4.2 of [RFC9457].

IANA is requested to register the following URIs in the "NIPC Problem Type" registry:

Problem Type URI	Description	Reference
https://www.iana.org/assignments/nipc-problem-types#invalid-id	Invalid device ID or gateway doesn’t recognize the ID	This document
https://www.iana.org/assignments/nipc-problem-types#invalid-sdf-url	Invalid SDF URL or SDF affordance not found	This document
https://www.iana.org/assignments/nipc-problem-types#extension-operation-not-executed	Operation was not executed since the previous operation failed	This document
https://www.iana.org/assignments/	SDF model	This

nipc-problem-types#sdf-model-already-registered	already registered	document
https://www.iana.org/assignments/nipc-problem-types#sdf-model-in-use	SDF model in use	This document
https://www.iana.org/assignments/nipc-problem-types#unsupported-uri-scheme	Unsupported URI scheme	This document
https://www.iana.org/assignments/nipc-problem-types#property-not-readable	Property not readable	This document
https://www.iana.org/assignments/nipc-problem-types#property-read-failed	Property read failed	This document
https://www.iana.org/assignments/nipc-problem-types#property-not-writable	Property not writable	This document
https://www.iana.org/assignments/nipc-problem-types#property-write-failed	Property write failed	This document
https://www.iana.org/assignments/nipc-problem-types#event-already-enabled	Event already enabled	This document
https://www.iana.org/assignments/nipc-problem-types#event-not-enabled	Event not enabled	This document
https://www.iana.org/assignments/nipc-problem-types#event-not-registered	Event not registered for any data application	This document
https://www.iana.org/assignments/nipc-problem-types#protocolmap-ble-already-connected	Device already connected	This document
https://www.iana.org/assignments/nipc-problem-types#protocolmap-ble-no-connection	No connection found for device	This document

https://www.iana.org/assignments/nipc-problem-types#protocolmap-ble-connection-timeout	BLE connection timeout	This document
https://www.iana.org/assignments/nipc-problem-types#protocolmap-ble-bonding-failed	BLE bonding failed	This document
https://www.iana.org/assignments/nipc-problem-types#protocolmap-ble-connection-failed	BLE connection failed	This document
https://www.iana.org/assignments/nipc-problem-types#protocolmap-ble-service-discovery-failed	BLE service discovery failed	This document
https://www.iana.org/assignments/nipc-problem-types#protocolmap-ble-invalid-service-or-characteristic	Invalid BLE service or characteristic ID	This document
https://www.iana.org/assignments/nipc-problem-types#protocolmap-zigbee-connection-timeout	Zigbee connection timeout	This document
https://www.iana.org/assignments/nipc-problem-types#protocolmap-zigbee-invalid-endpoint-or-cluster	Invalid Zigbee endpoint or cluster ID	This document
https://www.iana.org/assignments/nipc-problem-types#extension-transmit-invalid-data	Invalid transmit data	This document
https://www.iana.org/assignments/nipc-problem-types#extension-firmware-rollback	Firmware rollback	This document
https://www.iana.org/assignments/nipc-problem-types#extension-firmware-update-failed	Firmware update failed	This document

Table 8

Each Problem Type URI is intended for use as the "type" member in Problem Details responses as described.

12. References

12.1. Normative References

[I-D.ietf-asdf-sdf-protocol-mapping]
Mohan, R., Brinckman, B., and L. Corneo, "Protocol Mapping for SDF", Work in Progress, Internet-Draft, draft-ietf-asdf-sdf-protocol-mapping-04, 18 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-asdf-sdf-protocol-mapping-04>.

[I-D.ietf-scim-device-model]
Shahzad, M., Iqbal, H., and E. Lear, "Device Schema Extensions to the SCIM model", Work in Progress, Internet-Draft, draft-ietf-scim-device-model-18, 3 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-scim-device-model-18>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/info/rfc6125>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/info/rfc6570>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <https://www.rfc-editor.org/info/rfc6819>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7468] Josefsson, S. and S. Leonard, "Textual Encodings of PKIX, PKCS, and CMS Structures", RFC 7468, DOI 10.17487/RFC7468, April 2015, <https://www.rfc-editor.org/info/rfc7468>.

[RFC7521] Campbell, B., Mortimore, C., Jones, M., and Y. Goland, "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7521, DOI 10.17487/RFC7521, May 2015, <https://www.rfc-editor.org/info/rfc7521>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7525, DOI 10.17487/RFC7525, May 2015, <https://www.rfc-editor.org/info/rfc7525>.

[RFC7643] Hunt, P., Ed., Grizzle, K., Wahlstroem, E., and C. Mortimore, "System for Cross-domain Identity Management: Core Schema", RFC 7643, DOI 10.17487/RFC7643, September 2015, <https://www.rfc-editor.org/info/rfc7643>.

[RFC7644] Hunt, P., Ed., Grizzle, K., Ansari, M., Wahlstroem, E., and C. Mortimore, "System for Cross-domain Identity Management: Protocol", RFC 7644, DOI 10.17487/RFC7644, September 2015, <https://www.rfc-editor.org/info/rfc7644>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/info/rfc8615>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[RFC9457] Nottingham, M., Wilde, E., and S. Dalal, "Problem Details for HTTP APIs", RFC 9457, DOI 10.17487/RFC9457, July 2023, <https://www.rfc-editor.org/info/rfc9457>.

[RFC9562] Davis, K., Peabody, B., and P. Leach, "Universally Unique IDentifiers (UUIDs)", RFC 9562, DOI 10.17487/RFC9562, May 2024, <https://www.rfc-editor.org/info/rfc9562>.

[RFC9880] Koster, M., Ed., Bormann, C., Ed., and A. Kernen, "Semantic Definition Format (SDF) for Data and Interactions of Things", RFC 9880, DOI 10.17487/RFC9880, January 2026, <https://www.rfc-editor.org/info/rfc9880>.

12.2. Informative References

[BLE53] Bluetooth SIG, "Bluetooth Core Specification, Version 5.3", 2021.

[Gatt-REST-API] Bluetooth SIG, "A RESTful API used to access data in devices using the functionality defined in the Bluetooth GATT profile", 2017, <https://www.bluetooth.com/bluetooth-resources/gatt-rest-api/>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[Zigbee22] Connectivity Standards Alliance, "Zigbee Specification, Version 22 1.0", 2017.

Appendix A. OpenAPI definition

The following non-normative model is provided for convenience of the implementor.

<CODE BEGINS> file "openapi.yml"
=============== NOTE: '\\' line wrapping per RFC 8792 ===============

```
# yaml-language-server: $schema=https://json-schema.org/draft/2020-1\
\2/schema
openapi: 3.0.3
info:
  title: Non IP Device Control (NIPC) API
  description: |-
    This API specifies RESTful application layer interface for
    gateways providing operations against non-IP devices. The
    described interface is extensible. The examples includes
    leverage Bluetooth Low Energy and Zigbee as they are commonly
    deployed.
```

```
    termsOfService: http://swagger.io/terms/
    contact:
      email: bbrinckm@cisco.com
    license:
      name: TBD
      url: TBD
    version: 0.17.0
  externalDocs:
    description: NIPC IETF draft
    url: https://datatracker.ietf.org/doc/draft-ietf-asdf-nipc/
  servers:
    - url: "{gw_host}/nipc/draft-17"
      variables:
        gw_host:
          default: localhost
          description: Gateway Host
  tags:
    - name: NIPC property APIs
      description: |-
        APIs that allow apps to get and update device properties.
        If the underlying protocol requires connection management, it
        will be performed as part of the API call.
    - name: NIPC event APIs
      description: |-
        APIs that allow apps to enable or disable event reporting on
        devices. If the underlying protocol requires connection
        management, it will be performed as part of the API call.
    - name: NIPC action APIs
      description: |-
        APIs that perform actions on devices or groups.
    - name: NIPC trigger APIs
      description: |-
        APIs that install triggers on actions and events for devices
        or groups. A trigger always triggers an action.
    - name: NIPC management APIs
      description: |-
        APIs that manage device connections.
    - name: NIPC registration APIs
      description: |-
        APIs that register sdf models or data applications

  paths:
  ### NIPC Property APIs
    /devices/{id}/properties:
      put:
        tags:
          - NIPC property APIs
        summary: Update a value of one or more properties on a device
```

```
        description: |-
          Write a value to a property or multiple properties to a
          device. If underlying protocol requires a connection to be
          set up, this API call will perform the necessary connection
          management. If a connection is already active for this
          device, the existing connection will be leveraged without
          modifying it.
        operationId: UpdateProperties
        parameters:
        - name: id
          in: path
          description: The ID of the device. Group ID is not allowed.
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: propertyName
          in: query
          description: |-
            The SDF property name that needs to be written to.
          required: false
          allowReserved: true
          schema:
            type: string
            example: "https://example.com/heartrate#/sdfObject/thermos\
   \tat/sdfProperty/temperature"
        requestBody:
          description: |-
            The value to be written to the property or properties.
            If multiple properties are specified, the request body
            should be application/nipc+json.
          content:
            application/nipc+json:
              schema:
                $ref: '#/components/schemas/PropertyValueArray'
            "*/*":
              schema:
                description: |-
                  Any other content type, such as
                  application/octet-stream, application/json that will
                  be written to the device.
          required: true
        responses:
          '204':
            description: |-
              Success, no content, used for a single property write
          '200':
```

```
            description: Success, used for multiple property writes
            content:
              application/nipc+json:
                schema:
                  allOf:
                    - $ref: '#/components/schemas/PropertyValueRespons\
  \eArray'
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  $ref: '#/components/schemas/FailureResponse'

      get:
        tags:
          - NIPC property APIs
        summary: |-
          Read a value from one or multiple properties on a device
        description: |-
          Read a value to a property or multiple properties from a
          device. If underlying protocol requires a connection to be
          set up, this API call will perform the necessary connection
          management. If a connection is already active for this
          device, the existing connection will be leveraged without
          modifying it.
        operationId: GetProperties
        parameters:
        - name: id
          in: path
          description: The ID of the device. Group ID is not allowed.
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: propertyName
          in: query
          description: Properties to be read
          required: true
          allowReserved: true
          schema:
            type: array
            items:
              type: string
              example: "https://example.com/heartrate#/sdfObject/therm\
  \ostat/sdfProperty/temperature"
        responses:
```

```
          '200':
            description: Success
            content:
              application/nipc+json:
                schema:
                  allOf:
                    - $ref: '#/components/schemas/Id'
                    - $ref: '#/components/schemas/PropertyValueReadRes\
  \ponseArray'
              "*/*":
                schema:
                  type: string
                  description: |-
                    Any other content type, such as
                    application/octet-stream, application/json that
                    will be read from the device.
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  $ref: '#/components/schemas/FailureResponse'

    ### NIPC Event APIs
     /devices/{id}/events:
       post:
         tags:
           - NIPC event APIs
         summary: Enable an event on a specific device
         description: |-
           Enable an event on a specific device or for a group of
           devices. If the underlying protocol requires a connection to
           be set up, this API call will perform the necessary
           connection management. If a connection is already active for
           this device, the existing connection will be leveraged
           without modifying it.
         operationId: EnableEvent
         parameters:
         - name: id
           in: path
           description: The ID of the device.
           required: true
           schema:
             type: string
             format: uuid
             example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
         - name: eventName
           in: query
```

```
          description: event that needs to be enabled
          required: true
          allowReserved: true
          schema:
            type: string
            example: "https://example.com/heartrate#/sdfObject/healths\
   \ensor/sdfEvent/fallDetected"
        responses:
          '201':
            description: Success
            headers:
              Location:
                description: Location of the created event
                schema:
                  type: string
                  format: uri
                  example: "/devices/{id}/events?instanceId={instanceI\
   \d}"
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  $ref: '#/components/schemas/FailureResponse'

      delete:
        tags:
          - NIPC event APIs
        summary: Disable an event on a specific device
        description: |-
          Disable an event on a specific device or a group of devices.
          If the underlying protocol requires a connection to be set
          up, this API call will perform the necessary connection
          management. If a connection is already active for this
          device, the existing connection will be leveraged without
          modifying it.
        operationId: DisableEvent
        parameters:
        - name: id
          in: path
          description: The ID of the device.
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: instanceId
          in: query
```

```
          description: instance id of the event that needs to be disab\
   \led
          required: true
          schema:
            type: string
            format: uuid
            example: 02ee282c-8915-4b2e-bbd2-88966773134a
        responses:
          '204':
            description: Success, no content
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  $ref: '#/components/schemas/FailureResponse'

      get:
        tags:
          - NIPC event APIs
        summary: Get status of events on a device
        description: |-
          Get status of an event or multiple events on a specific devi\
   \ce
        operationId: GetEvents
        parameters:
        - name: id
          in: path
          description: The ID of the device.
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: instanceId
          in: query
          description: |-
            Instance ID of the events that need to be filtered
          required: false
          schema:
            type: array
            items:
              type: string
              format: uuid
              example: 02ee282c-8915-4b2e-bbd2-88966773134a
        responses:
          '200':
            description: Success
```

```
            content:
              application/nipc+json:
                schema:
                  $ref: '#/components/schemas/EventStatusResponseArray'
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  $ref: '#/components/schemas/FailureResponse'

    /groups/{id}/events:
      post:
        tags:
          - NIPC event APIs
        summary: Enable an event on a group of devices
        description: |-
          Enable an event on a group of devices.
          If the underlying protocol requires a connection to be set
          up, this API call will perform the necessary connection
          management. If a connection is already active for this
          device, the existing connection will be leveraged without
          modifying it.
        operationId: EnableGroupEvent
        parameters:
        - name: id
          in: path
          description: |-
            group id for which the event needs to be enabled
          required: true
          schema:
            type: string
            format: uuid
            example: 0dc729d7-f6c3-491d-9b9d-e7176d2be243
        - name: eventName
          in: query
          description: event that needs to be enabled
          required: true
          allowReserved: true
          schema:
            type: string
            example: "https://example.com/heartrate#/sdfObject/healths\
  \ensor/sdfEvent/fallDetected"
        responses:
          '201':
            description: Success, event enabled
            headers:
              Location:
```

```
                description: Location of the created event
                schema:
                  type: string
                  format: uri
                  example: "/groups/{id}/events?instanceId={instanceId\
   \}"
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  $ref: '#/components/schemas/FailureResponse'
      delete:
        tags:
          - NIPC event APIs
        summary: Disable an event on a group of devices
        description: |-
          Disable an event on a group of devices. If the underlying
          protocol requires a connection to be set up, this API call
          will perform the necessary connection management.
          If a connection is already active for this device, the
          existing connection will be leveraged without modifying it.
        operationId: DisableGroupEvent
        parameters:
        - name: id
          in: path
          description: |-
            group id for which the event needs to be disabled
          required: true
          schema:
            type: string
            format: uuid
            example: 0dc729d7-f6c3-491d-9b9d-e7176d2be243
        - name: instanceId
          in: query
          description: instance id of the event that needs to be disab\
   \led
          required: true
          schema:
            type: string
            format: uuid
            example: f1b9f26b-21ce-4deb-9d57-289ba7e17cce
        responses:
          '200':
            description: Success, event disabled
            content:
              application/nipc+json:
                schema:
```

```
                  $ref: '#/components/schemas/GroupEventStatusResponse\
   \Array'
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  $ref: '#/components/schemas/FailureResponse'
      get:
        tags:
          - NIPC event APIs
        summary: Get status of events on a group of devices
        description: |-
          Get status of an event or multiple events on a group of devi\
   \ces.
        operationId: GetGroupEvents
        parameters:
        - name: id
          in: path
          description: group id of the SCIM group
          required: true
          schema:
            type: string
            format: uuid
            example: 0dc729d7-f6c3-491d-9b9d-e7176d2be243
        - name: instanceId
          in: query
          description: |-
            Instance IDs of the events that need to be filtered
          required: false
          schema:
            type: string
            format: uuid
            example: f1b9f26b-21ce-4deb-9d57-289ba7e17cce
        responses:
          '200':
            description: Success, events retrieved
            content:
              application/nipc+json:
                schema:
                  $ref: '#/components/schemas/GroupEventStatusResponse\
   \Array'
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  $ref: '#/components/schemas/FailureResponse'
```

```
   ### NIPC action APIs
     /devices/{id}/actions:
       post:
         tags:
           - NIPC action APIs
         summary: Perform an action on a device
         description: |-
           Perform an action on a device.
           If the underlying protocol requires a connection to be set
           up, this API call will perform the necessary connection
           management. If a connection is already active for this
           device, the existing connection will be leveraged without
           modifying it.
         operationId: ActionProperty
         parameters:
         - name: id
           in: path
           description: The ID of the device. Group ID is not allowed.
           required: true
           schema:
             type: string
             format: uuid
             example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
         - name: actionName
           in: query
           description: action that needs to be performed
           required: true
           allowReserved: true
           schema:
             type: string
             example: "https://example.com/heartrate#/sdfObject/healths\
   \ensor/sdfAction/start"
         requestBody:
           content:
             application/octet-stream:
               schema:
                 type: string
                 format: binary
           required: false
         responses:
           '202':
             description: Accepted, action is being performed
             headers:
               Location:
                 description: Location of the action
                 schema:
                   type: string
                   format: uri
```

```
                example: "/devices/{id}/actions?instanceId={instance\
  \Id}"
        default:
          description: Error response
          content:
            application/problem+json:
              schema:
                $ref: '#/components/schemas/FailureResponse'
    get:
      tags:
        - NIPC action APIs
      summary: Get status of an action on a device
      description: |-
        Get status of an action on a specific device or a group of
        devices. Success is action is active, failure if action not
        active.
      operationId: GetAction
      parameters:
      - name: id
        in: path
        description: The ID of the device. Group ID is not allowed.
        required: true
        schema:
          type: string
          format: uuid
          example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
      - name: instanceId
        in: query
        description: |-
          instance id of the action that needs to be checked
        required: true
        schema:
          type: string
          format: uuid
          example: 02ee282c-8915-4b2e-bbd2-88966773134a
      responses:
        '200':
          description: Success, action is active
          content:
            application/nipc+json:
              schema:
                $ref: '#/components/schemas/ActionResponse'
        default:
          description: Error response
          content:
            application/problem+json:
              schema:
                $ref: '#/components/schemas/FailureResponse'
```

```
     /groups/{id}/actions:
       post:
         tags:
           - NIPC action APIs
         summary: Perform an action on a group
         description: |-
           Perform an action on a group of decvices
           If the underlying protocol requires a connection to be set
           up, this API call will perform the necessary connection
           management. If a connection is already active for this
           device, the existing connection will be leveraged without
           modifying it.
         operationId: ActionGroupProperty
         parameters:
         - name: id
           in: path
           description: The ID of the group.
           required: true
           schema:
             type: string
             format: uuid
             example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
         - name: actionName
           in: query
           description: action that needs to be performed
           required: true
           allowReserved: true
           schema:
             type: string
             example: "https://example.com/heartrate#/sdfObject/healths\
   \ensor/sdfAction/start"
         requestBody:
           content:
             application/octet-stream:
               schema:
                 type: string
                 format: binary
           required: false
         responses:
           '202':
             description: Accepted, action is being performed
             headers:
               Location:
                 description: Location of the action
                 schema:
                   type: string
                   format: uri
                   example: "/groups/{id}/actions?instanceId={instanceI\
```

```
   \d}"
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  $ref: '#/components/schemas/FailureResponse'
      get:
        tags:
          - NIPC action APIs
        summary: Get status of an action on a group
        description: |-
          Get status of an action on a specific group of
          devices. Success is action is active, failure if action not
          active.
        operationId: GetGroupAction
        parameters:
        - name: id
          in: path
          description: The ID of the device. Group ID is not allowed.
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: instanceId
          in: query
          description: |-
            instance id of the action that needs to be checked
          required: true
          schema:
            type: string
            format: uuid
            example: 02ee282c-8915-4b2e-bbd2-88966773134a
        responses:
          '200':
            description: Success, action is active
            content:
              application/nipc+json:
                schema:
                  $ref: '#/components/schemas/GroupActionStatusRespons\
   \eArray'
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  $ref: '#/components/schemas/FailureResponse'
```

```
   ### NIPC Trigger APIs
     /devices/{id}/triggers:
       post:
         tags:
           - NIPC trigger APIs
         summary: create a trigger on an affordance of a device
         description: |-
           Creates a trigger on an affordance of a device. A trigger
           will trigger an action on another device or a group of devic\
   \es
         operationId: CreateDeviceTrigger
         parameters:
         - name: id
           in: path
           description: The ID of the device.
           required: true
           schema:
             type: string
             format: uuid
             example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
         - name: sdfName
           in: query
           description: |-
             sdf affordance that will trigger this action, this can be
             either an event or an action
           required: true
           allowReserved: true
           schema:
             type: string
             example: "https://example.com/heartrate#/sdfObject/healths\
   \ensor/sdfEvent/fallDetected"
         requestBody:
           description: |-
             The NIPC API call to be called when the trigger is
             executed.
           content:
             application/nipc+json:
               schema:
                 $ref: '#/components/schemas/Action'
           required: true
         responses:
           '201':
             description: Success
             headers:
               Location:
                 description: Location of the created trigger
                 schema:
                   type: string
```

```
                  format: uri
                  example: "/devices/{id}/triggers?instanceId={instanc\
   \eId}"
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  $ref: '#/components/schemas/FailureResponse'

      get:
        tags:
          - NIPC trigger APIs
        summary: Get information about a trigger or all triggers
        description: |-
          Get information about a trigger for a device or all triggers
          if none specified
        operationId: GetDeviceTrigger
        parameters:
        - name: id
          in: path
          description: The ID of the device.
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: instanceId
          in: query
          description: |-
            instance id of the trigger that needs to be checked
          required: false
          schema:
            type: string
            format: uuid
            example: 02ee282c-8915-4b2e-bbd2-88966773134a
        responses:
          '200':
            description: Success, action is active
            content:
              application/nipc+json:
                schema:
                  $ref: '#/components/schemas/TriggerStatusResponseArr\
   \ay'
          default:
            description: Error response
            content:
              application/problem+json:
```

```
                schema:
                  $ref: '#/components/schemas/FailureResponse'

      delete:
        tags:
          - NIPC trigger APIs
        summary: Delete a trigger or all triggers for a device
        description: |-
          Delete a trigger for a device or all triggers
          if none specified
        operationId: DeleteDeviceTrigger
        parameters:
        - name: id
          in: path
          description: The ID of the device.
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: instanceId
          in: query
          description: |-
            instance id of the trigger that needs to be checked
          required: false
          schema:
            type: string
            format: uuid
            example: 02ee282c-8915-4b2e-bbd2-88966773134a
        responses:
          '204':
            description: Success, no content
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  $ref: '#/components/schemas/FailureResponse'

    /groups/{id}/triggers:
      post:
        tags:
          - NIPC trigger APIs
        summary: create a trigger on an affordance of a group
        description: |-
          Creates a trigger on an affordance of a group of devices. A \
  \trigger
          will trigger an action on a device or a group of devices
```

```
        operationId: CreateGroupTrigger
        parameters:
        - name: id
          in: path
          description: The ID of the group
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: sdfName
          in: query
          description: |-
            sdf affordance that will trigger this action, this can be
            either an event or an action
          required: true
          allowReserved: true
          schema:
            type: string
            example: "https://example.com/heartrate#/sdfObject/healths\
  \ensor/sdfEvent/fallDetected"
      requestBody:
        description: |-
          The NIPC API call to be called when the trigger is
          executed, as well as the device or group it is to be execu\
  \ted against.
        content:
          application/nipc+json:
            schema:
              $ref: '#/components/schemas/Action'
        required: true
      responses:
        '201':
          description: Success
          headers:
            Location:
              description: Location of the created trigger
              schema:
                type: string
                format: uri
                example: "/groups/{id}/triggers?instanceId={instance\
  \Id}"
        default:
          description: Error response
          content:
            application/problem+json:
              schema:
                $ref: '#/components/schemas/FailureResponse'
```

```
      get:
        tags:
          - NIPC trigger APIs
        summary: Get information about a trigger or all triggers
        description: |-
          Get information about a trigger or all triggers
          if none specified
        operationId: GetGroupTrigger
        parameters:
        - name: id
          in: path
          description: The ID of the group
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: instanceId
          in: query
          description: |-
            instance id of the trigger that needs to be checked
          required: false
          schema:
            type: string
            format: uuid
            example: 02ee282c-8915-4b2e-bbd2-88966773134a
        responses:
          '200':
            description: Success, action is active
            content:
              application/nipc+json:
                schema:
                  $ref: '#/components/schemas/GroupTriggerStatusRespon\
  \seArray'
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  $ref: '#/components/schemas/FailureResponse'

      delete:
        tags:
          - NIPC trigger APIs
        summary: Delete a trigger or all triggers for a grou of devices
        description: |-
          Delete a trigger for a group of devices or all triggers
          for a group of devices if none specified
```

```
        operationId: DeleteGroupTrigger
        parameters:
        - name: id
          in: path
          description: The ID of the group of devices.
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: instanceId
          in: query
          description: |-
            instance id of the trigger that needs to be checked
          required: false
          schema:
            type: string
            format: uuid
            example: 02ee282c-8915-4b2e-bbd2-88966773134a
        responses:
          '200':
            description: Success, trigger deleted
            content:
              application/nipc+json:
                schema:
                  $ref: '#/components/schemas/GroupTriggerStatusRespon\
  \seArray'
        default:
          description: Error response
          content:
            application/problem+json:
              schema:
                $ref: '#/components/schemas/FailureResponse'

  ### NIPC management APIs
    /devices/{id}/connections:
      post:
        tags:
          - NIPC management APIs
        summary: Connect a device
        description: |-
          Connect a device. 3 retries by default, optionally retry
          policy can be defined in the API body. If the protocol
          requires service discovery, full service discovery will be
          performed, unless specific services are described in the API
          body.
        operationId: ActionCreateConnection
        parameters:
```

```
          - name: id
            in: path
            description: The ID of the device. Group ID is not allowed.
            required: true
            schema:
              type: string
              format: uuid
              example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        requestBody:
          content:
            application/nipc+json:
              schema:
                anyOf:
                  - $ref: '#/components/schemas/Connection'
                  - $ref: './protocolinfo/ProtocolInfo.yaml#/component\
   \s/schemas/ProtocolInfo-ServiceMap'
              example:
                retries: 3
                protocolInformation:
                  ble:
                    services:
                      - serviceID: "00001809-0000-1000-8000-00805f9b34\
   \fb"
                    cached: true
                    cacheExpiryDuration: 3600
          required: false
        responses:
          '200':
            description: Success
            content:
              application/nipc+json:
                schema:
                  allOf:
                    - $ref: '#/components/schemas/Id'
                    - $ref: './protocolinfo/ProtocolInfo.yaml#/compone\
   \nts/schemas/ProtocolInfo-ServiceMap'
                example:
                  id: "1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30"
                  protocolInformation:
                    ble:
                      services:
                        - serviceID: "00001809-0000-1000-8000-00805f9b\
   \34fb"
                          characteristics:
                            - characteristicID: "00002a19-0000-1000-80\
   \00-00805f9b34fb"
                              flags: ["read", "notify"]
                              descriptors:
```

```
                                - descriptorID: "00002902-0000-1000-80\
   \00-00805f9b34fb"
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  $ref: '#/components/schemas/FailureResponse'

      put:
        tags:
          - NIPC management APIs
        summary: Update cached ServiceMap for a device.
        description: |-
          Update cached ServiceMap for a device. Full service discovery
          will be performed, unless specific services are described in
          the API body.
        operationId: ActionUpdateServiceMap
        parameters:
          - name: id
            in: path
            description: The ID of the device. Group ID is not allowed.
            required: true
            schema:
              type: string
              format: uuid
              example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        requestBody:
          content:
            application/nipc+json:
              schema:
                $ref: './protocolinfo/ProtocolInfo.yaml#/components/sc\
   \hemas/ProtocolInfo-ServiceMap'
              example:
                retries: 3
                protocolInformation:
                  ble:
                    services:
                      - serviceID: "00001809-0000-1000-8000-00805f9b34\
   \fb"
                    cached: true
                    cacheExpiryDuration: 3600
          required: false
        responses:
          '200':
            description: Success
            content:
              application/nipc+json:
```

```
                schema:
                  allOf:
                    - $ref: '#/components/schemas/Id'
                    - $ref: './protocolinfo/ProtocolInfo.yaml#/compone\
  \nts/schemas/ProtocolInfo-ServiceMap'
                example:
                  id: "1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30"
                  protocolInformation:
                    ble:
                      services:
                        - serviceID: "00001809-0000-1000-8000-00805f9b\
  \34fb"
                          characteristics:
                            - characteristicID: "00002a19-0000-1000-80\
  \00-00805f9b34fb"
                              flags: ["read", "notify"]
                              descriptors:
                                - descriptorID: "00002902-0000-1000-80\
  \00-00805f9b34fb"
        default:
          description: Error response
          content:
            application/problem+json:
              schema:
                $ref: '#/components/schemas/FailureResponse'

    delete:
      tags:
        - NIPC management APIs
      summary: Disconnect a device
      description: |-
        Disconnect a device.
      operationId: ActionDeleteConnection
      parameters:
        - name: id
          in: path
          description: The ID of the device. Group ID is not allowed.
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
      responses:
        '200':
          description: Success
          content:
            application/nipc+json:
              schema:
```

```
                  allOf:
                    - $ref: '#/components/schemas/Id'
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  $ref: '#/components/schemas/FailureResponse'

      get:
        tags:
          - NIPC management APIs
        summary: Get connection state for a device
        description: |-
          Get connection status for a device. Success when device(s)
          is/are connected, includes service map for the device if
          available. Failure when a device is not connected
        operationId: ActionGetConnection
        parameters:
          - name: id
            in: path
            description: The ID of the device. Group ID is not allowed.
            required: true
            schema:
              type: string
              format: uuid
              example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        responses:
          '200':
            description: Success
            content:
              application/nipc+json:
                schema:
                  allOf:
                    - $ref: '#/components/schemas/Id'
                    - $ref: './protocolinfo/ProtocolInfo.yaml#/compone\
  \nts/schemas/ProtocolInfo-ServiceMap'
                example:
                  id: "1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30"
                  protocolInformation:
                    ble:
                      services:
                        - serviceID: "00001809-0000-1000-8000-00805f9b\
  \34fb"
                          characteristics:
                            - characteristicID: "00002a19-0000-1000-80\
  \00-00805f9b34fb"
                              flags: ["read", "notify"]
```

```
                                descriptors:
                                  - descriptorID: "00002902-0000-1000-80\
   \00-00805f9b34fb"
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  $ref: '#/components/schemas/FailureResponse'


   ### Registrations
     /registrations/models:
       post:
         tags:
           - NIPC registration APIs
         summary: Register an sdfObject
         description: |-
           Register an sdfObject, including Properties, Events and
           actions
         operationId: registerSdfObject
         requestBody:
           content:
             application/sdf+json:
               schema:
                 $ref: '#/components/schemas/SdfModel'
           required: true
         responses:
           '200':
             description: Success
             content:
               application/nipc+json:
                 schema:
                   type: array
                   items:
                     allOf:
                       - $ref: '#/components/schemas/SdfReference'
           default:
             description: Error response
             content:
               application/problem+json:
                 schema:
                   allOf:
                     - $ref: '#/components/schemas/FailureResponse'

       get:
         tags:
           - NIPC registration APIs
```

```
        summary: Get all registered SDF model names
        description: |-
          Get all registered SDF model names.
        operationId: getSdfRefs
        parameters:
          - name: sdfName
            in: query
            description: |-
              sdfName can be a reference to an sdfThing or sdfObject
            required: false
            allowReserved: true
            schema:
              type: string
              example: "https://example.com/heartrate#/sdfObject/healt\
   \hsensor"
        responses:
          '200':
            description: Success
            content:
              application/sdf+json:
                schema:
                  $ref: '#/components/schemas/SdfModel'
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  allOf:
                    - $ref: '#/components/schemas/FailureResponse'
      put:
        tags:
          - NIPC registration APIs
        summary: Update an SDF model
        description: |-
          Update an SDF model, including Properties, Events and
          actions
        operationId: updateSdf
        parameters:
          - name: sdfName
            in: query
            description: |-
              sdfName can be a reference to an sdfThing or sdfObject
            required: true
            allowReserved: true
            schema:
              type: string
              example: "https://example.com/heartrate#/sdfObject/healt\
   \hsensor"
```

```
        requestBody:
          content:
            application/sdf+json:
              schema:
                $ref: '#/components/schemas/SdfModel'
          required: true
        responses:
          '200':
            description: Success
            content:
              application/nipc+json:
                schema:
                  allOf:
                    - $ref: '#/components/schemas/SdfReference'
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  allOf:
                    - $ref: '#/components/schemas/FailureResponse'

      delete:
        tags:
          - NIPC registration APIs
        summary: Delete an sdfObject
        description: |-
          Delete an sdfObject, including Properties, Events and
          actions
        operationId: deleteSdfObject
        parameters:
          - name: sdfName
            in: query
            description: sdfObject name
            required: true
            schema:
              type: string
              example: "https://example.com/heartrate#/sdfObject/healt\
  \hsensor"
        responses:
          '200':
            description: Success
            content:
              application/nipc+json:
                schema:
                  allOf:
                    - $ref: '#/components/schemas/SdfReference'
          default:
```

```
            description: Error response
            content:
              application/problem+json:
                schema:
                  allOf:
                    - $ref: '#/components/schemas/FailureResponse'

    /registrations/data-apps:
      post:
        tags:
          - NIPC registration APIs
        summary: Register a dataApp
        description: |-
          Register a dataApp that is able to receive device data.
        operationId: registerDataApp
        parameters:
          - name: dataAppId
            in: query
            description: id of the data app that will be registered
            required: true
            schema:
              type: string
              format: uuid
              example: 0927ce7c-b258-4bfa-a345-bcc9f74385b4
        requestBody:
          content:
            application/nipc+json:
              schema:
                $ref: '#/components/schemas/DataApp'
          required: true
        responses:
          '200':
            description: Success
            content:
              application/nipc+json:
                schema:
                  allOf:
                    - $ref: '#/components/schemas/DataApp'
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  allOf:
                    - $ref: '#/components/schemas/FailureResponse'

      put:
        tags:
```

```
          - NIPC registration APIs
        summary: Update registration of a dataApp
        description: |-
          Update registration of a dataApp that is able to receive dev\
   \ice data.
        operationId: UpdateDataApp
        parameters:
          - name: dataAppId
            in: query
            description: id of the data app that will be updated
            required: true
            schema:
              type: string
              format: uuid
              example: 0927ce7c-b258-4bfa-a345-bcc9f74385b4
        requestBody:
          content:
            application/nipc+json:
              schema:
                $ref: '#/components/schemas/DataApp'
          required: true
        responses:
          '200':
            description: Success
            content:
              application/nipc+json:
                schema:
                  allOf:
                    - $ref: '#/components/schemas/DataApp'
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  allOf:
                    - $ref: '#/components/schemas/FailureResponse'

      delete:
        tags:
          - NIPC registration APIs
        summary: Delete registration of a dataApp
        description: |-
          Delete registration of a dataApp that is able to receive
          device data.
        operationId: DeleteDataApp
        parameters:
          - name: dataAppId
            in: query
```

```
            description: id of the data app that will be updated
            required: true
            schema:
              type: string
              format: uuid
              example: 0927ce7c-b258-4bfa-a345-bcc9f74385b4
        responses:
          '200':
            description: Success
            content:
              application/nipc+json:
                schema:
                  allOf:
                    - $ref: '#/components/schemas/DataApp'
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  allOf:
                    - $ref: '#/components/schemas/FailureResponse'

      get:
        tags:
          - NIPC registration APIs
        summary: Get registration of a dataApp
        description: |-
          Get registrationdetails of a dataApp that is able to receive
          device data.
        operationId: GetDataApp
        parameters:
          - name: dataAppId
            in: query
            description: id of the data app that will be updated
            required: true
            schema:
              type: string
              format: uuid
              example: 0927ce7c-b258-4bfa-a345-bcc9f74385b4
        responses:
          '200':
            description: Success
            content:
              application/nipc+json:
                schema:
                  allOf:
                    - $ref: '#/components/schemas/DataApp'
          default:
```

```
            description: Error response
            content:
              application/problem+json:
                schema:
                  allOf:
                    - $ref: '#/components/schemas/FailureResponse'

  components:
    schemas:
  # Base objects
  ## A SCIM id, can be a device or a group
      Id:
        required:
          - id
        type: object
        properties:
          id:
            type: string
            format: uuid
            description: |-
              A SCIM-generated UUID, can be a device or group
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30

  ## A property
      Property:
        required:
          - property
        type: object
        properties:
          property:
            type: string
            example: "https://example.com/heartrate#/sdfObject/thermos\
  \tat/sdfProperty/temperature"

  ## A value
      Value:
        required:
          - value
        type: object
        properties:
          value:
            type: string
            format: byte
            example: dGVzdA==

  ## A value of an property of an Device
      PropertyValue:
        allOf:
```

```
          - $ref: '#/components/schemas/Property'
          - $ref: '#/components/schemas/Value'

   ## An array of Property values
       PropertyValueArray:
         type: array
         items:
           $ref: '#/components/schemas/PropertyValue'

   ## Event
       Event:
         required:
           - event
         type: object
         properties:
           event:
             type: string
             description: |-
               percent-encoded JSON pointer to the SDF event object
             example: https://example.com/heartrate#/sdfObject/healthse\
   \nsor/sdfEvent/fallDetected

       InstanceId:
         type: object
         properties:
           instanceId:
             type: string
             format: uuid
             description: |-
               A SCIM-generated UUID for the event instance
             example: 02ee282c-8915-4b2e-bbd2-88966773134a

   ## An Action
       Action:
         type: object
         properties:
           action:
             type: string
             description: |-
               NIPC action operation to execute
             example: /devices/3171ec43-42a5-4415-ab4b-afd0dfbe9615/act\
   \ions?actionName=https%3A%2F%2Fexample.com%2FAlarmSystem%23%2FsdfObj\
   \ect%2Fbell%2FsdfAction%2Fring

   ## A Connection
       Connection:
         type: object
         properties:
```

```
          retries:
            type: integer
            format: int32
            example: 3

    ## DataApp
       DataApp:
         oneOf:
           - $ref: '#/components/schemas/DataAppMqttClient'
           - $ref: '#/components/schemas/DataAppMqttBroker'
           - $ref: '#/components/schemas/DataAppWebhook'
           - $ref: '#/components/schemas/DataAppWebsocket'
         type: object
         properties:
           events:
             type: array
             items:
               $ref: '#/components/schemas/Event'


       DataAppMqttClient:
         type: object
         properties:
           mqttClient:
             type: boolean

       DataAppMqttBroker:
         type: object
         properties:
           mqttBroker:
             type: object
             required:
               - URI
               - username
               - password
             properties:
               URI:
                 type: string
                 example: mqtt.broker.com:8883
               username:
                 type: string
                 example: user1
               password:
                 type: string
                 example: password1
               brokerCACert:
                 description: PEM encoded CA certificate
                 type: string
```

```
              customTopic:
                type: string
                description: custom MQTT topic to publish to
                example: custom/topic

      DataAppWebhook:
        type: object
        properties:
          webhook:
            type: object
            properties:
              URI:
                type: string
                example: webhook.com:443
              headers:
                type: object
                additionalProperties:
                  type: string
                example:
                  x-api-key: fjelk-3dl33f-2wdsd
              serverCACert:
                type: string

      DataAppWebsocket:
        type: object
        properties:
          websocket:
            type: object
            properties:
              URI:
                type: string
                example: websocket.com:443
              headers:
                type: object
                additionalProperties:
                  type: string
                example:
                  x-api-key: fjelk-3dl33f-2wdsd
              serverCACert:
                type: string

   ## sdfObject registration definition
      SdfReference:
        type: object
        description: SDF URL referring to the sdfobject
        properties:
          sdfName:
            type: string
```

```
            example: "https://example.com/heartrate#/sdfObject/healths\
   \ensor"

        SdfModel:
          allOf:
            - type: object
              description: Sample SDF model
              properties:
                namespace:
                  type: object
                  additionalProperties:
                    type: string
                  example:
                    heartrate: https://example.com/heartrate
                defaultNamespace:
                  type: string
                  example: heartrate
            - oneOf:
              - $ref: '#/components/schemas/SdfThing'
              - $ref: '#/components/schemas/SdfObject'

        SdfThing:
          type: object
          description: Sample SDF thing
          properties:
            sdfThing:
              additionalProperties:
                anyOf:
                    - $ref: '#/components/schemas/SdfProperty'
                    - $ref: '#/components/schemas/SdfEvent'
                    - $ref: '#/components/schemas/SdfAction'
                    - $ref: '#/components/schemas/SdfObject'
              example:
                multipleSensor:
                  sdfEvent:
                    isPresent:
                      sdfProtocolMap:
                        ble:
                          type: advertisement
                  sdfObject:
                    healthsensor:
                      sdfProperty:
                        heartrate:
                          sdfProtocolMap:
                            ble:
                              serviceID: 00001809-0000-1000-8000-00805f9\
   \b34fb
                              characteristicID: 00002a1c-0000-1000-8000-\
```

```
   \00805f9b34fb
                    sdfEvent:
                      fallDetected:
                          sdfProtocolMap:
                            ble:
                              serviceID: 00001809-0000-1000-8000-00805\
   \f9b34fb
                              characteristicID: 00002a1c-0000-1000-800\
   \0-00805f9b34fb
                    sdfAction:
                      start:
                        sdfProtocolMap:
                          ble:
                            serviceID: 00001809-0000-1000-8000-00805f9\
   \b34fb
                            characteristicID: 00002a1c-0000-1000-8000-\
   \00805f9b34fb

        SdfObject:
          type: object
          description: Sample SDF object
          properties:
            sdfObject:
              additionalProperties:
                anyOf:
                  - $ref: '#/components/schemas/SdfProperty'
                  - $ref: '#/components/schemas/SdfEvent'
                  - $ref: '#/components/schemas/SdfAction'
              example:
                healthsensor:
                  sdfProperty:
                    heartrate:
                      sdfProtocolMap:
                        ble:
                          serviceID: 00001809-0000-1000-8000-00805f9b34fb
                          characteristicID: 00002a1c-0000-1000-8000-0080\
   \5f9b34fb
                  sdfEvent:
                    fallDetected:
                      sdfProtocolMap:
                        ble:
                          type: advertisements
                  sdfAction:
                    start:
                      sdfProtocolMap:
                        ble:
                          serviceID: 00001809-0000-1000-8000-00805f9b34fb
                          characteristicID: 00002a1c-0000-1000-8000-0080\
```

```
   \5f9b34fb

       SdfProperty:
         type: object
         description: Sample SDF property
         properties:
           sdfProperty:
             additionalProperties:
               allOf:
                 - $ref: ’./protocolmaps/ProtocolMap.yaml#/components/s\
   \chemas/ProtocolMap’
             example:
               heartrate:
                 sdfProtocolMap:
                   ble:
                     serviceID: 00001809-0000-1000-8000-00805f9b34fb
                     characteristicID: 00002a1c-0000-1000-8000-00805f9b\
   \34fb

       SdfEvent:
         type: object
         description: Sample SDF event
         properties:
           sdfEvent:
             additionalProperties: #example, this will be the registere\
   \d event
               allOf:
                 - $ref: ’./protocolmaps/ProtocolMap.yaml#/components/s\
   \chemas/ProtocolMap’
             example:
               fallDetected:
                   sdfProtocolMap:
                     ble:
                       type: gatt
                       serviceID: 00001809-0000-1000-8000-00805f9b34fb
                       characteristicID: 00002a1c-0000-1000-8000-00805f\
   \9b34fb

       SdfAction:
         type: object
         description: Sample SDF action
         properties:
           sdfAction:
             additionalProperties:
               allOf:
                 - $ref: ’./protocolmaps/ProtocolMap.yaml#/components/s\
   \chemas/ProtocolMap’
             example:
```

```
              start:
                sdfProtocolMap:
                  ble:
                    serviceID: 00001809-0000-1000-8000-00805f9b34fb
                    characteristicID: 00002a1c-0000-1000-8000-00805f9b\
   \34fb

   # responses

       SuccessResponse:
         type: object
         properties:
           status:
             type: integer
             format: int32
             example: 200
             description: HTTP status code

   ## Error 500 application Failure response
       FailureResponse:
         type: object
         properties:
           type:
             type: string
             description: URI to the error type
             enum:
               - https://www.iana.org/assignments/nipc-problem-types#in\
   \valid-id
               - https://www.iana.org/assignments/nipc-problem-types#in\
   \valid-sdf-url
               - https://www.iana.org/assignments/nipc-problem-types#ex\
   \tension-operation-not-executed
               - https://www.iana.org/assignments/nipc-problem-types#sd\
   \f-model-already-registered
               - https://www.iana.org/assignments/nipc-problem-types#sd\
   \f-model-in-use
               - https://www.iana.org/assignments/nipc-problem-types#pr\
   \operty-not-readable
               - https://www.iana.org/assignments/nipc-problem-types#pr\
   \operty-read-failed
               - https://www.iana.org/assignments/nipc-problem-types#pr\
   \operty-not-writable
               - https://www.iana.org/assignments/nipc-problem-types#pr\
   \operty-write-failed
               - https://www.iana.org/assignments/nipc-problem-types#ev\
   \ent-already-enabled
               - https://www.iana.org/assignments/nipc-problem-types#ev\
   \ent-not-enabled
```

```
              - https://www.iana.org/assignments/nipc-problem-types#ev\
   \ent-not-registered
              - https://www.iana.org/assignments/nipc-problem-types#pr\
   \otocolmap-ble-already-connected
              - https://www.iana.org/assignments/nipc-problem-types#pr\
   \otocolmap-ble-no-connection
              - https://www.iana.org/assignments/nipc-problem-types#pr\
   \otocolmap-ble-connection-timeout
              - https://www.iana.org/assignments/nipc-problem-types#pr\
   \otocolmap-ble-bonding-failed
              - https://www.iana.org/assignments/nipc-problem-types#pr\
   \otocolmap-ble-connection-failed
              - https://www.iana.org/assignments/nipc-problem-types#pr\
   \otocolmap-ble-service-discovery-failed
              - https://www.iana.org/assignments/nipc-problem-types#pr\
   \otocolmap-ble-invalid-service-or-characteristic
              - https://www.iana.org/assignments/nipc-problem-types#pr\
   \otocolmap-zigbee-connection-timeout
              - https://www.iana.org/assignments/nipc-problem-types#pr\
   \otocolmap-zigbee-invalid-endpoint-or-cluster
              - https://www.iana.org/assignments/nipc-problem-types#ex\
   \tension-transmit-invalid-data
              - https://www.iana.org/assignments/nipc-problem-types#ex\
   \tension-firmware-rollback
              - https://www.iana.org/assignments/nipc-problem-types#ex\
   \tension-firmware-update-failed
              - about:blank
            status:
              type: integer
              format: int32
              example: 400
              description: HTTP status code
            title:
              type: string
              example: Invalid Device ID
              description: Human-readable error title
            detail:
              type: string
              example: |-
                Device ID 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30 does not
                exist or is not a device
              description: Human-readable error message

   ## Property operations responses

       PropertyValueResponseArrayItem:
         oneOf:
           - $ref: '#/components/schemas/SuccessResponse'
```

```
          - $ref: '#/components/schemas/FailureResponse'

      PropertyValueResponseArray:
        type: array
        items:
            $ref: '#/components/schemas/PropertyValueResponseArrayItem'

      PropertyValueReadResponseArrayItem:
        oneOf:
          - $ref: '#/components/schemas/PropertyValue'
          - $ref: '#/components/schemas/FailureResponse'

      PropertyValueReadResponseArray:
        type: array
        items:
          allOf:
            - $ref: '#/components/schemas/PropertyValueReadResponseArr\
  \ayItem'

  ## Event operations responses
      EventStatusResponseArrayItem:
        oneOf:
          - allOf:
            - $ref: '#/components/schemas/Event'
            - $ref: '#/components/schemas/InstanceId'
          - $ref: '#/components/schemas/FailureResponse'

      EventStatusResponseArray:
        type: array
        items:
            $ref: '#/components/schemas/EventStatusResponseArrayItem'

      GroupEventStatusResponse:
        type: object
        oneOf:
          - allOf:
            - $ref: '#/components/schemas/Event'
            - type: object
              properties:
                deviceId:
                  type: string
                  format: uuid
                  example: 0dc729d7-f6c3-491d-9b9d-e7176d2be243
          - $ref: '#/components/schemas/FailureResponse'

      GroupEventStatusResponseArray:
        type: array
        items:
```

```
          $ref: '#/components/schemas/GroupEventStatusResponse'

    ActionResponse:
      required:
        - action
      type: object
      properties:
        status:
          type: string
          example: COMPLETED
          description: |-
            Status of the action, can be IN_PROGRESS or COMPLETED

    GroupActiontStatusResponse:
      type: object
      oneOf:
        - allOf:
          - $ref: '#/components/schemas/ActionResponse'
          - type: object
            properties:
              deviceId:
                type: string
                format: uuid
                example: 0dc729d7-f6c3-491d-9b9d-e7176d2be243
        - $ref: '#/components/schemas/FailureResponse'

    GroupActionStatusResponseArray:
      type: array
      items:
          $ref: '#/components/schemas/GroupActiontStatusResponse'

    TriggerResponse:
      type: object
      allOf:
        - $ref: '#/components/schemas/InstanceId'
        - $ref: '#/components/schemas/SdfReference'
        - $ref: '#/components/schemas/Action'

    TriggerStatusResponseArray:
      type: array
      items:
          $ref: '#/components/schemas/TriggerResponse'

    GroupTriggerResponse:
      type: object
      allOf:
        - type: object
          properties:
```

```
              deviceId:
                type: string
                format: uuid
                example: 0dc729d7-f6c3-491d-9b9d-e7176d2be243
          - $ref: '#/components/schemas/SdfReference'
          - $ref: '#/components/schemas/Action'

      GroupTriggerStatusResponseArray:
        type: array
        items:
            $ref: '#/components/schemas/GroupTriggerResponse'
  <CODE ENDS>
```

Figure 35

Appendix B. Protocol Mapping

The OpenAPI model for SDF protocol mapping is provided in Appendix B of [I-D.ietf-asdf-sdf-protocol-mapping].

Appendix C. Protocol Information

```
   <CODE BEGINS> file "ProtocolInfo.yaml"
   =============== NOTE: '\' line wrapping per RFC 8792 ================

   openapi: 3.0.3
   info:
     title: SDF Protocol Information
     description: |-
       SDF Protocol Information. When adding a
       new protocol information schema please add a reference to the pr\
   otocol info
       for all the schemas in this file.
     version: 0.10.0
   externalDocs:
     description: SDF Protocol Mapping IETF draft
     url: https://datatracker.ietf.org/doc/draft-ietf-asdf-sdf-protocol\
   -mapping/

   paths: {}

   components:
     schemas:
   # Protocol Information
   ## Protocol Info for Service Discovery result
       ProtocolInfo-ServiceMap:
         type: object
         properties:
           protocolInformation:
             oneOf:
               - $ref: './ProtocolInfo-BLE.yaml#/components/schemas/Pro\
   tocolInfo-BLE-ServiceMap'
               - $ref: './ProtocolInfo-Zigbee.yaml#/components/schemas/\
   ProtocolInfo-Zigbee-ServiceMap'

   ## Protocol Info for Broadcasts
       ProtocolInfo-Broadcast:
         type: object
         properties:
           protocolInformation:
             oneOf:
               - $ref: './ProtocolInfo-BLE.yaml#/components/schemas/Pro\
   tocolInfo-BLE-Broadcast'
               - $ref: './ProtocolInfo-Zigbee.yaml#/components/schemas/\
   ProtocolInfo-Zigbee-Broadcast'
   <CODE ENDS>
```

C.1. Protocol Information for BLE

```
   <CODE BEGINS> file "ProtocolInfo-BLE.yaml"
   ================ NOTE: '\' line wrapping per RFC 8792 ================

   openapi: 3.0.3
   info:
     title: SDF Protocol Information for BLE
     description: |-
       SDF Protocol Information for BLE devices.
     version: 0.10.0
   externalDocs:
     description: SDF Protocol Mapping IETF draft
     url: https://datatracker.ietf.org/doc/draft-ietf-asdf-sdf-protocol\
   -mapping/

   paths: {}

   components:
     schemas:
   # BLE Protocol Info
   ## A Service is a device with optional service IDs
       ProtocolInfo-BLE-ServiceMap:
         type: object
         properties:
           ble:
             type: object
             properties:
               services:
                 type: array
                 items:
                   type: object
                   allOf:
                     - $ref: '#/components/schemas/ProtocolInfo-BLE-Ser\
   vice'
               cached:
                 description: |-
                   If we can cache information, then device doesn't need
                   to be rediscovered before every connected.
                 type: boolean
                 default: false
               cacheExpiryDuration:
                 description: cache expiry period in seconds, when devi\
   ce allows
                 type: integer
                 example: 3600 # default 1 hour
               autoUpdate:
                 description: |-
                   autoupdate services if device supports it (default)
                 type: boolean
```

```
                example: true
              bonding: #optional, by default defined in SCIM object
                type: string
                example: default
                enum:
                  - default
                  - none
                  - justworks
                  - passkey
                  - oob

      ProtocolInfo-BLE-Service:
        required:
          - serviceID
        type: object
        properties:
          serviceID:
            type: string
            format: uuid
            example: 00001809-0000-1000-8000-00805f9b34fb
          characteristics:
            type: array
            items:
              $ref: '#/components/schemas/ProtocolInfo-BLE-Characteris\
  tic'

      ProtocolInfo-BLE-Characteristic:
        type: object
        properties:
          characteristicID:
            type: string
            format: uuid
            example: 00002a1c-0000-1000-8000-00805f9b34fb
          flags:
            type: array
            example:
            - read
            - write
            items:
              type: string
              enum:
                - read
                - write
                - notify
          descriptors:
            type: array
            items:
              $ref: '#/components/schemas/ProtocolInfo-BLE-Descriptor'
```

```
      ProtocolInfo-BLE-Descriptor:
        type: object
        properties:
          descriptorID:
            type: string
            format: uuid
            example: 00002902-0000-1000-8000-00805f9b34fb

  ##  Protocol Info for BLE Broadcast
      ProtocolInfo-BLE-Broadcast:
        required:
          - ble
        type: object
        properties:
          ble:
            type: object
            properties:
              connectable:
                type: boolean
  <CODE ENDS>
```

C.2. Protocol Information for Zigbee

```
  <CODE BEGINS> file "ProtocolInfo-Zigbee.yaml"
  =============== NOTE: '\' line wrapping per RFC 8792 ================

  openapi: 3.0.3
  info:
    title: SDF Protocol Information for Zigbee
    description: |-
      SDF Protocol Information for Zigbee devices.
    version: 0.10.0
  externalDocs:
    description: SDF Protocol Mapping IETF draft
    url: https://datatracker.ietf.org/doc/draft-ietf-asdf-sdf-protocol\
  -mapping/

  paths: {}

  components:
    schemas:
  # Zigbee Protocol Information
  ##  Protocol Information for Zigbee Service Map
      ProtocolInfo-Zigbee-ServiceMap:
        required:
          - zigbee
        type: object
        properties:
```

```
          zigbee:
            type: object
            properties:
              endpoints:
                type: array
                items:
                  $ref: '#/components/schemas/ProtocolInfo-Zigbee-Endp\
oint'

      ProtocolInfo-Zigbee-Endpoint:
        required:
          - endpointID
        type: object
        properties:
          endpointID:
            type: integer
            format: int32
            example: 10
          clusters:
            type: array
            items:
              $ref: '#/components/schemas/ProtocolInfo-Zigbee-Cluster'

      ProtocolInfo-Zigbee-Cluster:
        type: object
        properties:
          clusterID:
            type: integer
            format: int32
            example: 0
          properties:
            type: array
            items:
              $ref: '#/components/schemas/ProtocolInfo-Zigbee-Property'

      ProtocolInfo-Zigbee-Property:
        type: object
        properties:
          attributeID:
            type: integer
            format: int32
            example: 1
          propertyType:
            type: integer
            format: int32
            example: 32

   ## Protocol Information for Zigbee broadcast
```

```
      ProtocolInfo-Zigbee-Broadcast:
        required:
          - zigbee
        type: object
        properties:
          zigbee:
            type: object
  <CODE ENDS>
```

Appendix D. NIPC API extensions

The following OpenAPI models define a few example extensions to the NIPC API.

D.1. NIPC API write binary blob extension

```
  <CODE BEGINS> file "Extension-Blob.yaml"
  =============== NOTE: '\' line wrapping per RFC 8792 ================

  # yaml-language-server: $schema=https://json-schema.org/draft/2020-1\
  2/schema
  openapi: 3.0.3
  info:
    title: Non IP Device Control (NIPC) API write binary blob extension
    description: |-
      Non IP Device Control (NIPC) API write binary blob extension
    termsOfService: http://swagger.io/terms/
    contact:
      email: bbrinckm@cisco.com
    license:
      name: TBD
      url: TBD
    version: 0.17.0
  externalDocs:
    description: NIPC IETF draft
    url: https://datatracker.ietf.org/doc/draft-ietf-asdf-nipc/
  servers:
    - url: "{gw_host}/nipc/draft-17"
      variables:
        gw_host:
          default: localhost
          description: Gateway Host
  tags:
    - name:  NIPC API extensions
      description: |-
        APIs that simplify application interaction by implementing
        one or more basic APIs into a single API call.
```

```
   paths:
   ### Extensions
     /extensions/{id}/properties/blob:
       put:
         tags:
           - NIPC API extensions
         summary: Write a binary blob to a property on a device
         description: |-
           Write a binary blob to a property on a device. Will chunk up
           the binary blob and perform multiple writes. If the
           underlying protocol requires a connection to be set up,
           this API call will perform the necessary connection
           management. If a connection is already active for this
           device, the existing connection will be leveraged without
           modifying it. ID cannot be a group-id.
         operationId: writeBlob
         parameters:
         - name: id
           in: path
           description: The ID of the device. Group ID is not allowed.
           required: true
           schema:
             type: string
             format: uuid
             example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
         - name: propertyName
           in: query
           description: |-
             The SDF property name that needs to be written to.
           required: true
           schema:
             type: string
             example: "https://example.com/heartrate#/sdfObject/thermos\
   tat/sdfProperty/firmware"
         requestBody:
           content:
             application/json:
               schema:
                 $ref: '#/components/schemas/Extension-Blob'
           required: true
         responses:
           '204':
             description: Success, no content
           'default':
             description: Error response
             content:
               application/json:
                 schema:
```

```
                  $ref: '../NIPC.yaml#/components/schemas/FailureRespo\
nse'

components:
  schemas:
# Extensions
## A binary blob Extension
    Extension-Blob:
      required:
        - blob
      type: object
      properties:
        blob:
          type: string
          format: byte
        chunksize:
          type: integer
<CODE ENDS>
```

D.2. NIPC API bulk operations extension

```
<CODE BEGINS> file "Extension-Bulk.yaml"
================ NOTE: '\' line wrapping per RFC 8792 ================

# yaml-language-server: $schema=https://json-schema.org/draft/2020-1\
2/schema
openapi: 3.0.3
info:
  title: Non IP Device Control (NIPC) API bulk extension
  description: |-
    Non IP Device Control (NIPC) API bulk extension
  termsOfService: http://swagger.io/terms/
  contact:
    email: bbrinckm@cisco.com
  license:
    name: TBD
    url: TBD
  version: 0.17.0
externalDocs:
  description: NIPC IETF draft
  url: https://datatracker.ietf.org/doc/draft-ietf-asdf-nipc/
servers:
  - url: "{gw_host}/nipc/draft-17"
    variables:
      gw_host:
        default: localhost
        description: Gateway Host
tags:
```

```
    - name: NIPC API extensions
      description: |-
        APIs that simplify application interaction by implementing
        one or more basic APIs into a single API call.

  paths:
    ### Extensions
    /extensions/{id}/bulk:
      post:
        tags:
          - NIPC API extensions
        summary: Compound operations on a device
        description: Compound operations on a device
        operationId: Bulk
        parameters:
          - name: id
            in: path
            description: The ID of the device. Group ID is not allowed.
            required: true
            schema:
              type: string
              format: uuid
              example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        requestBody:
          content:
            application/json:
              schema:
                $ref: "#/components/schemas/Extension-Bulk"
              examples:
                bulkRequest:
                  $ref: "#/components/examples/bulkRequest"
                firmwareUpgradeRequest:
                  $ref: "#/components/examples/firmwareUpgradeRequest"
          required: true
        responses:
          "202":
            description: Accepted
            headers:
              Location:
                schema:
                  type: string
                  example: /extensions/1d3b2c36-8a65-45a6-87c1-bcdbe0a\
  32e30/bulk/status?instanceId=02ee282c-8915-4b2e-bbd2-88966773134a
                description: URL to get the bulk status response
          "401":
            description: Unauthorized
          "405":
            description: Invalid request
```

```
          "500":
            description: Server-side failure
            content:
              application/json:
                schema:
                  $ref: "../NIPC.yaml#/components/schemas/FailureRespo\
   nse"
      callbacks:
        bulkEvent:
          "{$request.body#/callback.url}":
            post:
              description: Callback for bulk response
              operationId: bulkCallback
              requestBody:
                content:
                  application/json:
                    schema:
                      allOf:
                        - $ref: "../NIPC.yaml#/components/schemas/Id"
                        - $ref: "#/components/schemas/Extension-Bulk\
   Response"
              responses:
                "200":
                  description: OK
                "400":
                  description: Bad request
                "401":
                  description: Unauthorized
                "405":
                  description: Invalid request
                "500":
                  description: Server-side failure
    get:
      tags:
        - NIPC API extensions
      summary: Get Bulk response
      description: Get Bulk response
      operationId: getBulkResponse
      parameters:
        - name: id
          in: path
          description: The ID of the device. Group ID is not allowed.
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: instanceId
```

```
            in: query
            description: Instance ID of the bulk operation
            required: true
            schema:
              type: string
              format: uuid
              example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        responses:
          "200":
            description: OK
            headers:
            content:
              application/json:
                schema:
                  allOf:
                    - $ref: "../NIPC.yaml#/components/schemas/Id"
                    - $ref: "#/components/schemas/Extension-BulkRespon\
se"
                examples:
                  bulkResponse:
                    $ref: "#/components/examples/bulkResponse"
                  firmwareUpgradeResponse:
                    $ref: "#/components/examples/firmwareUpgradeRespon\
se"
                  errorBulkResponse:
                    $ref: "#/components/examples/errorBulkResponse"

    /extensions/{id}/bulk/status:
      get:
        tags:
          - NIPC API extensions
        summary: Get Bulk status
        description: Get Bulk status
        operationId: getBulkStatus
        parameters:
          - name: id
            in: path
            description: The ID of the device. Group ID is not allowed.
            required: true
            schema:
              type: string
              format: uuid
              example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
          - name: instanceId
            in: query
            description: Instance ID of the bulk operation
            required: true
            schema:
```

```
              type: string
              format: uuid
              example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        responses:
          "200":
            description: OK
            headers:
            content:
              application/json:
                schema:
                  allOf:
                    - $ref: "./Extension-Async.yaml#/components/schema\
s/Extension-StatusResponse"
          "303":
            description: See Other
            headers:
              Location:
                schema:
                  type: string
                  example: /extensions/1d3b2c36-8a65-45a6-87c1-bcdbe0a\
32e30/bulk?instanceId=02ee282c-8915-4b2e-bbd2-88966773134a
                description: URL to get the bulk response
            content:
              application/json:
                schema:
                  allOf:
                    - $ref: "./Extension-Async.yaml#/components/schema\
s/Extension-StatusResponse"
                examples:
                  successExample:
                    summary: Success
                    value:
                      status: COMPLETED

components:
  schemas:
    # Extensions
    ## Bulk schema Extension
    Extension-Bulk:
      allOf:
        - $ref: "./Extension-Async.yaml#/components/schemas/Extensio\
n-Callback"
        - type: object
          properties:
            operations:
              type: array
              items:
                $ref: "#/components/schemas/Extension-BulkOperation"
```

```
              trigger:
                type: string
                description: |-
                  When to trigger the operations. If not specified, th\
   e operations are triggered immediately.
                default: immediate
                enum:
                  - immediate
                  - advertisement

      ## Extension that defines an operation in a bulk API
      Extension-BulkOperation:
        required:
          - method
          - path
        allOf:
          - type: object
            properties:
              method:
                type: string
                enum:
                  - POST
                  - PUT
                  - GET
              path:
                type: string
                enum:
                  - /devices/{id}/properties?propertyName={propertyNam\
   e}
                  - /devices/{id}/actions/?actionName={actionName}
                  - /extensions/{id}/properties/read/conditional?prope\
   rtyName={propertyName}
                  - /extensions/{id}/events/conditional?eventName={eve\
   ntName}
                  - /extensions/{id}/properties/file?propertyName={pro\
   pertyName}
                example: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30\
   /properties?propertyName=https%3A%2F%2Fexample.com%2Fthermometer%23%\
   2FsdfThing%2Fthermometer%2FsdfProperty%2Ftemperature
              data:
                type: object
                oneOf:
                  - $ref: "../NIPC.yaml#/components/schemas/Value"
                  - $ref: "./Extension-ReadConditional.yaml#/component\
   s/schemas/Extension-ConditionalRead"

      ## Multiple returns for a bulk operation
      Extension-BulkResponse:
```

```
        type: object
        properties:
          operations:
            type: array
            items:
              $ref: "#/components/schemas/Extension-OperationResponse"

      ## Return for an operation
      Extension-OperationResponse:
        allOf:
          - type: object
            properties:
              method:
                type: string
                enum:
                  - POST
                  - PUT
                  - GET
              path:
                type: string
                enum:
                  - /devices/{id}/properties?propertyName={propertyNam\
  e}
                  - /devices/{id}/actions/?actionName={actionName}
                  - /extensions/{id}/properties/read/conditional?prope\
  rtyName={propertyName}
                  - /extensions/{id}/events/conditional?eventName={eve\
  ntName}
                example: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30\
  /properties?propertyName=https%3A%2F%2Fexample.com%2Fthermometer%23%\
  2FsdfThing%2Fthermometer%2FsdfProperty%2Ftemperature
              response:
                anyOf:
                  - $ref: "../NIPC.yaml#/components/schemas/Value"
                  - $ref: "../NIPC.yaml#/components/schemas/SuccessRes\
  ponse"
                  - $ref: "../NIPC.yaml#/components/schemas/FailureRes\
  ponse"

    examples:
      bulkRequest:
        summary: Bulk request example
        value:
          operations:
            - method: GET
              path: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/prop\
  erties?propertyName=https%3A%2F%2Fexample.com%2Fthermometer%23%2Fsdf\
  Thing%2Fthermometer%2FsdfProperty%2Ftemperature
```

```
            - method: PUT
              path: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/prop\
   erties?propertyName=https%3A%2F%2Fexample.com%2Fthermometer%23%2Fsdf\
   Thing%2Fthermometer%2FsdfProperty%2Ftemperature
              data:
                value: dGVzdA==
            - method: POST
              path: /extensions/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/p\
   roperties/read/conditional?propertyName=https%3A%2F%2Fexample.com%2F\
   thermometer%23%2FsdfThing%2Fthermometer%2FsdfProperty%2Ftemperature
              data:
                value: dGVzdA==
                maxRepeat: 5
                retryTime: 1
        bulkResponse:
          summary: Bulk response example
          value:
            operations:
              - method: GET
                path: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/prop\
   erties?propertyName=https%3A%2F%2Fexample.com%2Fthermometer%23%2Fsdf\
   Thing%2Fthermometer%2FsdfProperty%2Ftemperature
                response:
                  value: dGVzdA==
              - method: PUT
                path: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/prop\
   erties?propertyName=https%3A%2F%2Fexample.com%2Fthermometer%23%2Fsdf\
   Thing%2Fthermometer%2FsdfProperty%2Ftemperature
                response:
                  status: 200
              - method: POST
                path: /extensions/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/p\
   roperties/read/conditional?propertyName=https%3A%2F%2Fexample.com%2F\
   thermometer%23%2FsdfThing%2Fthermometer%2FsdfProperty%2Ftemperature
                response:
                  value: dGVzdA==
        errorBulkResponse:
          summary: Error Bulk response example
          value:
            operations:
              - method: GET
                path: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/prop\
   erties?propertyName=https%3A%2F%2Fexample.com%2Fthermometer%23%2Fsdf\
   Thing%2Fthermometer%2FsdfProperty%2Ftemperature
                response:
                  type: https://www.iana.org/assignments/nipc-problem-ty\
   pes#property-not-readable
                  status: 400
```

```
                title: Property not readable
                detail: Property https://example.com/thermometer#/sdfT\
   hing/thermometer/sdfProperty/temperature is not readable
            - method: PUT
              path: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/prop\
   erties?propertyName=https%3A%2F%2Fexample.com%2Fthermometer%23%2Fsdf\
   Thing%2Fthermometer%2FsdfProperty%2Ftemperature
              response:
                type: https://www.iana.org/assignments/nipc-problem-ty\
   pes#extension-operation-not-executed
                status: 400
                title: Operation not executed
                detail: Operation was not executed since the previous \
   operation failed
            - method: POST
              path: /extensions/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/p\
   roperties/read/conditional?propertyName=https%3A%2F%2Fexample.com%2F\
   thermometer%23%2FsdfThing%2Fthermometer%2FsdfProperty%2Ftemperature
              response:
                type: https://www.iana.org/assignments/nipc-problem-ty\
   pes#extension-operation-not-executed
                status: 400
                title: Operation not executed
                detail: Operation was not executed since the previous \
   operation failed
        firmwareUpgradeRequest:
          summary: Firmware upgrade request example
          value:
            operations:
              - method: PUT
                path: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/prop\
   erties?propertyName=https%3A%2F%2Fexample.com%2Fthermometer%23%2Fsdf\
   Thing%2Fthermometer%2FsdfProperty%2FstartOTA
                data:
                  value: dGVzdA==
              - method: POST
                path: /extensions/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/e\
   vents/conditional?eventName=https%3A%2F%2Fexample.com%2Fthermometer%\
   23%2FsdfThing%2Fthermometer%2FsdfEvent%2FotaStarted
                data:
                  value: MQ==
                  timeout: 5
              - method: PUT
                path: /extensions/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/p\
   roperties/file?propertyName=https%3A%2F%2Fexample.com%2Fthermometer%\
   23%2FsdfThing%2Fthermometer%2FsdfProperty%2Ffirmware
                data:
                  fileURL: https://example.com/firmware.bin
```

```
                chunkSize: 20
                sha256Checksum: abcdef1234567890abcdef1234567890abcdef\
   1234567890abcdef1234567890
              - method: PUT
                path: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/prop\
   erties?propertyName=https%3A%2F%2Fexample.com%2Fthermometer%23%2Fsdf\
   Thing%2Fthermometer%2FsdfProperty%2FfinishOTA
                data:
                  value: dGVzdA==
              - method: POST
                path: /extensions/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/p\
   roperties/read/conditional?propertyName=https%3A%2F%2Fexample.com%2F\
   thermometer%23%2FsdfThing%2Fthermometer%2FsdfProperty%2FupdateStatus
                data:
                  value: MQ==
                  maxRepeat: 5
                  retryTime: 1
              - method: PUT
                path: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/prop\
   erties?propertyName=https%3A%2F%2Fexample.com%2Fthermometer%23%2Fsdf\
   Thing%2Fthermometer%2FsdfProperty%2FactivateOTA
                data:
                  value: dGVzdA==
        firmwareUpgradeResponse:
          summary: Firmware upgrade response example
          value:
            operations:
              - method: PUT
                path: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/prop\
   erties?propertyName=https%3A%2F%2Fexample.com%2Fthermometer%23%2Fsdf\
   Thing%2Fthermometer%2FsdfProperty%2FstartOTA
                response:
                  status: 200
              - method: POST
                path: /extensions/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/e\
   vents/conditional?eventName=https%3A%2F%2Fexample.com%2Fthermometer%\
   23%2FsdfThing%2Fthermometer%2FsdfEvent%2FotaStarted
                response:
                  value: MQ==
              - method: PUT
                path: /extensions/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/p\
   roperties/file?propertyName=https%3A%2F%2Fexample.com%2Fthermometer%\
   23%2FsdfThing%2Fthermometer%2FsdfProperty%2Ffirmware
                response:
                  status: 204
              - method: PUT
                path: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/prop\
   erties?propertyName=https%3A%2F%2Fexample.com%2Fthermometer%23%2Fsdf\
```

```
   Thing%2Fthermometer%2FsdfProperty%2FfinishOTA
               response:
                 status: 200
             - method: POST
               path: /extensions/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/p\
   roperties/read/conditional?propertyName=https%3A%2F%2Fexample.com%2F\
   thermometer%23%2FsdfThing%2Fthermometer%2FsdfProperty%2FupdateStatus
               response:
                 value: MQ==
             - method: PUT
               path: /devices/1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30/prop\
   erties?propertyName=https%3A%2F%2Fexample.com%2Fthermometer%23%2Fsdf\
   Thing%2Fthermometer%2FsdfProperty%2FactivateOTA
               data:
                 status: 200
   <CODE ENDS>
```

D.3. NIPC API write file extension

```
   <CODE BEGINS> file "Extension-File.yaml"
   =============== NOTE: '\\' line wrapping per RFC 8792 ===============

   # yaml-language-server: $schema=https://json-schema.org/draft/2020-1\
   \2/schema
   openapi: 3.0.3
   info:
     title: Non IP Device Control (NIPC) API write file extension
     description: |-
       Non IP Device Control (NIPC) API write file extension
     termsOfService: http://swagger.io/terms/
     contact:
       email: bbrinckm@cisco.com
     license:
       name: TBD
       url: TBD
     version: 0.17.0
   externalDocs:
     description: NIPC IETF draft
     url: https://datatracker.ietf.org/doc/draft-ietf-asdf-nipc/
   servers:
     - url: "{gw_host}/nipc/draft-17"
       variables:
         gw_host:
           default: localhost
           description: Gateway Host
   tags:
     - name:  NIPC API extensions
       description: |-
```

```
        APIs that simplify application interaction by implementing
        one or more basic APIs into a single API call.

   paths:
   ### Extensions
     /extensions/{id}/properties/file:
       put:
         tags:
           - NIPC API extensions
         summary: Write a file to a property on a device
         description: |-
           Write a file to a property on a device. Will chunk up the
           file and perform multiple writes. If the underlying protocol
           requires a connection to be set up, this API call will
           perform the necessary connection management. If a connection
           is already active for this device, the existing connection
           will be leveraged without modifying it. ID cannot be a
           group-id.
         operationId: writeFile
         parameters:
         - name: id
           in: path
           description: The ID of the device. Group ID is not allowed.
           required: true
           schema:
             type: string
             format: uuid
             example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
         - name: propertyName
           in: query
           description: |-
             The SDF property name that needs to be written to.
           required: true
           schema:
             type: string
             example: "https://example.com/heartrate#/sdfObject/thermos\
   \tat/sdfProperty/firmware"
         requestBody:
           content:
             application/json:
               schema:
                 allOf:
                   - $ref: '#/components/schemas/Extension-File'
                   - $ref: './Extension-Async.yaml#/components/schemas/\
   \Extension-Callback'
           required: true
         responses:
           '202':
```

```
            description: Accepted
            headers:
              Location:
                schema:
                  type: string
                  example: /extensions/1d3b2c36-8a65-45a6-87c1-bcdbe0a\
  \32e30/properties/file/status?propertyName=https%3A%2F%2Fexample.com\
  \%2Fheartrate%23%2FsdfObject%2Fthermostat%2FsdfProperty%2Ffirmware&i\
  \nstanceId=02ee282c-8915-4b2e-bbd2-88966773134a
                description: |-
                  URL to get the file write status
              Retry-After:
                schema:
                  type: integer
                description: |-
                  Time in seconds to wait before retrying
          'default':
            description: Error response
            content:
              application/json:
                schema:
                  $ref: '../NIPC.yaml#/components/schemas/FailureRespo\
  \nse'
        callbacks:
          callbackEvent:
            "{$request.body#/callback.url}":
              post:
                requestBody:
                  content:
                    application/json:
                      schema:
                        anyOf:
                          - allOf:
                            - $ref: '../NIPC.yaml#/components/schemas/\
  \Id'
                            - $ref: '../NIPC.yaml#/components/schemas/\
  \PropertyValue'
                          - $ref: '../NIPC.yaml#/components/schemas/Fa\
  \ilureResponse'
                      examples:
                        successExample:
                          summary: Success
                          value:
                            id: 02ee282c-8915-4b2e-bbd2-88966773134a
                            property: https://example.com/heartrate#/s\
  \dfObject/thermostat/sdfProperty/firmware
                        failedResponse:
                          summary: Failed
```

```
                          value:
                            id: 02ee282c-8915-4b2e-bbd2-88966773134a
                            status: 400
                            type: https://www.iana.org/assignments/nip\
   \c-problem-types#invalid-id
                            title: Invalid ID
                            detail: "Invalid request"
                            property: https://example.com/heartrate#/s\
   \dfObject/thermostat/sdfProperty/firmware
                responses:
                  '200':
                    description: Success
      get:
        tags:
          - NIPC API extensions
        summary: Get the status of a file write operation
        description: |-
          Get the status of a file write operation. This will return
          the status of the file write operation, including any errors
          that may have occurred.
        operationId: getFileWriteStatus
        parameters:
        - name: id
          in: path
          description: The ID of the device. Group ID is not allowed.
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: propertyName
          in: query
          description: |-
            The SDF property name that needs to be written to.
          required: true
          schema:
            type: string
            example: "https://example.com/heartrate#/sdfObject/thermos\
   \tat/sdfProperty/firmware"
        - name: instanceId
          in: query
          description: |-
            The Instance ID for the file write operation.
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
```

```
        responses:
          '204':
            description: Success, no content
          default:
            description: Error response
            content:
              application/json:
                schema:
                  $ref: '../NIPC.yaml#/components/schemas/FailureRespo\
  \nse'
    /extensions/{id}/properties/file/status:
      get:
        tags:
          - NIPC API extensions
        summary: Get the status of a file write operation
        description: |-
          Get the status of a file write operation. This will return
          the status of the file write operation, including any errors
          that may have occurred.
        operationId: getFileWriteStatus
        parameters:
        - name: id
          in: path
          description: The ID of the device. Group ID is not allowed.
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: propertyName
          in: query
          description: |-
            The SDF property name that needs to be written to.
          required: true
          schema:
            type: string
            example: "https://example.com/heartrate#/sdfObject/thermos\
  \tat/sdfProperty/firmware"
        - name: instanceId
          in: query
          description: |-
            The Instance ID for the file write operation.
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        responses:
```

```
          '200':
            description: Success, returns the status of the file write\
   \ operation.
            content:
              application/json:
                schema:
                  $ref: './Extension-Async.yaml#/components/schemas/Ex\
   \tension-StatusResponse'
          '303':
            description: See Other
            headers:
              Location:
                schema:
                  type: string
                  example: /extensions/1d3b2c36-8a65-45a6-87c1-bcdbe0a\
   \32e30/properties/file?propertyName=https%3A%2F%2Fexample.com%2Fhear\
   \trate%23%2FsdfObject%2Fthermostat%2FsdfProperty%2Ffirmware&instance\
   \Id=02ee282c-8915-4b2e-bbd2-88966773134a
                description: URL to get the file write response
            content:
              application/json:
                schema:
                  $ref: './Extension-Async.yaml#/components/schemas/Ex\
   \tension-StatusResponse'
                examples:
                  successExample:
                    summary: Completed
                    value:
                      id: 02ee282c-8915-4b2e-bbd2-88966773134a
                      status: COMPLETED

   components:
     schemas:
   # Extensions
   ## A File Extension
       Extension-File:
         required:
           - fileURL
         type: object
         properties:
           fileURL:
             type: string
             example: "https://domain.com/firmware.dat"
             description: |-
               URL to the firmware file.
               The HTTP method is assumed to be a GET.
           chunkSize:
             type: integer
```

```
            sha256Checksum:
              type: string
              description: firmware checksum
            headers:
              type: object
              additionalProperties:
                type: string
              example:
                x-api-key: fjelk-3dl33f-2wdsd
   <CODE ENDS>
```

D.4. NIPC API conditional read extension

```
   <CODE BEGINS> file "Extension-ReadConditional.yaml"
   =============== NOTE: '\' line wrapping per RFC 8792 ================

   # yaml-language-server: $schema=https://json-schema.org/draft/2020-1\
   2/schema
   openapi: 3.0.3
   info:
     title: Non IP Device Control (NIPC) API read conditional extension
     description: |-
       Non IP Device Control (NIPC) API read conditional extension
     termsOfService: http://swagger.io/terms/
     contact:
       email: bbrinckm@cisco.com
     license:
       name: TBD
       url: TBD
     version: 0.17.0
   externalDocs:
     description: NIPC IETF draft
     url: https://datatracker.ietf.org/doc/draft-ietf-asdf-nipc/
   servers:
     - url: "{gw_host}/nipc/draft-17"
       variables:
         gw_host:
           default: localhost
           description: Gateway Host
   tags:
     - name:  NIPC API extensions
       description: |-
         APIs that simplify application interaction by implementing
         one or more basic APIs into a single API call.

   paths:
   ### Extensions
     /extensions/{id}/properties/read/conditional:
```

```
      post:
        tags:
          - NIPC API extensions
        summary: Conditional read of a property
        description: Conditional read of a property
        operationId: conditionalRead
        parameters:
        - name: id
          in: path
          description: The ID of the device. Group ID is not allowed.
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: propertyName
          in: query
          description: |-
            The SDF property name that needs to be read conditionally.
          required: true
          allowReserved: true
          schema:
            type: string
            example: "#/sdfObject/thermostat/sdfProperty/temperature"
        requestBody:
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Extension-ConditionalRead'
          required: true
        responses:
          '202':
            description: Accepted
            headers:
              Location:
                schema:
                  type: string
                  example: /extensions/1d3b2c36-8a65-45a6-87c1-bcdbe0a\
   32e30/properties/read/conditional/status?propertyName=https%3A%2F%2F\
   example.com%2Fheartrate%23%2FsdfObject%2Fthermostat%2FsdfProperty%2F\
   temperature&instanceId=02ee282c-8915-4b2e-bbd2-88966773134a
                description: |-
                  URL to get the conditional read status
              Retry-After:
                schema:
                  type: integer
                description: |-
                  Time in seconds to wait before retrying
```

```
          'default':
            description: Error response
            content:
              application/json:
                schema:
                  $ref: '../NIPC.yaml#/components/schemas/FailureRespo\
   nse'
      callbacks:
        callbackEvent:
          "{$request.body#/callback.url}":
            post:
              requestBody:
                content:
                  application/json:
                    schema:
                      anyOf:
                        - allOf:
                          - $ref: '../NIPC.yaml#/components/schemas/\
   Id'
                          - $ref: '../NIPC.yaml#/components/schemas/\
   PropertyValue'
                        - $ref: '../NIPC.yaml#/components/schemas/Fa\
   ilureResponse'
                    examples:
                      successExample:
                        summary: Success
                        value:
                          id: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
                          property: https://example.com/heartrate#/s\
   dfObject/thermostat/sdfProperty/temperature
                          value: dGVzdA==
                      failedResponse:
                        summary: Failed
                        value:
                          id: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
                          status: 400
                          type: https://www.iana.org/assignments/nip\
   c-problem-types#invalid-id
                          title: Invalid ID
                          detail: "Invalid request"
                          property: https://example.com/heartrate#/s\
   dfObject/thermostat/sdfProperty/temperature
                          value: dGVzdA==

              responses:
                '200':
                  description: Success
    get:
```

```
        tags:
          - NIPC API extensions
        summary: Get Conditional read response of a property
        description: Conditional read response of a property
        operationId: getConditionalRead
        parameters:
        - name: id
          in: path
          description: The ID of the device. Group ID is not allowed.
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: propertyName
          in: query
          description: |-
            The SDF property name that needs to be read conditionally.
          required: true
          allowReserved: true
          schema:
            type: string
            example: "#/sdfObject/thermostat/sdfProperty/temperature"
        - name: instanceId
          in: query
          description: |-
            Instance ID of the conditional read operation
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        responses:
          '200':
            description: Success
            headers:
            content:
              application/json:
                schema:
                  allOf:
                    - $ref: '../NIPC.yaml#/components/schemas/Value'
              application/octet-stream:
                schema:
                  type: string
                  format: binary
                description: Binary data of the property value
          default:
            description: Error response
```

```
            content:
              application/problem+json:
                schema:
                  allOf:
                    - $ref: '../NIPC.yaml#/components/schemas/FailureR\
  esponse'
    /extensions/{id}/properties/read/conditional/status:
      get:
        tags:
          - NIPC API extensions
        summary: Get Conditional read status of a property
        description: Conditional read status of a property
        operationId: getConditionalReadStatus
        parameters:
        - name: id
          in: path
          description: The ID of the device. Group ID is not allowed.
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: propertyName
          in: query
          description: |-
            The SDF property name that needs to be read conditionally.
          required: true
          allowReserved: true
          schema:
            type: string
            example: "#/sdfObject/thermostat/sdfProperty/temperature"
        - name: instanceId
          in: query
          description: Instance ID of the conditional read operation
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        responses:
          '200':
            description: OK
            headers:
            content:
              application/json:
                schema:
                  $ref: './Extension-Async.yaml#/components/schemas/Ex\
  tension-StatusResponse'
```

```
          '303':
            description: See Other
            headers:
              Location:
                schema:
                  type: string
                  example: /extensions/1d3b2c36-8a65-45a6-87c1-bcdbe0a\
   32e30/properties/read/conditional?propertyName=https%3A%2F%2Fexample\
   .com%2Fheartrate%23%2FsdfObject%2Fthermostat%2FsdfProperty%2Ftempera\
   ture&instanceId=02ee282c-8915-4b2e-bbd2-88966773134a
                description: URL to get the conditional read response
            content:
              application/json:
                schema:
                  $ref: './Extension-Async.yaml#/components/schemas/Ex\
   tension-StatusResponse'
                examples:
                  successExample:
                    summary: Completed
                    value:
                      id: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
                      status: COMPLETED

   components:
     schemas:
   # Extensions
       Extension-ConditionalRead:
         allOf:
           - $ref: '../NIPC.yaml#/components/schemas/Value'
           - $ref: './Extension-Async.yaml#/components/schemas/Extensio\
   n-Callback'
           - type: object
             properties:
               maxRepeat:
                 description: |-
                   maximum time the conditional read should repeat
                   (default 5, max 10)
                 type: integer
                 example: 5
               retryTime:
                 description: |-
                   time between reads in seconds (default 1, max 10)
                 type: integer
                 example: 1
   <CODE ENDS>
```

D.5. NIPC API conditional event extension

```
<CODE BEGINS> file "Extension-EventConditional.yaml"
=============== NOTE: '\' line wrapping per RFC 8792 ================

# yaml-language-server: $schema=https://json-schema.org/draft/2020-1\
2/schema
openapi: 3.0.3
info:
  title: Non IP Device Control (NIPC) API event conditional extension
  description: |-
    Non IP Device Control (NIPC) API event conditional extension
  termsOfService: http://swagger.io/terms/
  contact:
    email: bbrinckm@cisco.com
  license:
    name: TBD
    url: TBD
  version: 0.17.0
externalDocs:
  description: NIPC IETF draft
  url: https://datatracker.ietf.org/doc/draft-ietf-asdf-nipc/
servers:
  - url: "{gw_host}/nipc/draft-17"
    variables:
      gw_host:
        default: localhost
        description: Gateway Host
tags:
  - name:  NIPC API extensions
    description: |-
      APIs that simplify application interaction by implementing
      one or more basic APIs into a single API call.

paths:
### Extensions
  /extensions/{id}/events/conditional:
    post:
      tags:
        - NIPC API extensions
      summary: Enable an event until a condition is met
      description: Enable an event until a condition is met
      operationId: conditionalEvent
      parameters:
      - name: id
        in: path
        description: The ID of the device. Group ID is not allowed.
        required: true
        schema:
          type: string
```

```
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: eventName
          in: query
          description: |-
            The SDF event name that needs to be enabled.
          required: true
          allowReserved: true
          schema:
            type: string
            example: "#/sdfObject/thermostat/sdfEvent/temperature"
        requestBody:
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Extension-ConditionalEvent'
          required: true
        responses:
          '202':
            description: Accepted
            headers:
              Location:
                schema:
                  type: string
                  example: /extensions/1d3b2c36-8a65-45a6-87c1-bcdbe0a\
   32e30/events/conditional/status?eventName=https%3A%2F%2Fexample.com%\
   2Fheartrate%23%2FsdfObject%2Fthermostat%2FsdfEvent%2Ftemperature&ins\
   tanceId=02ee282c-8915-4b2e-bbd2-88966773134a
                description: |-
                  URL to get the conditional event status
              Retry-After:
                schema:
                  type: integer
                description: |-
                  Time in seconds to wait before retrying
          'default':
            description: Error response
            content:
              application/json:
                schema:
                  $ref: '../NIPC.yaml#/components/schemas/FailureRespo\
   nse'
        callbacks:
          callbackEvent:
            "{$request.body#/callback.url}":
              post:
                requestBody:
                  content:
```

```
                    application/json:
                      schema:
                        anyOf:
                          - allOf:
                            - $ref: '../NIPC.yaml#/components/schemas/\
   Id'
                            - $ref: '../NIPC.yaml#/components/schemas/\
   PropertyValue'
                          - $ref: '../NIPC.yaml#/components/schemas/Fa\
   ilureResponse'
                      examples:
                        successExample:
                          summary: Success
                          value:
                            id: 02ee282c-8915-4b2e-bbd2-88966773134a
                            event: https://example.com/heartrate#/sdfO\
   bject/thermostat/sdfEvent/temperature
                            value: dGVzdA==
                        failedResponse:
                          summary: Failed
                          value:
                            id: 02ee282c-8915-4b2e-bbd2-88966773134a
                            status: 400
                            type: https://www.iana.org/assignments/nip\
   c-problem-types#invalid-id
                            title: Invalid ID
                            detail: "Invalid request"
                            event: https://example.com/heartrate#/sdfO\
   bject/thermostat/sdfEvent/temperature
                            value: dGVzdA==

               responses:
                 '200':
                   description: Success
    get:
      tags:
        - NIPC API extensions
      summary: Get Conditional event response
      description: Conditional event response
      operationId: getConditionalEvent
      parameters:
      - name: id
        in: path
        description: The ID of the device. Group ID is not allowed.
        required: true
        schema:
          type: string
          format: uuid
```

```
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: eventName
          in: query
          description: |-
            The name of the SDF event that is enabled.
          required: true
          allowReserved: true
          schema:
            type: string
            example: "#/sdfObject/thermostat/sdfEvent/temperature"
        - name: instanceId
          in: query
          description: |-
            Instance ID of the conditional event operation
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        responses:
          '200':
            description: Success
            headers:
            content:
              application/json:
                schema:
                  allOf:
                    - $ref: '../NIPC.yaml#/components/schemas/Value'
              application/octet-stream:
                schema:
                  type: string
                  format: binary
                description: Binary data of the event value
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  allOf:
                    - $ref: '../NIPC.yaml#/components/schemas/FailureR\
  esponse'
    /extensions/{id}/events/conditional/status:
      get:
        tags:
          - NIPC API extensions
        summary: Get Conditional event status
        description: Conditional event status
        operationId: getConditionalEventStatus
```

```
        parameters:
        - name: id
          in: path
          description: The ID of the device. Group ID is not allowed.
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        - name: eventName
          in: query
          description: |-
            The name of the SDF event that is enabled.
          required: true
          allowReserved: true
          schema:
            type: string
            example: "#/sdfObject/thermostat/sdfEvent/temperature"
        - name: instanceId
          in: query
          description: Instance ID of the conditional event operation
          required: true
          schema:
            type: string
            format: uuid
            example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        responses:
          '200':
            description: OK
            headers:
            content:
              application/json:
                schema:
                  $ref: './Extension-Async.yaml#/components/schemas/Ex\
   tension-StatusResponse'
          '303':
            description: See Other
            headers:
              Location:
                schema:
                  type: string
                  example: /extensions/1d3b2c36-8a65-45a6-87c1-bcdbe0a\
   32e30/events/conditional?eventName=https%3A%2F%2Fexample.com%2Fheart\
   rate%23%2FsdfObject%2Fthermostat%2FsdfEvent%2Ftemperature&instanceId\
   =02ee282c-8915-4b2e-bbd2-88966773134a
                description: URL to get the conditional event response
            content:
              application/json:
```

```
                schema:
                  $ref: './Extension-Async.yaml#/components/schemas/Ex\
   tension-StatusResponse'
                examples:
                  successExample:
                    summary: Completed
                    value:
                      id: 02ee282c-8915-4b2e-bbd2-88966773134a
                      status: COMPLETED

   components:
     schemas:
   # Extensions
       Extension-ConditionalEvent:
         allOf:
           - $ref: './Extension-Async.yaml#/components/schemas/Extensio\
   n-Callback'
           - $ref: '../NIPC.yaml#/components/schemas/Value'
           - type: object
             properties:
               timeout:
                 description: |-
                   Time in seconds to keep the event enabled.
                   If the event condition is not met within this time,
                   the event will be disabled and marked as failed.
                 type: integer
                 example: 5
   <CODE ENDS>
```

D.6. NIPC API property extensions

```
   <CODE BEGINS> file "Extension-Property.yaml"
   =============== NOTE: '\' line wrapping per RFC 8792 ================

   # yaml-language-server: $schema=https://json-schema.org/draft/2020-1\
   2/schema
   openapi: 3.0.3
   info:
     title: Non IP Device Control (NIPC) API properties extension
     description: |-
       Non IP Device Control (NIPC) API properties extension
     termsOfService: http://swagger.io/terms/
     contact:
       email: bbrinckm@cisco.com
     license:
       name: TBD
       url: TBD
     version: 0.17.0
```

```
   externalDocs:
     description: NIPC IETF draft
     url: https://datatracker.ietf.org/doc/draft-ietf-asdf-nipc/
   servers:
     - url: "{gw_host}/nipc/draft-17"
       variables:
         gw_host:
           default: localhost
           description: Gateway Host
   tags:
     - name:  NIPC API extensions
       description: |-
         APIs that simplify application interaction by implementing
         one or more basic APIs into a single API call.

   paths:
   ### Extensions
     /extensions/{id}/transmit:
       post:
         tags:
           - NIPC API extensions
         summary: Transmit to a device
         description: |-
           Transmit a payload to a device. The transmission is performe\
   d on the AP where the device was last seen
         operationId: ActionTransmit
         parameters:
           - name: id
             in: path
             description: The ID of the device. Group ID is not allowed.
             required: true
             schema:
               type: string
               format: uuid
               example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
         requestBody:
           content:
             application/json:
               schema:
                 $ref: '#/components/schemas/Transmit'
           required: true
         responses:
           '200':
             description: Success
           default:
             description: Error response
             content:
               application/problem+json:
```

```
                schema:
                  $ref: '../NIPC.yaml#/components/schemas/FailureRespo\
   nse'

    /extensions/{id}/properties/write:
      post:
        tags:
          - NIPC API extensions
        summary: Write a value to an property using protocol mapping
        description: |-
          Write a value to an unregistered property, embedding property
          protocol mapping in the API, this does not require
          property registration. You cannot write to a group id.
        operationId: ActionPropWrite
        parameters:
          - name: id
            in: path
            description: The ID of the device. Group ID is not allowed.
            required: true
            schema:
              type: string
              format: uuid
              example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        requestBody:
          content:
            application/json:
              schema:
                allOf:
                  - $ref: '../NIPC.yaml#/components/schemas/Value'
                  - $ref: '../protocolmaps/ProtocolMap.yaml#/component\
   s/schemas/ProtocolMap-Property'
          required: true
        responses:
          '204':
            description: Success, no content
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  $ref: '../NIPC.yaml#/components/schemas/FailureRespo\
   nse'

    /extensions/{id}/properties/read:
      post:
        tags:
          - NIPC API extensions
        summary: Read a value to an property using protocol mapping
```

```
        description: |-
          Read a value from an unregistered property, embedding
          property protocol mapping in the API, this does not require
          property registration. You cannot read from a group id.
        operationId: ActionPropRead
        parameters:
          - name: id
            in: path
            description: The ID of the device. Group ID is not allowed.
            required: true
            schema:
              type: string
              format: uuid
              example: 1d3b2c36-8a65-45a6-87c1-bcdbe0a32e30
        requestBody:
          content:
            application/json:
              schema:
                $ref: '../protocolmaps/ProtocolMap.yaml#/components/sc\
  hemas/ProtocolMap-Property'
          required: true
        responses:
          '200':
            description: Success
            content:
              application/json:
                schema:
                  allOf:
                    - $ref: '../NIPC.yaml#/components/schemas/Value'
          default:
            description: Error response
            content:
              application/problem+json:
                schema:
                  $ref: '../NIPC.yaml#/components/schemas/FailureRespo\
  nse'

  components:
    schemas:
      Transmit:
        allOf:
          - $ref: '../protocolinfo/ProtocolInfo.yaml#/components/schem\
  as/ProtocolInfo-Broadcast'
        required:
          - cycle
        type: object
        properties:
          cycle:
```

```
            type: string
            example: single
            enum:
              - single
              - repeat
          # transmit time in ms
          transmitTime:
            type: integer
            example: 3000
          # interval between transmits in ms
          transmitInterval:
            type: integer
            example: 500
          payload:
            type: string
            format: byte
            example: AgEaAgoMFv9MABAHch9BsDkgeA==
```

<CODE ENDS>

Appendix E. NIPC API CDDL Definition

The following is a combined reference of all NIPC API CDDL definitions used in this document.

<CODE BEGINS> file "combined.cddl"

```
=============== NOTE: '\' line wrapping per RFC 8792 ================

; This file is auto-generated from individual NIPC API CDDL files

; ============================================
; From: cddl/api/action_response.cddl
; ============================================
ActionResponse = {
  status: ActionStatus
}

ActionStatus = "IN_PROGRESS" / "COMPLETED"
; ============================================
; From: cddl/api/action.cddl
; ============================================
Action = {
  ? action: text  ; NIPC action operation to execute
}

; ============================================
; From: cddl/api/data_app.cddl
; ============================================
DataApp = {
```

```
    events: [* EventRef],
    ( DataAppMqttClient //
      DataAppMqttBroker //
      DataAppWebhook //
      DataAppWebsocket )
  }

  EventRef = {
    event: text    ; SDF global name (absolute URI with fragment)
  }

  DataAppMqttClient = {
    mqttClient: bool
  }

  DataAppMqttBroker = {
    mqttBroker: {
      URI: text,
      username: text,
      password: text,
      ? brokerCACert: text,   ; PEM-encoded CA certificate
      ? customTopic: text     ; optional custom MQTT topic
    }
  }

  DataAppWebhook = {
    webhook: {
      URI: text,
      ? headers: { * text => text },  ; key/value headers
      ? serverCACert: text
    }
  }

  DataAppWebsocket = {
    websocket: {
      URI: text,
      ? headers: { * text => text },  ; key/value headers
      ? serverCACert: text
    }
  }
  ; ===========================================
  ; From: cddl/api/event_status_array.cddl
  ; ===========================================
  EventStatusResponseArray = [* EventStatusResponseArrayItem]

  EventStatusResponseArrayItem = ( EventInstanceSuccess // FailureResp\
  onse )
```

```
   ; Success item = { event, instanceId }
   EventInstanceSuccess = {
     event: text,        ; SDF global name of the event (absolute URI w\
   ith fragment)
     instanceId: text    ; UUID (as text)
   }

   ; ================================================
   ; From: cddl/api/failure_response.cddl
   ; ================================================
   FailureResponse = {
     ? type: FailureTypeURI,
     ? status: uint,
     ? title: text,
     ? detail: text
   }

   ; Enumerated problem type URIs registered for NIPC
   FailureTypeURI = (
     "https://www.iana.org/assignments/nipc-problem-types#invalid-id" /
     "https://www.iana.org/assignments/nipc-problem-types#invalid-sdf-u\
   rl" /
     "https://www.iana.org/assignments/nipc-problem-types#extension-ope\
   ration-not-executed" /
     "https://www.iana.org/assignments/nipc-problem-types#sdf-model-alr\
   eady-registered" /
     "https://www.iana.org/assignments/nipc-problem-types#sdf-model-in-\
   use" /
     "https://www.iana.org/assignments/nipc-problem-types#property-not-\
   readable" /
     "https://www.iana.org/assignments/nipc-problem-types#property-read\
   -failed" /
     "https://www.iana.org/assignments/nipc-problem-types#property-not-\
   writable" /
     "https://www.iana.org/assignments/nipc-problem-types#property-writ\
   e-failed" /
     "https://www.iana.org/assignments/nipc-problem-types#event-already\
   -enabled" /
     "https://www.iana.org/assignments/nipc-problem-types#event-not-ena\
   bled" /
     "https://www.iana.org/assignments/nipc-problem-types#event-not-reg\
   istered" /
     "https://www.iana.org/assignments/nipc-problem-types#protocolmap-b\
   le-already-connected" /
     "https://www.iana.org/assignments/nipc-problem-types#protocolmap-b\
   le-no-connection" /
     "https://www.iana.org/assignments/nipc-problem-types#protocolmap-b\
   le-connection-timeout" /
```

```
    "https://www.iana.org/assignments/nipc-problem-types#protocolmap-b\
  le-bonding-failed" /
    "https://www.iana.org/assignments/nipc-problem-types#protocolmap-b\
  le-connection-failed" /
    "https://www.iana.org/assignments/nipc-problem-types#protocolmap-b\
  le-service-discovery-failed" /
    "https://www.iana.org/assignments/nipc-problem-types#protocolmap-b\
  le-invalid-service-or-characteristic" /
    "https://www.iana.org/assignments/nipc-problem-types#protocolmap-z\
  igbee-connection-timeout" /
    "https://www.iana.org/assignments/nipc-problem-types#protocolmap-z\
  igbee-invalid-endpoint-or-cluster" /
    "https://www.iana.org/assignments/nipc-problem-types#extension-tra\
  nsmit-invalid-data" /
    "https://www.iana.org/assignments/nipc-problem-types#extension-fir\
  mware-rollback" /
    "https://www.iana.org/assignments/nipc-problem-types#extension-fir\
  mware-update-failed" /
    "about:blank"
  )
  ; ==========================================
  ; From: cddl/api/group_event_status_response_array.cddl
  ; ==========================================
  GroupEventStatusResponseArray = [* GroupEventStatusResponse]

  GroupEventSuccessResponse = { event: text, deviceId: text }

  ; Each item is either an event+deviceId success or a GroupFailureRes\
  ponse
  GroupEventStatusResponse = (GroupEventSuccessResponse // GroupFailur\
  eResponse)

  GroupFailureResponse = {
    FailureResponse,
    ? deviceId: text
  }

  ; ==========================================
  ; From: cddl/api/group_trigger_status_array.cddl
  ; ==========================================
  ; Group Trigger status response array and item shape

  GroupTriggerStatusResponseArray = [* GroupTriggerResponse]

  GroupTriggerResponse = (GroupTriggerSuccessResponse // GroupTriggerF\
  ailureResponse)

  GroupTriggerSuccessResponse = {
```

```
    SdfReference,
    ? action: text,      ; NIPC action operation to execute
    deviceId: text     ; UUID (as text)
  }

  GroupTriggerFailureResponse = {
    FailureResponse,
    ? deviceId: text
  }

  ; ==========================================
  ; From: cddl/api/trigger_status_array.cddl
  ; ==========================================
  ; Trigger status response array and item shape

  TriggerStatusResponseArray = [* TriggerResponse]

  TriggerResponse = {
    SdfReference,
    ? action: text,      ; NIPC action operation to execute
    instanceId: text     ; UUID (as text)
  }

  ; ==========================================
  ; From: cddl/api/property_value_array.cddl
  ; ==========================================
  PropertyValueArray = [* PropertyValue]

  ; Minimal PropertyValue shape (matches allOf of Property + Value)
  PropertyValue = {
    property: text,        ; SDF global name of the property
    value: b64text         ; base64-encoded bytes (RFC 4648 Section 5)
  }

  ; Helper type for base64-with-padding encoded text
  b64text = text
  ; ==========================================
  ; From: cddl/api/property_value_read_response_array.cddl
  ; ==========================================
  PropertyValueReadResponseArray = [* PropertyValueReadResponseArrayIt\
  em]

  PropertyValueReadResponseArrayItem = ( PropertyValue // FailureRespo\
  nse )

  ; ==========================================
  ; From: cddl/api/property_value_response_array.cddl
  ; ==========================================
```

```
   PropertyValueResponseArray = [* PropertyValueResponseArrayItem]

   PropertyValueResponseArrayItem = ( SuccessResponse // FailureRespons\
   e )

   ; Minimal success shape (may be extended)
   SuccessResponse = {
     ? status: uint
   }

   ; ===========================================
   ; From: cddl/api/sdf_reference.cddl
   ; ===========================================
   SdfGlobalName = text    ; absolute URI with fragment referencing an \
   sdfThing or sdfObject

   SdfReference = {
     sdfName: SdfGlobalName
   }

   SdfReferenceArray = [* SdfReference]
   ; ===========================================
   ; From: cddl/api/connection.cddl
   ; ===========================================
   ConnectionRequest = {
     ? retries: uint,
     ? protocol-info-service-map
   }

   ConnectionResponse = {
     id: text,  ; UUID of the connection
     ? protocol-info-service-map
   }

   ; ===========================================
   ; From: cddl/api/protocolinfo.cddl
   ; ===========================================
   ; Top-level wrappers
   protocol-info-service-map = (
     ? protocolInformation: ble-service-map / zigbee-service-map
   )

   protocol-info-broadcast = (
     ? protocolInformation: ble-broadcast / zigbee-broadcast
   )

   ; BLE protocol information
   ble-service-map = {
```

```
    ble: {
      ? services: [* ble-service],
      ? cached: bool,
      ? cacheExpiryDuration: int,
      ? autoUpdate: bool,
      ? bonding: bonding-type,
    }
  }

  bonding-type = "default" / "none" / "justworks" / "passkey" / "oob"

  ble-service = {
    serviceID: uuid,
    ? characteristics: [* ble-characteristic],
  }

  ble-characteristic = {
    characteristicID: uuid,
    ? flags: [* ble-flag],
    ? descriptors: [* ble-descriptor],
  }

  ble-flag = "read" / "write" / "notify"

  ble-descriptor = {
    descriptorID: uuid,
  }

  ble-broadcast = {
    ble: {
      ? connectable: bool,
    },
  }

  ; Zigbee protocol information
  zigbee-service-map = {
    zigbee: {
      ? endpoints: [* zigbee-endpoint],
    },
  }

  zigbee-endpoint = {
    endpointID: uint,
    ? clusters: [* zigbee-cluster],
  }

  zigbee-cluster = {
    clusterID: uint,
```

```
  ? properties: [* zigbee-property],
}

zigbee-property = {
  attributeID: uint,
  propertyType: uint,
}

zigbee-broadcast = {
  zigbee: {
  },
}

; Basic types
uuid = tstr .regexp "(?i)^[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-\
f]{4}-[0-9a-f]{12}$"
<CODE ENDS>
```

Appendix F. Example SDF model with protocol mappings for BLE

```
<CODE BEGINS> file "thermometer.sdf.json"
{
  "namespace": {
    "thermometer": "https://example.com/thermometer"
  },
  "defaultNamespace": "thermometer",
  "sdfThing": {
    "thermometer": {
      "sdfObject": {
        "health_thermometer": {
          "description": "Health Thermometer",
          "sdfProperty": {
            "temperature_type": {
              "description": "Temperature Type",
              "observable": false,
              "writable": false,
              "readable": true,
              "sdfProtocolMap": {
                "ble": {
                  "serviceID": "1809",
                  "characteristicID": "2A1D"
                }
              }
            },
            "measurement_interval": {
              "description": "Measurement Interval",
              "observable": false,
              "writable": false,
```

```
                "readable": true,
                "sdfProtocolMap": {
                  "ble": {
                    "serviceID": "1809",
                    "characteristicID": "2A21"
                  }
                }
              }
            },
            "sdfEvent": {
              "temperature_measurement": {
                "description": "Temperature Measurement",
                "sdfProtocolMap": {
                  "ble": {
                    "type": "gatt",
                    "serviceID": "1809",
                    "characteristicID": "2A1C"
                  }
                }
              },
              "intermediate_temperature": {
                "description": "Intermediate Temperature",
                "sdfProtocolMap": {
                  "ble": {
                    "type": "gatt",
                    "serviceID": "1809",
                    "characteristicID": "2A1E"
                  }
                }
              }
            }
          }
        },
        "description": "Generic Access, Device Information",
        "sdfProperty": {
          "device_name": {
            "description": "Device Name",
            "observable": false,
            "writable": true,
            "readable": true,
            "sdfProtocolMap": {
              "ble": {
                "serviceID": "1800",
                "characteristicID": "2A00"
              }
            }
          },
          "appearance": {
```

```
        "description": "Appearance",
        "observable": false,
        "writable": false,
        "readable": true,
        "sdfProtocolMap": {
          "ble": {
            "serviceID": "1800",
            "characteristicID": "2A01"
          }
        }
      },
      "manufacturer_name_string": {
        "description": "Manufacturer Name String",
        "observable": false,
        "writable": false,
        "readable": true,
        "sdfProtocolMap": {
          "ble": {
            "serviceID": "180A",
            "characteristicID": "2A29"
          }
        }
      },
      "model_number_string": {
        "description": "Model Number String",
        "observable": false,
        "writable": false,
        "readable": true,
        "sdfProtocolMap": {
          "ble": {
            "serviceID": "180A",
            "characteristicID": "2A24"
          }
        }
      },
      "hardware_revision_string": {
        "description": "Hardware Revision String",
        "observable": false,
        "writable": false,
        "readable": true,
        "sdfProtocolMap": {
          "ble": {
            "serviceID": "180A",
            "characteristicID": "2A27"
          }
        }
      },
      "firmware_revision_string": {
```

```
            "description": "Firmware Revision String",
            "observable": false,
            "writable": false,
            "readable": true,
            "sdfProtocolMap": {
              "ble": {
                "serviceID": "180A",
                "characteristicID": "2A26"
              }
            }
          },
          "system_id": {
            "description": "System ID",
            "observable": false,
            "writable": false,
            "readable": true,
            "sdfProtocolMap": {
              "ble": {
                "serviceID": "180A",
                "characteristicID": "2A23"
              }
            }
          }
        },
        "sdfEvent": {
          "isPresent": {
            "description": "BLE advertisements",
            "sdfProtocolMap": {
              "ble": {
                "type": "advertisements"
              }
            }
          },
          "isConnected": {
            "description": "BLE connection events",
            "sdfProtocolMap": {
              "ble": {
                "type": "connection_events"
              }
            }
          }
        }
      }
    }
  }
  <CODE ENDS>
```

Figure 36: Example SDF model with protocol mappings for BLE

Appendix G. Acknowledgements

This document relies on SDF models described in [RFC9880], as such, we are grateful to the authors of this document for putting their time and effort into defining SDF in depth, allowing us to make use of it. The authors would also like to thank the ASDF working group for their excellent feedback and steering of the document.

Authors' Addresses

Bart Brinckman
Cisco Systems
Brussels
Belgium
Email: bbrinckm@cisco.com

Rohit Mohan
Cisco Systems
170 West Tasman Drive
San Jose, 95134
United States of America
Email: rohitmo@cisco.com

Braeden Sanford
Philips
Cambridge,
United States of America
Email: braeden.sanford@philips.com