

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 29 January 2026

B. Brinckman
R. Mohan
Cisco Systems
B. Sanford
Philips
28 July 2025

An Application Layer Interface for Non-IP device control (NIPC)
draft-ietf-asdf-nipc-11

Abstract

This memo specifies RESTful application layer interface for gateways providing operations against non-IP devices, as well as a CBOR-based publish-subscribe interface for streaming data. The described interfaces are extensible. The specification also defines a protocol mapping function to map this interface to commonly used non-IP protocols.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 29 January 2026.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	4
1.1. Scope	4
1.2. Non-IP Gateway	5
1.3. Terminology	7
2. Architecture	7
2.1. Overview	7
2.2. Onboarding	8
2.3. Registrations	8
2.3.1. SDF model registrations	8
2.3.2. Data application registrations	8
2.4. Protocol mapping	8
2.5. NIPC Operations	10
2.5.1. Properties APIs	10
2.5.2. Actions	11
2.5.3. Events	11
2.5.4. Connection management for NIPC Operations	11
2.5.5. Extensions	11
2.6. Events publish subscribe interface	11
2.7. Protocols	12
2.7.1. NIPC APIs	12
2.7.2. NIPC publish/subscribe events	12
2.8. Paths	12
2.8.1. General	12
2.8.2. NIPC Registrations	13
2.8.3. NIPC Operations	14
2.9. Schema	14
2.9.1. SDF model registrations	14
2.9.2. NIPC Operations	14
2.9.3. SDF Name	15
2.9.4. Responses	15
3. NIPC Registrations	17
3.1. SDF model registrations APIs	17
3.1.1. Register an SDF model	17
3.1.2. Get all SDF models	17
3.1.3. Get an SDF model	18
3.1.4. Delete an SDF model	18
3.1.5. Update an SDF model	19

3.2. Data application registrations APIs	19
3.2.1. Register a data application	20
3.2.2. Update a data application	22
3.2.3. Get a data application	22
3.2.4. Delete a data application	23
4. NIPC APIs	23
4.1. NIPC Property APIs	24
4.1.1. Write multiple values	24
4.1.2. Read multiple values	26
4.2. NIPC Event APIs	27
4.2.1. Enable event reporting	28
4.2.2. Disable event reporting	29
4.2.3. Get status of one or more events	29
4.2.4. Enable event reporting on a group of devices	30
4.2.5. Disable event reporting on a group of devices	31
4.2.6. Get event status on a group of devices	32
4.3. NIPC Action APIs	32
4.3.1. Perform an action	33
4.3.2. Check action status	33
4.4. NIPC explicit connections management APIs	34
4.4.1. Connect to a device	34
4.4.2. Update a connection	38
4.4.3. Disconnect from a device	40
4.4.4. Get connection status	41
5. NIPC Extensibility	42
5.1. Protocol extensions	43
5.2. API extensions	43
6. NIPC Error Handling	44
7. Publish/Subscribe Interface	46
7.1. CDDL Definition	46
7.2. CBOR Examples	48
8. Examples	49
8.1. Property Read/Write	49
8.2. Enabling an Event	51
9. Security Considerations	54
9.1. API authorization	54
10. IANA Considerations	54
10.1. Media Type Registration	54
10.2. Protocol mapping	55
10.3. API extensions	56
10.4. Well-known URIs	57
10.5. Problem Details for NIPC APIs	58
11. References	60
11.1. Normative References	60
11.2. Informative References	62
Appendix A. OpenAPI definition	62
Appendix B. Protocol mapping	93
B.1. Protocol mapping OpenAPI model	93

B.2. Protocol map for BLE	95
B.3. Protocol map for Zigbee	99
Appendix C. NIPC API extensions	102
C.1. NIPC API write binary blob extension	102
C.2. NIPC API bulk operations extension	104
C.3. NIPC API write file extension	112
C.4. NIPC API firmware update extension	114
C.5. NIPC API conditional read extension	119
C.6. NIPC API property extensions	125
Appendix D. Example SDF model with protocol mappings for BLE . .	129
Authors' Addresses	133

1. Introduction

1.1. Scope

Low-power sensors, actuators and other connected devices introduced in environments and use cases such as building management, healthcare, workplaces, manufacturing, logistics and hospitality are often battery-powered. With limited power budget, they may not be able to support the IP protocol on their wired or wireless interfaces, hence they support protocols that require a lower power budget. Prominent examples of such protocols are [BLE53] and [Zigbee22]. These devices typically do require to communicate with devices or applications that are connected to IP-based networking infrastructure. Therefore, applications on the IP network that need to communicate or receive telemetry from these non-IP low-power devices must do so through a gateway function on the IP network. This gateway functions then translates the communication to the non-IP protocol that the low-power device supports.

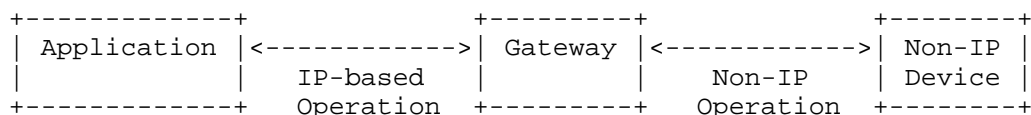


Figure 1: Gateway for non-IP Devices

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 non-IP devices, bespoke and vendor-specific implementations have proliferated. This results in parallel infrastructure of both gateways and non-IP networks being deployed on a case by case basis, each connecting separately into the IP network, with a distinct set of APIs. At the same time, wireless access points supporting both

IP-based wireless connectivity as well as non-IP based wireless technologies are deployed ubiquitiously. Many of these wireless access points are equipped with radios that can transmit and receive different frame types, such as [BLE53] and [Zigbee22]. This specification aims to define a Gateway API for these Non-IP protocols that can be leveraged by this wireless infrastructure in order to connect Non-IP devices into IP networks. The specification aims to be extensible, in order to support existing and future non-IP protocols.

A standardized Non-IP Gateway interface has following benefits:

- * Avoid the need for parallel Non-IP infrastructure.
- * Avoid the need for applications to perform bespoke integrations for different environments.
- * Faster and more cost-effective adoption of Non-IP devices in IP network environments.

1.2. Non-IP Gateway

A non-IP gateway MUST provide at least following functions:

- * Authentication and authorization of application clients that will leverage the gateway API to communicate with Non-IP devices.
- * Access to a database of onboarded devices. Onboarding ensures that the Non-IP Gateway can identify a specific device and has sufficient context about the device to service gateway API requests.
- * The ability to consume an interaction model for a class of devices. This allows the gateway to understand how to interact with a device.
- * An API that allows for bi-directional communication to non-IP devices.
- * One or more channels to process requests, responses, and asymmetric communications with the non-IP radio resources (Access Points) at its disposal.
- * The ability to stream telemetry received from non-IP devices in real-time to applications on the IP network.

The definition of the onboarding function is out of scope of this document, but can be provided by a provisioning interface such as [RFC7644] leveraging [I-D.ietf-scim-device-model]. NIPC performs operations on a device or group object, hence it requires device onboarding to be performed prior to performing a NIPC operation on a device. NIPC APIs will reference a device or group id generated at the time of onboarding as a unique identifier.

The Application gateway is a network function, so its goal is to proxy payloads between Non-IP and IP networks. It is not intended to be a middleware function that interprets, decodes or modifies these payloads.

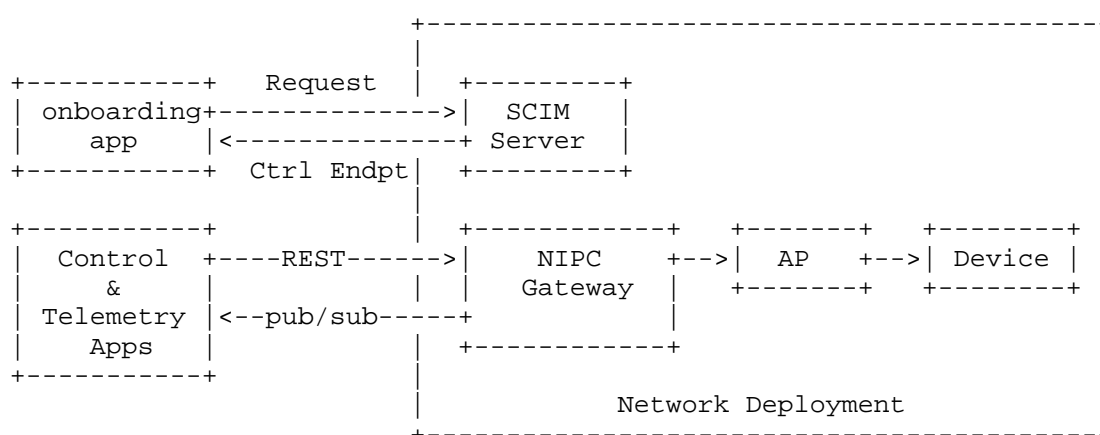


Figure 2: Basic Architecture

Figure 2 shows us applications, the NIPC application layer gateway (ALG), an access point (AP), and a device (D). The applications, application layer gateway and access point are deployed on an IP-Network. The AP supports a Non-IP interface, which it uses to communicate with the device. The Application is deployed in a different administrative domain than the network elements (ALG & AP). The role of the application layer gateway is to provide a gateway function to applications wishing to communicate with non-IP devices in the network domain served by the gateway. Applications implementing Non-IP Control can leverage RESTful interfaces to communicate with Non-IP devices in the network domain and subscribe to events leveraging a CBOR-based pub/sub interface.

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.

2. Architecture

2.1. Overview

Non-IP protocols, such as BLE or Zigbee, typically define a number of basic operations that are similar across protocols. Examples of this are read and write data. NIPC provides a unified API to support those operations.

To perform NIPC operations on a device, the gateways needs 2 things: 1) Information about the instance of the device or thing: The device must be onboarded on the gateway (e.g. by means of SCIM). This allows the NIPC Gateway to retrieve the device object, identified by an id referenced in the path of the NIPC API. 2) Information about the interaction model: The Gateway must be able to perform protocol-neutral operations, and hence requires a mapping of protocol-neutral operations to protocol specific operations. These are supplied to the gateway by means of an SDF model, described in [I-D.ietf-asdf-sdf].

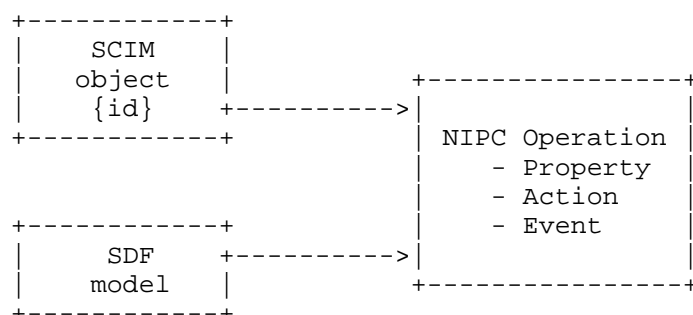


Figure 3: NIPC prerequisites

Once these 2 prerequisites have been fulfilled, applications that are authorized can perform NIPC operations on device ids. NIPC operations are operations on properties, or constitute actions or events on devices, as per the affordances defined in an SDF model.

2.2. Onboarding

In order to perform a NIPC operation on a device, the device has to have its identity declared to the NIPC gateway. We refer to this as 'onboarding'. Apart from the device identity, it is also necessary that the device object contains all required information to bootstrap trust with the device, as well as establish connectivity, as NIPC operations assume that connectivity is there. Although onboarding could theoretically be performed in other ways, it is strongly recommended to leverage [RFC7644] with [I-D.ietf-scim-device-model], as the SCIM device schema has been developed to contain all necessary attributes and extensions to support NIPC.

2.3. Registrations

NIPC registrations APIs allow applications to declare information that is not related to a device instance. Registrations can be information about an interaction model for a class of devices, or information about an application that is required to interact with the gateway.

2.3.1. SDF model registrations

The SDF model for a class of devices determines how a gateway may interact with these devices in a protocol-neutral way. In order to do that, the SDF model must contain protocol mappings, mapping protocol-neutral SDF affordances to protocol-specific ones. The SDF affordances supported by the device, as well as its protocol-mappings, are provided to the gateway by means of an SDF model registration. SDF models are described in [I-D.ietf-asdf-sdf].

2.3.2. Data application registrations

An application authorized to perform NIPC operations on devices needs to be able to define which applications can receive streaming event data from the gateway. The data-app registrations API allows mapping of an event to a data app as well as a protocol.

2.4. Protocol mapping

An SDF model describes a thing, including all its SDF affordances. To leverage an SDF model to perform protocol specific operations on an instance of that thing, a mapping of the SDF affordance to a protocol-specific attribute is required. In NIPC, this is achieved through protocol mapping.

Protocol mapping is required to map a protocol agnostic affordance to a protocol-specific operation, as implementations of the same affordance will differ between protocols. For example BLE will address a property as a service characteristic, while a property in Zigbee is addressed as a property in a cluster of an endpoint.

protocol mapping allows for integration of new protocols in NIPC, new mappings can be created without the need to update the base schema.

a protocol mapping object is a JSON object identified by the `sdfProtocolMap` keyword.

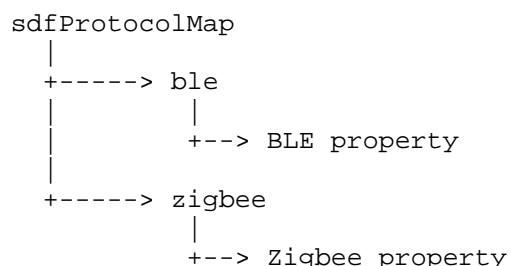


Figure 4: Property Mapping

As shown in Figure 4, protocol-specific properties must be embedded in an `sdfProtocolMap` object, for example a "ble" or a "zigbee" object.

Attribute	Req	Type	Example
ble	T	object	an object with BLE-specific attributes
zigbee	T	object	an object with Zigbee-specific attributes

Table 1: Protocol objects

where-

- * "ble" is an object containing properties that are specific to the BLE protocol.
- * "zigbee" is an object containing properties that are specific to the Zigbee protocol.

- * Other protocol mapping objects can be added by creating a new protocol object

Example protocol mapping:

```
"sdfObject": {
  "healthsensor": {
    "sdfProperty": {
      "heartrate": {
        "description": "The current measured heart rate",
        "type": "number",
        "unit": "beat/min",
        "observable": false,
        "writable": false,
        "sdfProtocolMap": {
          "ble": {
            "serviceID": "12345678-1234-5678-1234-56789abcdef4",
            "characteristicID":
              "12345678-1234-5678-1234-56789abcdef4"
          }
        }
      }
    }
  }
}
```

Figure 5: Example property mapping

A protocol map is provided as part of the SDF model, in the SDF affordance definition. Alternatively, it can be provided directly in a NIPC API.

2.5. NIPC Operations

NIPC operations are protocol-neutral operations on SDF affordances, more specifically properties, actions & events. NIPC operations can happen against affordances that were registered in an SDF model. If connection management is required to execute a NIPC operation, it is assumed that the gateway implicitly establishes and tears down required connections.

2.5.1. Properties APIs

Property APIs allow applications perform operations on properties, such as to read or write values to them.

2.5.2. Actions

Action APIs perform actions on devices, such as enabling or disabling a feature on a device.

2.5.3. Events

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

2.5.4. Connection management for NIPC Operations

For protocols that require connection management before executing an operation, a NIPC gateway will perform implicit connection management. When executing a NIP operation, a NIPC Gateway can set up a connection with a device as well as tear down the connection after the operation has completed. A NIPC Gateway may support explicit connection management as well. Explicit connection management can be used by an app that wants to perform multiple NIPC operations in a single connection. Explicit connection management can be performed by calling the /devices/{id}/connections API. When after establishing an explicit connection to a device, an application calls a NIPC Operation, the Gateway will leverage the existing connection and will also not tear the connection down after the operation completes. The app will have to explicitly close the connection.

2.5.5. Extensions

NIPC is extendable as a NIPC gateway may want to provide a way to execute a complex set of NIPC operations in a single API call, or may want to perform an operation that can deliver an outcome more efficiently than a NIPC API. An example of an extension that uses compound operation is the bulk API.

Extensions must leverage the /extension path element. In order to assure inter-operability, extensions should be IANA registered Section 10.3.

2.6. Events publish subscribe interface

Events are published on a publish/subscribe interface. Events can be of different types:

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

- * Broadcasts from devices
- * Connection events: Devices connecting & disconnecting

2.7. Protocols

2.7.1. NIPC APIs

NIPC is a protocol that is based on RESTful HTTP [RFC9114]. Along with HTTP headers and URIs, NIPC uses JSON [RFC7159] payloads to convey NIPC operations, such as registrations, actions, event and property operations. This is the case for both request and response parameters, as well as errors. NIPC uses the JSON media type "application/nipc+json" that is registered in Section 10.1, except for the SDF model registrations APIs, where the media type reflects the content as an SDF model, and hence is media type "application/sdf+json". The NIPC property APIs also support the use of other media types to describe the content of the property.

2.7.2. NIPC publish/subscribe events

NIPC publish/subscribe events are encoded in CBOR ([RFC8949]) and can be delivered over either:

- * MQTT
- * Webhook
- * WebSocket

2.8. Paths

2.8.1. 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 10.4 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 can 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.

```
{
  "base_path": "/nipc",
  "versions": [
    "/v1"
  ],
  "extensions": [
    "/extensions/{id}/bulk",
    "/extensions/{id}/firmware",
    "/extensions/{id}/properties/blob",
    "/extensions/{id}/properties/file",
    "/extensions/{id}/properties/read/conditional",
    "/extensions/{id}/properties/write"
  ]
}
```

Figure 6: Example response for `/.well-known/nipc`

2.8.2. NIPC Registrations

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

paths:

- * `/registrations/models`
- * `/registrations/data-apps`

2.8.3. NIPC Operations

Every NIPC Operations API pertains to either a device or group of devices, identified by an id, hence the id will 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:

- * property
- * event
- * action
- * extension

These are described in Section 2.5.

2.9. Schema

The NIPC schema leans heavily on the SDF schema, as defined in [I-D.ietf-asdf-sdf]. NIPC operations map directly to SDF affordances.

2.9.1. SDF model registrations

In order to perform a NIPC operation on a device, an SDF interaction model needs to be declared that provides protocol mappings for the SDF affordances the operations will be performed on.

The SDF model can be registered by means of a registrations API POST with the SDF model in the body of the request. A registered SDF model can be fetched by a registrations API GET with an sdfReference.

2.9.2. NIPC Operations

NIPC operations require 2 key parameters: 1) A device ID identifying the device the operation should be executed on 2) an SDF reference for the SDF affordance the operations pertain to

2.9.2.1. 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.

Attribute	Req	Type	Example
id	T	uuid	12345678-1234-5678-1234-56789abcdef4

Table 2: Definition of a device our group of devices

Id is the unique uuid of the device. This id is generated when registering the device, for example against a SCIM server. As such, this id is a common identifier, known both to the application as well as the NIPC Server.

2.9.3. SDF Name

NIPC operations happen against SDF affordances and are referenced with an global sdfName, which is the full path including the namespace.

The operations are either Properties, Events or Actions and their references are of type string

For example:

Attribute	Req	Type	Example
Property	T	string	https://example.com/hearttrate#/ sdfObject/thermostat/sdfProperty/ temperature

Table 3: Definition of a NIPC operation on a property

2.9.4. Responses

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

The exception to the above are NIPC extensions, Section 5.2. These contain compound statements, and thus require the gateway to execute multiple NIPC operations. In this case the NIPC gateway will return HTTP status code 202 after receiving the request and verifying it is able to execute it. The client can then perform an HTTP GET of the extension API to get the execution status for the request. If a callback URL was address was defined in the request, the NIPC Gateway can optionally perform a callback with a response to the compound request after the compound statement completes.

Actions also follow an asynchronous pattern, returning HTTP status code 202 when the action is accepted, along with a Location header pointing to the action instance for status tracking.

A failure response will consist of a HTTP status code of 4xx or 5xx, and will follow the [RFC9457] Problem Details format with application/problem+json media type. The response will contain a type field with a URI identifying the error type, and a human-readable detail field. The type field is a URI and is described in Section 6.

Failure response:

Example of a failure response:

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

```
{
  "type": "https://www.iana.org/assignments/nipc-problem-types#invalid-id",
  "status": 400,
  "title": "Invalid Device ID",
  "detail": "Device ID 12345678-1234-5678-1234-56789abcdef4 does not\
\ exist or is not a device"
}
```

Figure 7: Example failure response

where-

- * "type" is a URI identifying the specific error type
- * "status" is the HTTP status code
- * "title" is a brief, human-readable summary of the error type
- * "detail" is a human-readable explanation specific to this occurrence

3. NIPC Registrations

NIPC allows an application to register an SDF model for a class of devices, as well as a data application that will receive streaming data from the gateway.

3.1. SDF model registrations APIs

These 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 [I-D.ietf-asdf-sdf].

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:

- * an SDF document in JSON format containing one or more sdfThings or sdfObjects, similar to the example in Appendix D.

Response:

Example of a response:

```
[
  {
    "sdfName": "https://example.com/heartrate#/sdfObject/healthsensor"
  }
]
```

Figure 8: Example register SDF model response

where-

- * "sdfName" is the reference to the top-level sdfThing or sdfObject in the SDF model

3.1.2. Get all SDF models

Method: GET /registrations/models

Description: Gets all SDF models registered with the gateway

Response:

Example of a response:

```
[
  {
    "sdfName": "https://example.com/hearttrate#/sdfObject/healthsensor"
  }
]
```

Figure 9: Example get all SDF models response

where-

- * "sdfName" is the reference to the top-level sdfThing or sdfObject in the SDF model

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 reference to 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 D.

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 reference to the top-level sdfThing or sdfObject in the SDF model.

Response:

Example of a response:

```
{
  "sdfName": "https://example.com/heartrate#/sdfObject/healthsensor"
}
```

Figure 10: Example delete SDF model response

where-

- * "sdfName" is the reference to the top-level sdfThing or sdfObject in the SDF model

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 reference to the top-level sdfThing or sdfObject in the SDF model.

Request Body:

- * an SDF model in JSON format, similar to the example in Appendix D.

Response:

Example of a response:

```
{
  "sdfName": "https://example.com/heartrate#/sdfObject/healthsensor"
}
```

Figure 11: Example update SDF model response

where-

- * "sdfName" is the reference to the top-level sdfThing or sdfObject in the SDF model

3.2. Data application registrations APIs

These 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. The endpoint app is defined in Section 6 of [I-D.ietf-scim-device-model].

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 wants to receive streaming data for.

The request body also contains one of the following:

- * mqttClient: a boolean that denotes that the data application is an MQTT client that will receive streaming data over MQTT
- * mqttBroker: an object that contains the MQTT broker information where the broker will publish the streaming data.
 - 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 CA certificate of the MQTT broker (optional)
 - customTopic: the custom topic to publish the streaming data to (optional)
- * webhook: an object that contains a webhook URL along with any credentials that are required to authenticate the webhook. The webhook URL is the endpoint where the streaming data will be sent.
 - URI: the webhook URL
 - 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.
 - serverCACert: the CA certificate of the webhook server (optional)

- * websocket: an object that 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 URL
 - 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.
 - serverCACert: the CA certificate of the websocket server (optional)

Example of a request body:

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

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

Figure 12: Example with mqttClient

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

```
===== 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"  
  }  
}
```

Figure 13: 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 with the gateway

Query Parameters:

* dataAppId: the id of the data application

Request Body: Same as the request body for the register data application API.

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 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.

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:

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

4. NIPC APIs

The primary goal of the NIPC APIs is to perform operations on SDF Affordances, such as properties, events & actions. This allows a user of the NIPC API to get or update properties of devices, perform actions on devices, and consume events from devices.

The NIPC APIs consists of 3 main collections which reflect SDF Affordances as defined in Section 1.2 of [I-D.ietf-asdf-sdf]:

- * NIPC Property APIs: These APIs allow applications to get and update device properties.
- * NIPC Event APIs: These APIs allow applications to enable or disable event reporting on devices.
- * NIPC Action APIs: These APIs allow applications to perform actions on devices.

One or more SDF models must be registered in order to use these NIPC Property, Event and Action APIs. The SDF models can be a top-level sdfThing with multiple sdfObjects or a top-level sdfObject. These APIs depend on the SDF affordances (i.e. sdfProperty, sdfEvent and sdfAction) defined in the SDF model and a device ID that is defined in [I-D.ietf-scim-device-model]. An SDF affordance can be referenced using the global name of the SDF affordance as described in Section 4 of [I-D.ietf-asdf-sdf].

The SDF global name will be used against the registered SDF model to determine the protocol-specific protocolmap that the NIPC API will operate on.

4.1. NIPC Property APIs

These APIs allow applications to get and update device properties. These operations may require a connection to the device to be established. This connection can be established as part of the same API call implicitly. If a connection is already active for this device, the existing connection will be leveraged without modifying it.

These APIs support multiple media types based on Content-Type and Accept headers to accommodate different data formats.

When using application/nipc+json, the request and response bodies follow the format shown in the examples above, with binary data encoded as base64 in the "value" field. For other content types, the data is transmitted according to the specific format requirements of that media type.

4.1.1. Write 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`: a single property to update. If this parameter is provided, the request body can contain any content type payload with the value to write to the property. If this parameter is not provided, the request body **MUST** contain an `application/nipc+json` payload with an array of properties to update, each containing a property and a value.

Request Body:

- * If the query parameter `propertyName` is provided, the request body can contain any content type payload with the value to write to the property. The value is the raw binary data to write to the property.

or

- * an array of properties to update, each containing a property and a value. The value is the raw binary data, encoded in base64 with padding as per Section 5 of [RFC4648].

Example body updating multiple properties:

===== 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=="
  }
]
```

Figure 14: Example updating multiple properties

Response:

If the `Accept` header is set to `application/nipc+json`, the response will 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.

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

Example of a response:

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

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

Figure 15: Example update multiple properties response

where-

- * "properties" is an array of properties that were updated, each containing a property and a value

A failure will generate a standard failed response. Please refer to Figure 7 definition of failed response.

4.1.2. Read 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 will be an array of properties, each containing a property and a value. The value is the raw binary data read from the property, encoded in base64 with padding as per Section 5 of [RFC4648].

Example of a response:

===== 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=="
  }
]
```

Figure 16: Example read multiple properties response

where-

- * "property" is the property that was read from
- * "value" is the bytes that were read in base64 encoding

A failure will generate a standard failed response. Please refer to Figure 7 definition of failed response.

4.2. NIPC Event APIs

These APIs allow applications to enable or disable event reporting on devices. These operations may require a connection to the device to be established. This connection can be established as part of the same API call implicitly. If a connection is already active for this device, the existing connection will be leveraged without modifying it.

The event is a global name reference to an sdfEvent.

The ID in the path is the id of the device or group of devices. An event can be enabled on a group of devices if it is supported by the underlying protocol. For example, if the underlying protocol is BLE, the event can be enabled on a group of devices if the event is an advertisement event or connection status event.

If the data application registered for this event is an MQTT broker or client, the event is 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 "12345678-1234-5678-1234-56789abcdef4" 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/12345678-1234-5678-1234-56789abcdef4/thermometer/sdfThing/\
thermometer/sdfEvent/isPresent

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

If a custom topic is provided for an MQTT broker, the custom topic is used as the MQTT topic instead.

4.2.1. Enable event reporting

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

Description: Enables an event on a specific device

Parameters:

- * id: the id of the device

Query Parameters:

- * eventName: the event to enable

The eventName is a URL encoded string that is the absolute URI that is a global name reference to 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/12345678-1234-5678-1234-56789abcdef4/events?instanceId=87654321-4321-8765-4321-fedcba9876543
```

The Location header contains the URI for the created event instance, which can be used to check status or disable the event.

A failure will generate a standard failed response. Please refer to Figure 7 definition of failed response.

4.2.2. Disable event reporting

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

Description: Disables an event on a specific 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
```

A failure will generate a standard failed response. Please refer to Figure 7 definition of failed response.

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:

Example of a response:

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

```
[
  {
    "instanceId": "87654321-4321-8765-4321-fedcba9876543",
    "event": "https://example.com/heartrate#/sdfObject/healthsensor/\
sdfEvent/fallDetected"
  }
]
```

Figure 17: Example get multiple events status response

where-

- * "instanceId" is the unique instance ID for each enabled event
- * "event" is the event URI for each enabled event

A failure will generate a standard failed response. Please refer to Figure 7 definition of failed 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 a global name reference to 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: /groups/12345678-1234-5678-1234-56789abcdef4/events?instanceId=87654321-4321-8765-4321-fedcba9876543
```

The Location header contains the URI for the created event instance, which can be used to check status or disable the event.

A failure will generate a standard failed response. Please refer to Figure 7 definition of failed response.

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:

Returns HTTP status code 204 No Content on successful disable.

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

A failure will generate a standard failed response. Please refer to Figure 7 definition of failed response.

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 on a group of devices

Parameters:

- * id: the id of the group of devices

Query Parameters:

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

Response:

Example of a response:

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

```
[
  {
    "instanceId": "87654321-4321-8765-4321-fedcba9876543",
    "event": "https://example.com/heartrate#/sdfObject/healthsensor/\
sdfEvent/fallDetected"
  }
]
```

Figure 18: Example get multiple group events status response

where-

- * "instanceId" is the unique instance ID for each enabled event
- * "event" is the event URI for each enabled event

A failure will generate a standard failed response. Please refer to Figure 7 definition of failed response.

4.3. NIPC Action APIs

These APIs allow applications to perform actions on devices. These operations may require a connection to the device to be established. This connection can be established as part of the same API call implicitly. If a connection is already active for this device, the existing connection will be leveraged without modifying it.

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.

Example of a successful response:

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

HTTP/1.1 202 Accepted

Location: /devices/12345678-1234-5678-1234-56789abcdef4/actions?instanceId=87654321-4321-8765-4321-fedcba9876543

The Location header contains the URI for the action instance, which can be used to check the action status.

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:

Example of a response:

```
{
  "status": "COMPLETED"
}
```

Figure 19: Example action status response

where "status" indicates the current state of the action (e.g., "IN_PROGRESS" or "COMPLETED").

A failure will generate a standard failed response. Please refer to Figure 7 definition of failed response.

4.4. NIPC explicit connections management APIs

A lot of protocols do not require connection management, so for these protocols, these APIs will not apply. NIPC Gateways perform implicit connection management for protocols that require connection management (such as BLE), so in principle the user of the NIPC API does not have to perform connection management. In some cases however, a user might want to keep a connection open, perform an action, evaluate the result and perform a second action based on that result. In this case it is useful to perform explicit connection management so the connections remains established to execute subsequent actions. These APIs allow applications to explicitly manage device connections. The examples in this section will be based on BLE, which requires connection management.

4.4.1. 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 map object. In the case of BLE, if no protocol map 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 discover can be buffered across connections, so the API also supports caching parameters.

Example body of a connection without specific discovery of properties:

```
{  
  "retries": 3,  
  "retryMultipleAPs": true  
}
```

Figure 20: Example connection

where-

- * "retries" defines the number of retries in case the operations does not succeed
- * "retryMultipleAPs" can be used in case there is an infrastructure with multiple access points or radios that can reach the device. If set to "true" a different access point may be used for retries.

In case the application would like to discover specific properties of a device, a protocol mapping can be added that defines what properties should be discovered.

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

```
{
  "retries": 3,
  "retryMultipleAPs": true,
  "sdfProtocolMap": {
    "ble": {
      "services": [
        {
          "serviceID": "12345678-1234-5678-1234-56789abcdef4"
        }
      ],
      "cached": false,
      "cacheIdlePurge": 3600,
      "autoUpdate": true,
      "bonding": "default"
    }
  }
}
```

Figure 21: 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 cached for subsequent connects, in order not to perform service discovery on each request.
- * "cacheIdlePurge" 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 when onboarding the device

Response:

Success responses include a protocol mapping 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 22.

```

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

```

Figure 22: BLE Services

Example of a response:

```

{
  "id": "12345678-1234-5678-1234-56789abcdef4",
  "sdfProtocolMap": {
    "ble": [
      {
        "serviceID": "12345678-1234-5678-1234-56789abcdef4",
        "characteristics": [
          {
            "characteristicID":
              "12345678-1234-5678-1234-56789abcdef4",
            "flags": [
              "read",
              "write"
            ],
            "descriptors": [
              {
                "descriptorID":
                  "12345678-1234-5678-1234-56789abcdef4"
              }
            ]
          }
        ]
      }
    ]
  }
}

```

Figure 23: Example connection response

where-

* "id" is the id of the device

- * "sdfProtocolMap" contains an Array of BLE services as shown in Figure 22

A failure will generate a standard failed response. Please refer to Figure 7 definition of failed response.

4.4.2. Update a connection

Method: PUT /devices/{id}/connections

Description: Update 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 map object. In the case of BLE, if no protocol map 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 discover can be buffered across connections, so the API also supports caching parameters.

Example body of an update connection:

```
{
  "sdfProtocolMap": {
    "ble": {
      "services": [
        {
          "serviceID": "12345678-1234-5678-1234-56789abcdef4"
        }
      ],
      "cached": false,
      "cacheIdlePurge": 3600,
      "autoUpdate": true
    }
  }
}
```

Figure 24: 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
- * "cacheIdlepurge" 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 mapping 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 22.

Example of a response:

```

{
  "id": "12345678-1234-5678-1234-56789abcdef4",
  "sdfProtocolMap": {
    "ble": [
      {
        "serviceID": "12345678-1234-5678-1234-56789abcdef4",
        "characteristics": [
          {
            "characteristicID":
              "12345678-1234-5678-1234-56789abcdef4",
            "flags": [
              "read",
              "write"
            ],
            "descriptors": [
              {
                "descriptorID":
                  "12345678-1234-5678-1234-56789abcdef4"
              }
            ]
          }
        ]
      }
    ]
  }
}

```

Figure 25: Example connection response

where-

- * "id" is the id of the device
- * "sdfProtocolMap" contains an Array of BLE services as shown in Figure 22

A failure will generate a standard failed response. Please refer to Figure 7 definition of failed response.

4.4.3. 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:

```
{
  "id": "12345678-1234-5678-1234-56789abcdef4"
}
```

Figure 26: Example disconnect response

where "id" is the id of the device.

A failure will generate a standard failed response. Please refer to Figure 7 definition of failed response.

4.4.4. 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:**

```

{
  "id": "12345678-1234-5678-1234-56789abcdef4",
  "sdfProtocolMap": {
    "ble": [
      {
        "serviceID": "12345678-1234-5678-1234-56789abcdef4",
        "characteristics": [
          {
            "characteristicID":
              "12345678-1234-5678-1234-56789abcdef4",
            "flags": [
              "read",
              "write"
            ],
            "descriptors": [
              {
                "descriptorID":
                  "12345678-1234-5678-1234-56789abcdef4"
              }
            ]
          }
        ]
      }
    ]
  }
}

```

Figure 27: Example connection status response

where-

- * "id" is the id of the device
- * "sdfProtocolMap" contains an Array of BLE services as shown in Figure 22

A failure will generate a standard failed response. Please refer to Figure 7 definition of failed response.

5. NIPC Extensibility

NIPC is extensible in two ways:

- * Protocol mapping: New protocol mapping can extend NIPC with support for new non-IP protocols

- * API extensions: API extensions leverage compound statements of basic NIPC action APIs to simplify common operations for applications.

5.1. Protocol extensions

As described in Section 2.4, NIPC supports mapping protocol specific properties to NIPC properties. BLE and Zigbee are used as examples, but protocol mapping is extensible to other protocols, so now non-IP protocols can be supported by NIPC without a schema change.

The protocol objects need to be extended with the new protocol as well. Protocol objects will be extended as follows:

Attribute	Req	Type	Example
ble	T	object	an object with BLE-specific properties
zigbee	T	object	an object with Zigbee-specific properties
newProtocol	T	object	an object with newProtocol-specific props

Table 4: Adding Protocol mappings

In the new protocol object, protocol specific properties can be added.

Protocol mappings need to be IANA registered, see Section 10.2.

5.2. API extensions

/extension

The extension APIs allow for extensibility of the APIs, either IANA registered extensions (Section 10.3) or vendor-specific extensions. Extension APIs must leverage the basic NIPC defined APIs and combine them in compound statements in order to streamline application operation against devices, make operations more expediant and convenient in one API call. In principle they do not add any basic functionality. In the OpenAPI model Appendix C below, we have defined a few example extensions.

The extensions can contain long running operations, such as firmware updates, or other bulk operations that can be performed on a device. For long running operations, the extension API will return a 202 Accepted status code and a location header with the URL to check the status of the operation. The status of the operation can be checked by calling the status extension API with the same device ID. The status extension API will return a 200 OK status code when the operation is in progress. When the operation is complete, the status extension API will return a 303 See Other status code with a location header with the URL to check the status of the operation. The GET operation on the extension API will return a 200 OK status code with the actual response once the operation is complete.

6. NIPC Error Handling

The error types in the NIPC APIs use URI-based error type identifiers as defined in Section 10.5. 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.

The specific 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
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-characteristic	Invalid BLE service or characteristic ID	Protocol specific
protocolmap-zigbee-connection-timeout	Zigbee connection timeout	Protocol specific
protocolmap-zigbee-invalid-endpoint-or-cluster	Invalid Zigbee endpoint or cluster ID	Protocol specific

extension-broadcast- invalid-data	Invalid transmit data	Transmit APIs
extension-firmware- rollback	Firmware rollback	Extension APIs
extension-firmware- update-failed	Firmware update failed	Extension APIs

Table 5: Error Codes

The appropriate HTTP status code is returned in the response.

7. Publish/Subscribe Interface

The publish/subscribe interface, or data streaming interface, is an MQTT publishing interface. Pub/sub topics can be created and managed by means of the `/registration/data-app` API.

In this memo, we propose the data format to be CBOR [RFC8949].

7.1. CDDL Definition

We have a CDDL [RFC8610] definition where we define the `DataSubscription` struct that will be used by all the messages published to the MQTT broker.

The `DataSubscription` struct is a CBOR map that will contain the raw data in bytes and a timestamp of the data. Optionally, the message will also have a `deviceId` that corresponds to the SCIM ID of the device if the payload is associated to a known device.

Other fields in the CDDL such as `apMacAddress` and `rsssi` can be optionally included but these fields can expose the underlying network topology.

Each message also has a subscription choice group that will define the type of data that is being published.

Each MQTT message can be a collection of `DataSubscription` structs. This collection is represented as `DataBatch` in the CDDL.

```
DataBatch = [* DataSubscription]

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

subscription = (
  bleSubscription: BleSubscription //
  bleAdvertisement: BleAdvertisement //
  bleConnectionStatus: BleConnectionStatus //
  zigbeeSubscription: ZigbeeSubscription //
  rawPayload: RawPayload
)

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
}

RawPayload = {
  contextID: text
}
```

7.2. CBOR Examples

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

```
[
  {
    "data": h'02011A020A0C16FF4C001007721F41B0392078',
    "deviceID": "75fde96d-886f-4ac0-ald5-df79f76e7c9c",
    "timestamp": 1727484393,
    "bleAdvertisement": {
      "macAddress": "C1:5C:00:00:00:01",
      "rssi": -25
    }
  }
]
```

Figure 28: Onboarded BLE Device Advertisement

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

Figure 29: Non-onboarded BLE Device Advertisement

```
[
  {
    "data": h'434630374346303739453036',
    "deviceID": "75fde96d-886f-4ac0-ald5-df79f76e7c9c",
    "timestamp": 1727484393,
    "bleSubscription": {
      "serviceID": "a4e649f4-4be5-11e5-885d-feff819cdc9f",
      "characteristicID": "c4c1f6e2-4be5-11e5-885d-feff819cdc9f"
    }
  }
]
```

Figure 30: BLE GATT Notification


```
[
  {
    "deviceID": "75fde96d-886f-4ac0-ald5-df79f76e7c9c",
    "timestamp": 1727484393,
    "bleConnectionStatus": {
      "macAddress": "C1:5C:00:00:00:01",
      "connected": true
    }
  }
]
```

Figure 31: BLE Connection status event

```
[
  {
    "data": h'434630374346303739453036',
    "deviceID": "75fde96d-886f-4ac0-ald5-df79f76e7c9c",
    "timestamp": 1727484393,
    "zigbeeSubscription": {
      "endpointID": 1,
      "clusterID": 6,
      "attributeID": 12,
      "type": 1
    }
  }
]
```

Figure 32: 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:

- * Onboard a device using the SCIM Interface (out of scope of this memo)
- * Register an SDF model for the device

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

```
POST /registration/model
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 D

- * Read a property from the BLE device

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

```
GET /devices/12345678-1234-5678-1234-56789abcdef4/properties?prop\
ertyName=https://example.com/thermometer%23/sdfThing/thermometer/\
sdfProperty/device_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/12345678-1234-5678-1234-56789abcdef4/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

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

The sequence of operations for this are:

- * Onboard a device 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 /registration/data-app?dataAppId=23456789-1234-5678-1234-567\89abcdef4

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
}
```

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

- * Register the data app with the event

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

POST /registration/data-app?dataAppId=23456789-1234-5678-1234-567\89abcdef4

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/12345678-1234-5678-1234-56789abcdef4/events?eventNa\me=https://example.com/thermometer%23/sdfThing/thermometer/sdfEve\nt/isPresent

Host: localhost

Content-Length: 0

HTTP/1.1 201 Created

Location: /devices/12345678-1234-5678-1234-56789abcdef4/events?in\stanceId=87654321-4321-8765-4321-fedcba9876543

- * Check the status of the event

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

```
GET /devices/12345678-1234-5678-1234-56789abcdef4/events?instance\
Id=87654321-4321-8765-4321-fedcba9876543
```

```
Host: localhost
```

```
HTTP/1.1 200 OK
```

```
content-type: application/nipc+json
```

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

9. Security Considerations

9.1. API authorization

In order to enable a network wishing to offer NIPC ALG functions, the network administrator authorizes application(s) to perform operations on the Gateway. This happens out of band and may be accomplished by means of exchanging tokens or public keys. Authorization can be role-based. The 3 primary roles are:

1. Onboarding: Authorize an onboarding application against a SCIM server co-located with the gateway.
2. Control: Authorize applications that may control devices.
3. Data: Authorize applications that may receive telemetry.
It is possible to further refine roles down to an API basis.

10. IANA Considerations

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

10.1. 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 9 section of this document.

Interoperability considerations: none

Published specification: This document, the NIPC API specification.

Applications that use this media type:

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

10.2. Protocol mapping

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

The registry must contain following attributes:

- * Protocol map name

- * Protocol name

- * Description
- * Reference of the specification describing the protocol mapping. This specification must be reviewed by an expert.

Following protocol mappings are described in this document:

Protocol map	Protocol Name	Description	Reference
ble	Bluetooth Low Energy (BLE)	Protocol mapping for BLE devices	This document
zigbee	Zigbee	Protocol mapping for Zigbee devices	This document

Table 6

10.3. API extensions

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

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
/extension/{id}/bulk	Bulk API	Call multiple NIPC's in a single request	Appendix C
/extension/{id}/properties/file	File write API	Write a file with multiple property ops	Appendix C
/extension/{id}/properties/blob	Binary write API	Write a binary blob with multiple property ops	Appendix C
/extensions/{id}/properties/read/conditional	Read conditional API	Read a property until a condition is fulfilled	Appendix C
/extensions/{id}/firmware	Firmware upgrade API	Firmware upgrade API, leveraging mutiple opsed	Appendix C
/extensions/{id}/manage/transmit	Transmit API	Transmits a payload to a device	Appendix C

Table 7

10.4. 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 2.8.1

Table 8

The well-known URI for NIPC is:

`/.well-known/nipc`

10.5. 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>.

Registrations MUST use the prefix "<https://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/nipc-problem-types#sdf-model	SDF model already	This document

already-registered	registered	
+-----+-----+-----+		
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#property-not-readable	Property not readable	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#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	BLE service	This

nipc-problem-types#protocolmap-ble-service-discovery-failed	discovery failed	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-broadcast-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 9

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

11. References

11.1. Normative References

[I-D.ietf-asdf-sdf]

Koster, M., Bormann, C., and A. Keränen, "Semantic Definition Format (SDF) for Data and Interactions of Things", Work in Progress, Internet-Draft, draft-ietf-asdf-sdf-24, 27 July 2025, <<https://datatracker.ietf.org/doc/html/draft-ietf-asdf-sdf-24>>.

- [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-17, 25 July 2025, <<https://datatracker.ietf.org/doc/html/draft-ietf-scim-device-model-17>>.
- [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>>.
- [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>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<https://www.rfc-editor.org/info/rfc7159>>.
- [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>>.

- [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>>.
- [RFC9114] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <<https://www.rfc-editor.org/info/rfc9114>>.
- [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>>.

11.2. 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/>>.
- [Zigbee22] Connectivity Standards Alliance, "zigbee Specification, Version 22 1.0", 2017.

Appendix A. OpenAPI definition

The following non-normative model is provide 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: 0.9.0
externalDocs:
  description: NIPC IETF draft
  url: https://datatracker.ietf.org/doc/draft-ietf-asdf-nipc/
servers:
  - url: "{gw_host}/nipc/draft-11"
    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.
  - 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: 12345678-1234-5678-1234-56789abcdef4
- 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/PropertyValueResponse'\
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: 12345678-1234-5678-1234-56789abcdef4
    - 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: device id or group id
              required: true
            schema:
                type: string
                format: uuid

```

```

        example: 12345678-1234-5678-1234-56789abcdef4
    - name: eventName
      in: query
      description: event that needs to be enabled
      required: true
      allowReserved: true
      schema:
        type: string
        example: "https://example.com/hearttrate#/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: device id or group id
      required: true
      schema:
        type: string
        format: uuid

```

```
        example: 12345678-1234-5678-1234-56789abcdef4
- name: instanceId
  in: query
  description: instance id of the event that needs to be disab\
\led
  required: true
  schema:
    type: string
    format: uuid
    example: 12345678-1234-5678-1234-56789abcdef4
  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: 12345678-1234-5678-1234-56789abcdef4
- 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: 12345678-1234-5678-1234-56789abcdef4
```

```
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: 12345678-1234-5678-1234-56789abcdef4
      - name: eventName
        in: query
        description: event that needs to be enabled
        required: true
        allowReserved: true
        schema:
          type: string
          example: "https://example.com/hearttrate#/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: 12345678-1234-5678-1234-56789abcdef4
      - name: instanceId
        in: query
        description: instance id of the event that needs to be disabled
    \led
      required: true
      schema:
        type: string
        format: uuid
        example: 12345678-1234-5678-1234-56789abcdef4
    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 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: 12345678-1234-5678-1234-56789abcdef4
      - name: instanceId
        in: query
        description: |-
          Instance IDs of the events that need to be filtered
        required: false
        schema:
          type: array
          items:
            type: string
            format: uuid
            example: 12345678-1234-5678-1234-56789abcdef4
    responses:
      '200':
        description: Success, events retrieved
        content:
          application/nipc+json:
            schema:
              $ref: '#/components/schemas/EventStatusResponseArray'
    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: 12345678-1234-5678-1234-56789abcdef4
      - 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: 12345678-1234-5678-1234-56789abcdef4
      - name: instanceId
        in: query
        description: |-
          instance id of the action that needs to be checked
        required: true
        schema:
          type: string
          format: uuid
          example: 12345678-1234-5678-1234-56789abcdef4
    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'
```

```

/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: 12345678-1234-5678-1234-56789abcdef4
    requestBody:
      content:
        application/nipc+json:
          schema:
            anyOf:
              - $ref: '#/components/schemas/Connection'
              - $ref: './protocolmaps/ProtocolMap.yaml#/components\
\schemas/ProtocolMap-ServiceList'
            required: false
    responses:
      '200':
        description: Success
        content:
          application/nipc+json:
            schema:
              allOf:
                - $ref: '#/components/schemas/Id'
                - $ref: './protocolmaps/ProtocolMap.yaml#/componen\
\ts/schemas/ProtocolMap-ServiceMap'
      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: 12345678-1234-5678-1234-56789abcdef4
requestBody:
  content:
    application/nipc+json:
      schema:
        $ref: './protocolmaps/ProtocolMap.yaml#/components/sch\
\emas/ProtocolMap-ServiceList'
      required: false
responses:
  '200':
    description: Success
    content:
      application/nipc+json:
        schema:
          allOf:
            - $ref: '#/components/schemas/Id'
            - $ref: './protocolmaps/ProtocolMap.yaml#/componen\
\ts/schemas/ProtocolMap-ServiceMap'
        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: 12345678-1234-5678-1234-56789abcdef4
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: 12345678-1234-5678-1234-56789abcdef4
  responses:
    '200':
      description: Success
      content:
        application/nipc+json:
          schema:
```

```
      allof:
        - $ref: '#/components/schemas/Id'
        - $ref: './protocolmaps/ProtocolMap.yaml#/componen\
\ts/schemas/ProtocolMap-ServiceMap'
    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/hearttrate#/sdfObject/health\
\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/hearttrate#/sdfObject/health\

```

```
\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/hearttrate#/sdfObject/heart\
\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: 12345678-1234-5678-1234-56789abcdef4
    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: 12345678-1234-5678-1234-56789abcdef4
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: 12345678-1234-5678-1234-56789abcdef4
  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: 12345678-1234-5678-1234-56789abcdef4
  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: 12345678-1234-5678-1234-56789abcdef4

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

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

## 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: 12345678-1234-5678-1234-56789abcdef4

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

```
    retries:
      type: integer
      format: int32
      example: 3
    retryMultipleAPs:
      type: boolean
      example: true

## 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/hearttrate#/sdfObject/healths\
\ensor"

SdfModel:
  allof:
    - type: object
      description: Sample SDF model
      properties:
        namespace:
          type: object
          additionalProperties:
            type: string
          example:
            hearttrate: https://example.com/hearttrate
        defaultNamespace:
          type: string
          example: hearttrate
    - 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:
            sdfOutputData:
              sdfProtocolMap:
                ble:
                  type: advertisement
      sdfObject:
        healthsensor:
          sdfProperty:
            hearttrate:
              sdfProtocolMap:
```

```

        ble:
            serviceID: 12345678-1234-5678-1234-56789ab\
\cdef4
            characteristicID: 12345678-1234-5678-1234-\
\56789abcdef4
        sdfEvent:
            fallDetected:
                sdfOutputData:
                    sdfProtocolMap:
                        ble:
                            serviceID: 12345678-1234-5678-1234-56789\
\abcdef4
                            characteristicID: 12345678-1234-5678-123\
\4-56789abcdef4
                        sdfAction:
                            start:
                                sdfProtocolMap:
                                    ble:
                                        serviceID: 12345678-1234-5678-1234-56789ab\
\cdef4
                                        characteristicID: 12345678-1234-5678-1234-\
\56789abcdef4
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: 12345678-1234-5678-1234-56789abcdef4
                            characteristicID: 12345678-1234-5678-1234-5678\
\9abcdef4
            sdfEvent:
                fallDetected:
                    sdfOutputData:
                        sdfProtocolMap:
                            ble:
                                serviceID: 12345678-1234-5678-1234-56789abcd\

```



```

\ef4
                                characteristicID: 12345678-1234-5678-1234-56\
\789abcdef4
    sdfAction:
      start:
        sdfProtocolMap:
          ble:
            serviceID: 12345678-1234-5678-1234-56789abcdef4
            characteristicID: 12345678-1234-5678-1234-5678\
\9abcdef4

SdfProperty:
  type: object
  description: Sample SDF property
  properties:
    sdfProperty:
      additionalProperties:
        allOf:
          - $ref: './protocolmaps/ProtocolMap.yaml#/components/s\
\chemas/ProtocolMap-Property'
      example:
        heartrate:
          sdfProtocolMap:
            ble:
              serviceID: 12345678-1234-5678-1234-56789abcdef4
              characteristicID: 12345678-1234-5678-1234-56789abc\
\def4

SdfEvent:
  type: object
  description: Sample SDF property
  properties:
    sdfEvent:
      additionalProperties: #example, this will be the registere\
\d event
      type: object
      properties:
        sdfOutputData:
          allOf:
            - $ref: './protocolmaps/ProtocolMap.yaml#/componen\
\ts/schemas/ProtocolMap-Event'
      example:
        fallDetected:
          sdfOutputData:
            sdfProtocolMap:
              ble:
                serviceID: 12345678-1234-5678-1234-56789abcdef4
                characteristicID: 12345678-1234-5678-1234-56789a\

```

```
\bcdef4

  SdfAction:
    type: object
    description: Sample SDF property
    properties:
      sdfAction:
        additionalProperties:
          allOf:
            - $ref: './protocolmaps/ProtocolMap.yaml#/components/s\
\chemas/ProtocolMap-Property'
        example:
          start:
            sdfProtocolMap:
              ble:
                serviceID: 12345678-1234-5678-1234-56789abcdef4
                characteristicID: 12345678-1234-5678-1234-56789abc\
\def4

# 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-not-writable
- 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-broadcast-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 12345678-1234-5678-1234-56789abcdef4 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'
<CODE ENDS>

```

Figure 33

Appendix B. Protocol mapping

NIPC requires that a protocol mapping be provided as part of the SDF model for a device or have one provided using the NIPC action APIs with embedded protocol mapping. The protocol mapping is a JSON object that describes the underlying technology used to communicate with the device along with any additional information needed to communicate with the device.

The JSON format of the protocol mapping is provided as a non-normative OpenAPI model for the convenience of the implementor.

B.1. Protocol mapping OpenAPI model

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

openapi: 3.0.3
info:
  title: Non IP Device Control (NIPC) Protocol Mapping
  description: |-
    Non IP Device Control (NIPC) Protocol Mapping. When adding a
    new protocol mapping pls add a reference to the protocol map
    for all the schemas in this file.
  termsOfService: http://swagger.io/terms/
  contact:
    email: bbrinckm@cisco.com
  license:
    name: TBD
    url: TBD
  version: 0.10.0
  url: https://datatracker.ietf.org/doc/draft-ietf-asdf-nipc/

components:
  schemas:
# Protocol Mapping
## Protocol Map for Service Discovery
    ProtocolMap-ServiceList:
      type: object
      properties:
        sdfProtocolMap:
          oneOf:
            - $ref: './ProtocolMap-BLE.yaml#/components/schemas/Prot\
ocolMap-BLE-ServiceList'

## Protocol Map for Service Discovery result
    ProtocolMap-ServiceMap:
      type: object
```

```
    properties:
      sdfProtocolMap:
        oneOf:
          - $ref: './ProtocolMap-BLE.yaml#/components/schemas/Prot\
ocolMap-BLE-ServiceMap'
          - $ref: './ProtocolMap-Zigbee.yaml#/components/schemas/P\
rotocolMap-Zigbee-ServiceMap'

## Protocol Map for Error Codes
ProtocolMap-ErrorCodes:
  type: object
  properties:
    sdfProtocolMap:
      oneOf:
        - $ref: './ProtocolMap-BLE.yaml#/components/schemas/Prot\
ocolMap-BLE-ErrorCodes'
        - $ref: './ProtocolMap-Zigbee.yaml#/components/schemas/P\
rotocolMap-Zigbee-ErrorCodes'

## Protocol Map for Broadcasts
ProtocolMap-Broadcast:
  type: object
  properties:
    sdfProtocolMap:
      oneOf:
        - $ref: './ProtocolMap-BLE.yaml#/components/schemas/Prot\
ocolMap-BLE-Broadcast'
        - $ref: './ProtocolMap-Zigbee.yaml#/components/schemas/P\
rotocolMap-Zigbee-Broadcast'

## Protocol Map for a property
ProtocolMap-Property:
  type: object
  properties:
    sdfProtocolMap:
      oneOf:
        - $ref: './ProtocolMap-BLE.yaml#/components/schemas/Prot\
ocolMap-BLE-Propmap'
        - $ref: './ProtocolMap-Zigbee.yaml#/components/schemas/P\
rotocolMap-Zigbee-Propmap'

## Protocol Map for an event
ProtocolMap-Event:
  type: object
  properties:
    sdfProtocolMap:
      oneOf:
        - $ref: './ProtocolMap-BLE.yaml#/components/schemas/Prot\
```

```
ocolMap-BLE-Event'
  - $ref: './ProtocolMap-Zigbee.yaml#/components/schemas/P\
rotocolMap-Zigbee-Event'
<CODE ENDS>
```

Figure 34

B.2. Protocol map for BLE

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

openapi: 3.0.3
info:
  title: Non IP Device Control (NIPC) BLE Protocol Mapping
  description: |-
    Non IP Device Control (NIPC) BLE Protocol Mapping.
  termsOfService: http://swagger.io/terms/
  contact:
    email: bbrinckm@cisco.com
  license:
    name: TBD
    url: TBD
  version: 0.10.0
externalDocs:
  description: NIPC IETF draft
  url: https://datatracker.ietf.org/doc/draft-ietf-asdf-nipc/

components:
  schemas:
# BLE Protocol Mapping
## A Service is a device with optional service IDs
    ProtocolMap-BLE-ServiceList:
      type: object
      properties:
        ble:
          type: object
          properties:
            services:
              type: array
              items:
                type: object
                properties:
                  serviceID:
                    type: string
                    format: uuid
                    example: 12345678-1234-5678-1234-56789abcdef4
      cached:
```

```
description: |-
  If we can cache information, then device doesn't need
  to be rediscovered before every connected.
type: boolean
default: false
cacheIdlePurge:
  description: cache expiry period, when device 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
```

Protocol Mapping for BLE Service Map

ProtocolMap-BLE-ServiceMap:

```
required:
  - services
type: object
properties:
  ble:
    type: array
    items:
      $ref: '#/components/schemas/ProtocolMap-BLE-Service'
```

ProtocolMap-BLE-Service:

```
required:
  - serviceID
  - characteristics
type: object
properties:
  serviceID:
    type: string
    format: uuid
    example: 12345678-1234-5678-1234-56789abcdef4
  characteristics:
    type: array
    items:
```



```
        $ref: '#/components/schemas/ProtocolMap-BLE-Characteristic\
ic'
```

ProtocolMap-BLE-Characteristic:

```
  required:
    - characteristicID
    - flags
    - descriptors
  type: object
  properties:
    characteristicID:
      type: string
      format: uuid
      example: 12345678-1234-5678-1234-56789abcdef4
    flags:
      type: array
      example:
        - read
        - write
      items:
        type: string
        enum:
          - read
          - write
          - notify
    descriptors:
      type: array
      items:
        $ref: '#/components/schemas/ProtocolMap-BLE-Descriptor'
```

ProtocolMap-BLE-Descriptor:

```
  required:
    - descriptorID
  type: object
  properties:
    descriptorID:
      type: string
      format: uuid
      example: 12345678-1234-5678-1234-56789abcdef4
```

Protocol Mapping for BLE Broadcast

ProtocolMap-BLE-Broadcast:

```
  required:
    - ble
  type: object
  properties:
    ble:
      type: object
```

```
## Protocol Mapping for BLE Property
ProtocolMap-BLE-Propmap:
  required:
    - ble
  type: object
  properties:
    ble:
      required:
        - serviceID
        - characteristicID
      type: object
      properties:
        serviceID:
          type: string
          format: uuid
          example: 12345678-1234-5678-1234-56789abcdef4
        characteristicID:
          type: string
          format: uuid
          example: 12345678-1234-5678-1234-56789abcdef4
```

```
## Defines different types of BLE events
ProtocolMap-BLE-Event:
  required:
    - ble
  type: object
  properties:
    ble:
      required:
        - type
      type: object
      properties:
        type:
          type: string
          example: gatt
          enum:
            - gatt
            - connection_events
            - advertisements
        serviceID:
          type: string
          example: 12345678-1234-5678-1234-56789abcdef0
        characteristicID:
          type: string
          example: 12345678-1234-5678-1234-56789abcdef1
```

```
## BLE Error codes
ProtocolMap-BLE-ErrorCodes:
```

```

    type: object
    properties:
      nipcStatus:
        type: integer
        format: int32
        enum:
          - 1011 # BLE bonding failed
          - 1013 # BLE service discovery failed
<CODE ENDS>

```

Figure 35

B.3. Protocol map for Zigbee

```

<CODE BEGINS> file "ProtocolMap-Zigbee.yaml"
openapi: 3.0.3
info:
  title: Non IP Device Control (NIPC) Zigbee Protocol Mapping
  description: |-
    Non IP Device Control (NIPC) Zigbee Protocol Mapping.
  termsOfService: http://swagger.io/terms/
  contact:
    email: bbrinckm@cisco.com
  license:
    name: TBD
    url: TBD
  version: 0.10.0
externalDocs:
  description: NIPC IETF draft
  url: https://datatracker.ietf.org/doc/draft-ietf-asdf-nipc/

components:
  schemas:
# Zigbee Protocol Mapping
## Protocol Mapping for Zigbee Service Map
    ProtocolMap-Zigbee-ServiceMap:
      required:
        - zigbee
      type: object
      properties:
        zigbee:
          type: array
          items:
            $ref: '#/components/schemas/ProtocolMap-Zigbee-Endpoint'

    ProtocolMap-Zigbee-Endpoint:
      required:
        - endpointID

```

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

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

```
ProtocolMap-Zigbee-Property:
  required:
    - propertyID
    - propertyType
  type: object
  properties:
    propertyID:
      type: integer
      format: int32
      example: 1
    propertyType:
      type: integer
      format: int32
      example: 32
```

Protocol Mapping for Zigbee broadcast

```
ProtocolMap-Zigbee-Broadcast:
  required:
    - zigbee
  type: object
  properties:
```

```
    zigbee:
      type: object

## Protocol mapping for Zigbee property
ProtocolMap-Zigbee-Propmap:
  required:
    - zigbee
  type: object
  properties:
    zigbee:
      required:
        - endpointID
        - clusterID
        - propertyID
      type: object
      properties:
        endpointID:
          type: integer
          format: int32
          example: 1
        clusterID:
          type: integer
          format: int32
          example: 6
        propertyID:
          type: integer
          format: int32
          example: 16
      type:
        type: integer
        format: int32
        example: 1

ProtocolMap-Zigbee-Event:
  allOf:
    - $ref: '#/components/schemas/ProtocolMap-Zigbee-Propmap'

## Zigbee Error codes
ProtocolMap-Zigbee-ErrorCodes:
  type: object
  properties:
    nipcStatus:
      type: integer
      format: int32
      enum:
        - 1021 # Zigbee join failed
<CODE ENDS>
```

Figure 36

Appendix C. NIPC API extensions

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

C.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/2020-12/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: 0.9.0
externalDocs:
  description: NIPC IETF draft
  url: https://datatracker.ietf.org/doc/draft-ietf-asdf-nipc/
servers:
- url: "{gw_host}/nipc/draft-11"
  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: 12345678-1234-5678-1234-56789abcdef4
- name: propertyName
  in: query
  description: |-
    The SDF property name that needs to be written to.
  required: true
  schema:
    type: string
    example: "https://example.com/hearttrate#/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>
```

C.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/2020-12/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: 0.9.0
externalDocs:
  description: NIPC IETF draft
  url: https://datatracker.ietf.org/doc/draft-ietf-asdf-nipc/
servers:
  - url: "{gw_host}/nipc/draft-11"
    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: 12345678-1234-5678-1234-56789abcdef4
    requestBody:
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Extension-Bulk'
          examples:
            bulkRequest:
              $ref: '#/components/examples/bulkRequest'
        required: true
    responses:
      '202':
        description: Accepted
        headers:
          Location:
            schema:
              type: string
              description: URL to get the bulk status response
              example: /extensions/12345678-1234-5678-1234-56789abcd\
ef4/bulk/status?requestId=12345678-1234-5678-1234-56789abcdef4
      '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: 12345678-1234-5678-1234-56789abcdef4
      - name: requestId
        in: query
        description: Request ID of the bulk operation
        required: true
        schema:
          type: string
          format: uuid
          example: 12345678-1234-5678-1234-56789abcdef4
    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'
          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: 12345678-1234-5678-1234-56789abcdef4
      - name: requestId
        in: query
        description: Request ID of the bulk operation
        required: true
        schema:
          type: string
          format: uuid
          example: 12345678-1234-5678-1234-56789abcdef4
    responses:
      '200':
        description: OK
        headers:
        content:
          application/json:
            schema:
              allOf:
                - $ref: '../Extension-ReadConditional.yaml#/compone\

```

```
nts/schemas/Extension-StatusResponse'
  '303':
    description: See Other
    headers:
      Location:
        schema:
          type: string
        description: URL to get the bulk response
        example: /extensions/12345678-1234-5678-1234-56789abcd\
ef4/bulk?requestId=12345678-1234-5678-1234-56789abcdef4
    content:
      application/json:
        schema:
          allOf:
            - $ref: './Extension-ReadConditional.yaml#/compone\
nts/schemas/Extension-StatusResponse'
        examples:
          successExample:
            summary: Success
            value:
              status: COMPLETED

components:
  schemas:
# Extensions
## Bulk schema Extension
  Extension-Bulk:
    allOf:
      - $ref: './Extension-ReadConditional.yaml#/components/schema\
s/Extension-Callback'
      - type: object
        properties:
          operations:
            type: array
            items:
              $ref: './#/components/schemas/Extension-BulkOperation'

## 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={propertyName\
e}
        - /devices/{id}/actions/?actionName={actionName}
        - /extensions/{id}/properties/read/conditional?prope\
rtyName={propertyName}
      example: /devices/12345678-1234-5678-1234-56789abcdef4\
/properties?propertyName=https://example.com/thermometer%23/sdfThing\
/thermometer/sdfProperty/temperature
    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={propertyName\
e}
            - /devices/{id}/actions/?actionName={actionName}

```

```

      - /extensions/{id}/properties/read/conditional?prope\
rtyName={propertyName}
      example: /devices/12345678-1234-5678-1234-56789abcdef4\
/properties?propertyName=https://example.com/thermometer%23/sdfThing\
/thermometer/sdfProperty/temperature
      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/12345678-1234-5678-1234-56789abcdef4/prop\
erties?propertyName=https://example.com/thermometer%23/sdfThing/ther\
mometer/sdfProperty/temperature
            - method: PUT
              path: /devices/12345678-1234-5678-1234-56789abcdef4/prop\
erties?propertyName=https://example.com/thermometer%23/sdfThing/ther\
mometer/sdfProperty/temperature
              data:
                value: dGVzdA==
            - method: POST
              path: /extensions/12345678-1234-5678-1234-56789abcdef4/p\
roperties/read/conditional?propertyName=https://example.com/thermome\
ter%23/sdfThing/thermometer/sdfProperty/temperature
              data:
                value: dGVzdA==
                maxRepeat: 5
                retryTime: 1
      bulkResponse:
        summary: Bulk response example
        value:
          operations:
            - method: GET
              path: /devices/12345678-1234-5678-1234-56789abcdef4/prop\
erties?propertyName=https://example.com/thermometer%23/sdfThing/ther\
mometer/sdfProperty/temperature
              response:
                value: dGVzdA==
            - method: PUT
              path: /devices/12345678-1234-5678-1234-56789abcdef4/prop\

```

```

erties?propertyName=https://example.com/thermometer%23/sdfThing/ther\
mometer/sdfProperty/temperature
    response:
        status: 200
    - method: POST
        path: /extensions/12345678-1234-5678-1234-56789abcdef4/p\
roperties/read/conditional?propertyName=https://example.com/thermome\
ter%23/sdfThing/thermometer/sdfProperty/temperature
    response:
        value: dGVzdA==
errorBulkResponse:
    summary: Error Bulk response example
    value:
        operations:
            - method: GET
                path: /devices/12345678-1234-5678-1234-56789abcdef4/prop\
erties?propertyName=https://example.com/thermometer%23/sdfThing/ther\
mometer/sdfProperty/temperature
            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/12345678-1234-5678-1234-56789abcdef4/prop\
erties?propertyName=https://example.com/thermometer%23/sdfThing/ther\
mometer/sdfProperty/temperature
            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/12345678-1234-5678-1234-56789abcdef4/p\
roperties/read/conditional?propertyName=https://example.com/thermome\
ter%23/sdfThing/thermometer/sdfProperty/temperature
            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
<CODE ENDS>

```

C.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/2020-12/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: 0.9.0
externalDocs:
  description: NIPC IETF draft
  url: https://datatracker.ietf.org/doc/draft-ietf-asdf-nipc/
servers:
  - url: "{gw_host}/nipc/draft-11"
    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: 12345678-1234-5678-1234-56789abcdef4
  - name: propertyName
    in: query
    description: |-
      The SDF property name that needs to be written to.
    required: true
    schema:
      type: string
      example: "https://example.com/hearttrate#/sdfObject/thermos\
tat/sdfProperty/firmware"
  requestBody:
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Extension-File'
    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 File Extension
    Extension-File:
      required:
        - fileURL
      type: object
      properties:
        fileURL:
          type: string

```

```
        example: "https://domain.com/firmware.dat"
        chunksize:
          type: integer
<CODE ENDS>
```

C.4. NIPC API firmware update extension

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

# yaml-language-server: $schema=https://json-schema.org/draft/2020-12/schema
openapi: 3.0.3
info:
  title: Non IP Device Control (NIPC) API firmware upgrade extension
  description: |-
    Non IP Device Control (NIPC) API firmware upgrade extension,
    requires the file extension
  termsOfService: http://swagger.io/terms/
  contact:
    email: bbrinckm@cisco.com
  license:
    name: TBD
    url: TBD
  version: 0.9.0
externalDocs:
  description: NIPC IETF draft
  url: https://datatracker.ietf.org/doc/draft-ietf-asdf-nipc/
servers:
  - url: "{gw_host}/nipc/draft-11"
    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}/firmware:
    put:
      tags:
        - NIPC API extensions
      summary: Upgrade the firmware of a device
      description: |-
```

```

    Update the firmware of a device. Will perform all operations
    required to upgrade the firmware. ID cannot be a group-id.
  operationId: upgradeFirmware
  parameters:
  - name: id
    in: path
    description: The ID of the device. Group ID is not allowed.
    required: true
    schema:
      type: string
      format: uuid
      example: 12345678-1234-5678-1234-56789abcdef4
  requestBody:
    content:
      application/json:
        schema:
          allOf:
            - $ref: './Extension-File.yaml#/components/schemas/E\
xtension-File'
            - $ref: './#/components/schemas/Extension-Firmware'
            - $ref: './Extension-ReadConditional.yaml#/component\
s/schemas/Extension-Callback'
          required: true
        responses:
          '202':
            description: Accepted
            headers:
              Location:
                schema:
                  type: string
                  description: Location of the resource
                  example: /12345678-1234-5678-1234-56789abcdef4/extensi\
on/firmware/status?requestId=12345678-1234-5678-1234-56789abcdef4
            default:
              description: Error response
              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:
        anyOf:
          - $ref: '../NIPC.yaml#/components/schemas/Id'
          - allOf:
            - $ref: '../NIPC.yaml#/components/schemas/\
FailureResponse'
            - $ref: '#/components/schemas/ExtensionErr\
orCodes'
      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 the status of a firmware upgrade of a device
  description: |-
    Get the status of a firmware upgrade of a device.
    Returns success when ongoing or completed, with a reason.
    Returns failure when upgrade has failed.
    ID cannot be a group-id.
  operationId: upgradeFirmwareStatus
  parameters:
    - name: id
      in: path
      description: The ID of the device. Group ID is not allowed.
      required: true
      schema:
        type: string
        format: uuid
        example: 12345678-1234-5678-1234-56789abcdef4
    - name: requestId
      in: query
      description: Request ID of the firmware upgrade operation
      required: true
      schema:
        type: string
        example: 12345678-1234-5678-1234-56789abcdef4
  responses:
```

```

    '200':
      description: OK
      headers:
        Location:
          schema:
            type: string
            description: Location of the resource
      content:
        application/json:
          schema:
            $ref: '../NIPC.yaml#/components/schemas/Id'
    '400':
      description: Bad request
    '401':
      description: Unauthorized
    '405':
      description: Invalid request
    '500':
      description: Server-side failure
      content:
        application/json:
          schema:
            allOf:
              - $ref: '../NIPC.yaml#/components/schemas/FailureR\
esponse'
              - $ref: '#/components/schemas/ExtensionErrorCodes'
    /extensions/{id}/firmware/status:
      get:
        tags:
          - NIPC API extensions
        summary: Get the status of a firmware upgrade of a device
        description: |-
          Get the status of a firmware upgrade of a device.
          Returns success when ongoing or completed, with a reason.
          Returns failure when upgrade has failed.
          ID cannot be a group-id.
        operationId: upgradeFirmwareStatus
        parameters:
          - name: id
            in: path
            description: The ID of the device. Group ID is not allowed.
            required: true
            schema:
              type: string
              format: uuid
              example: 12345678-1234-5678-1234-56789abcdef4
          - name: requestId
            in: query

```

```
    description: Request ID of the firmware upgrade operation
    required: true
    schema:
      type: string
      example: 12345678-1234-5678-1234-56789abcdef4
  responses:
    '200':
      description: Success
      headers:
      content:
        application/json:
          schema:
            allOf:
              - $ref: '../NIPC.yaml#/components/schemas/Id'
              - $ref: '../Extension-ReadConditional.yaml#/components/schemas/Extension-StatusResponse'
    '303':
      description: See Other
      headers:
        Location:
          schema:
            type: string
            description: URL to get the firmware response
            example: /12345678-1234-5678-1234-56789abcdef4/extension/firmware?requestId=12345678-1234-5678-1234-56789abcdef4
      content:
        application/json:
          schema:
            allOf:
              - $ref: '../NIPC.yaml#/components/schemas/Id'
              - $ref: '../Extension-ReadConditional.yaml#/components/schemas/Extension-StatusResponse'
      examples:
        successExample:
          summary: Completed
          value:
            id: 12345678-1234-5678-1234-56789abcdef4
            status: COMPLETED

  components:
    schemas:
      # Extensions
      ## a Firmware Extension
      Extension-Firmware:
        type: object
        properties:
          firmware:
            type: string
```

```
    enum:
      - nordic
      - silabs

  ExtensionErrorCodes:
    type: object
    properties:
      nipcStatus:
        type: integer
        format: int32
        enum:
          - 1600 # Firmware rollback
          - 1601 # Firmware update failed

<CODE ENDS>
```

C.5. 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/2020-12/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: 0.9.0
externalDocs:
  description: NIPC IETF draft
  url: https://datatracker.ietf.org/doc/draft-ietf-asdf-nipc/
servers:
  - url: "{gw_host}/nipc/draft-11"
    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: 12345678-1234-5678-1234-56789abcdef4
        - 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
                description: |-
                  URL to get the conditional read status
                example: /12345678-1234-5678-1234-56789abcdef4/extension/property/temperature/read/conditional/status?requestId=12345678-1\
234-5678-1234-56789abcdef4
              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: 12345678-1234-5678-1234-56789abcdef4
                                property: https://example.com/heartrate#/s\
dfObject/thermostat/sdfProperty/temperature
                                value: dGVzdA==
                        failedResponse:
                            summary: Failed
                            value:
                                id: 12345678-1234-5678-1234-56789abcdef4
                                status: 400
                                nipcStatus: 1000
                                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: 12345678-1234-5678-1234-56789abcdef4
- 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: requestId
  in: query
  description: |-
    Request ID of the conditional read operation
  required: true
  schema:
    type: string
    format: uuid
    example: 12345678-1234-5678-1234-56789abcdef4
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: 12345678-1234-5678-1234-56789abcdef4
            - 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: requestId
              in: query
              description: Request ID of the conditional read operation
              required: true
              schema:
                type: string
                format: uuid
                example: 12345678-1234-5678-1234-56789abcdef4
          responses:
            '200':
              description: OK
              headers:
              content:
                application/json:
                  schema:
                    allOf:
                      - $ref: '#/components/schemas/Extension-StatusResp\
```

```
onse'
  '303':
    description: See Other
    headers:
      Location:
        schema:
          type: string
        description: URL to get the conditional read response
        example: /12345678-1234-5678-1234-56789abcdef4/extensi\
on/property/temperature/read/conditional?requestId=12345678-1234-567\
8-1234-56789abcdef4
      content:
        application/json:
          schema:
            allOf:
              - $ref: '#/components/schemas/Extension-StatusResp\
onse'
    examples:
      successExample:
        summary: Completed
        value:
          id: 12345678-1234-5678-1234-56789abcdef4
          status: COMPLETED

components:
  schemas:
# Extensions
  Extension-Callback:
    type: object
    properties:
      callback:
        type: object
        properties:
          url:
            description: |-
              URL to send the callback to
              (default is the same as the request URL)
            type: string
            example: "http://localhost:8080/callback"
      headers:
        description: |-
          Headers to include in the callback
          (default is empty)
        type: object
        example:
          x-api-key: "1234567890"
        additionalProperties:
          type: string
```

```
Extension-StatusResponse:
  type: object
  properties:
    status:
      description: |-
        Status of the callback
      type: string
      enum:
        - IN_PROGRESS
        - COMPLETED

Extension-ConditionalRead:
  allof:
    - $ref: '../NIPC.yaml#/components/schemas/Value'
    - $ref: '#/components/schemas/Extension-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>

C.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 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: 0.9.0
externalDocs:
  description: NIPC IETF draft
  url: https://datatracker.ietf.org/doc/draft-ietf-asdf-nipc/
servers:
  - url: "{gw_host}/nipc/draft-11"
    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}/manage/transmit:
  post:
    tags:
      - NIPC API extensions
    summary: Broadcast to a device
    description: |-
      Broadcast a payload to a device. The broadcast is performed \
on the AP where the device was last seen
    operationId: ActionBroadcast
    parameters:
      - name: id
        in: path
        description: The ID of the device. Group ID is not allowed.
        required: true
        schema:
          type: string
          format: uuid
          example: 12345678-1234-5678-1234-56789abcdef4
    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: 12345678-1234-5678-1234-56789abcdef4
    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: 12345678-1234-5678-1234-56789abcdef4
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: '../protocolmaps/ProtocolMap.yaml#/components/schema\
s/ProtocolMap-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 D. 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",
      "sdfOutputData": {
        "sdfProtocolMap": {
          "ble": {
            "type": "gatt",
            "serviceID": "1809",
            "characteristicID": "2A1C"
          }
        }
      }
    },
    "intermediate_temperature": {
      "description": "Intermediate Temperature",
      "sdfOutputData": {
        "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",
        "sdfOutputData": {
            "sdfProtocolMap": {
                "ble": {
                    "type": "advertisements"
                }
            }
        }
    }
},
"isConnected": {
    "description": "BLE connection events",
    "sdfOutputData": {
        "sdfProtocolMap": {
            "ble": {
                "type": "connection_events"
            }
        }
    }
}
```

```
<CODE ENDS>  
}  
}  
}  
}  
}  
}
```

Figure 37: Example SDF model with protocol mappings for BLE

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