

Independant Submission
Internet-Draft
Intended status: Informational
Expires: 13 June 2026

P. Duffy (ed)
Cisco Systems, Inc.
10 December 2025

Cisco's CoAP Simple Management Protocol
draft-duffy-csmp-10

Abstract

CoAP Simple Management Protocol (CSMP) is purpose-built to provide lifecycle management for resource constrained IoT devices deployed within large-scale, bandwidth constrained IoT networks. CSMP offers an efficient transport and message encoding supporting classic NMS functions such as device on-boarding, device configuration, device status reporting, securing the network, etc.

This document describes the design and operation of CSMP. This document does not represent an IETF consensus.

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 13 June 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.

Table of Contents

1. Introduction	3
2. Conventions and Definitions	4
3. Protocol Specification	4
3.1. CoAP Usage Profile	4
3.2. Interface Specification	5
3.2.1. CSMP Device Interface	5
3.2.2. CSMP NMS Interface	9
3.2.3. CSMP Common Components	11
3.3. Resources	18
3.3.1. Base URL	18
3.3.2. Resource Encoding	19
3.3.3. Large Requests	63
3.4. CSMP Security Model	63
3.4.1. Signature Exemption	64
3.5. Device Groups	64
3.5.1. Reserved Group Types	65
3.6. Device TLV Processing Order	65
4. Functional Description	66
4.1. Device Lifecycle States	66
4.2. NMS Discovery	67
4.3. Device Registration and Configuration	67
4.3.1. Device Registration Request	68
4.3.2. Registration POST Payload	68
4.3.3. NMS Registration Response	69
4.3.4. Registration Complete	71
4.4. Device Metrics Reporting	71
4.5. Device Firmware Update	73
4.5.1. Firmware Image Format	73
4.5.2. Firmware Download	76
4.5.3. Image Activation	78
4.5.4. Set Backup Image	79
4.6. Device Commands	80
5. Security Considerations	81
6. IANA Considerations	82
7. Contributors	82
8. Appendix A Registration Retry Example	82
9. Appendix B Change Log	83
9.1. draft 00 to 01	83
9.2. draft 01 to 02	83
9.3. draft 02 to 03	84
9.4. draft 03 to 04	84
9.5. draft 04 to 05	84
9.6. draft 05 to 06	84
9.7. draft 06 to 07	84
9.8. draft 07 to 08	84
9.9. draft 08 to 09	84

9.10. draft 09 to 10	84
10. Normative References	84
Author's Address	85

1. Introduction

Low Power Wide Area Network (LPWAN) technologies provide long range, low power connectivity for Internet of Things (IoT) applications. LPWANs typically operate over distances of several kilometers with link bandwidths as low as 10s of Kbps. LPWAN devices are often compute, storage and power constrained (often optimized to operate for years on a single battery charge).

This specification describes the design and operation of Cisco's CSMP which today is in-field managing approximately 25 million LPWAN devices deployed by a number of vendors. These devices are supporting a variety of critical infrastructure use cases for electric, water, and gas utilities along with a variety of smart cities use cases (municipal lighting, etc.).

There is industry demand that CSMP managed devices be more quickly and widely available for an ever expanding set of use cases. This specification is intended to promote development of interoperable CSMP device and management system implementations. It is anticipated that this specification will be referenced by several industry fora.

CSMP features include:

1. Onboarding. Device startup registration and capabilities announcement with an NMS.
2. Configuration management. Device acquisition of configuration from the NMS. Subsequent NMS configuration reads and updates to the device.
3. Metrics reporting. Periodic device metrics reporting to the NMS. NMS on-demand metrics requests to a device.
4. NMS commanded device operations. NMS command issuance to a single device or group of devices.
5. Secure device firmware update.

2. Conventions and Definitions

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.

3. Protocol Specification

CSMP is a usage profile of the Constrained Application Protocol [RFC7252], which is designed for implementing RESTful messaging for resource constrained devices. It is fair to view CoAP as a binary encoded functional subset of HTTP operating over UDP which also supports multicast messaging. Resources (addressable objects) transported within CSMP message payloads are implemented using the Protocol Buffers compact binary encoding [PB].

It is assumed the reader is familiar with:

1. The basic concepts of RESTful architecture.
2. The operation, message formats, and terminology of CoAP [RFC7252].
3. Protocol Buffers [PB], which is used to describe and encode CSMP message payloads.
4. OpenAPI [OPENAPI] is the interface definition language used to define CSMP Device and NMS interfaces.

3.1. CoAP Usage Profile

The NMS and devices communicate directly using CoAP. Acting as a CoAP client, a device issues RESTful requests to read or modify specific resources (objects) exposed by the NMS server. Likewise, the device serves requests from the NMS to manipulate resources exposed by the device.

CSMP specializes the usage of CoAP in the following ways:

1. CSMP does not use the Token capabilities described in [RFC7252]. Request/response messaging MUST be implemented using the "synchronous" form of a CON request with response piggybacked in the subsequent ACK. A client MUST elide the Token field from the request message. The server SHOULD ignore the Token if received from a client request.

2. Due to high latencies typical of LPWAN technologies, a client MUST NOT use the CoAP retransmission model when sending a CON message to a server. After sending a CON message, the client MUST accept a response from the server at any time up until the device sends its next CON message (containing a new CoAP Message ID).

3.2. Interface Specification

CSMP defines a CSMP Device interface and a CSMP NMS interface. Each of these interfaces defines a set of resources (objects), the addresses at which these resources exist (URLs), and the methods (GET, PUT, POST, DELETE) which may be used to manipulate the resources to implement device management operations. The CSMP Device and CSMP NMS interfaces are expressed using the OpenAPI interface definition language.

OpenAPI's heritage is the expression of interfaces typically constructed with HTTP and JSON. This document defines a few conventions required to express a CoAP interface in OpenAPI:

1. The generic RESTful verb designators GET, PUT, POST, and DELETE are used along with the text presentation of a resource's URL. This is done for developer readability. An actual implementation must properly encode the CoAP message Code field along with the Uri-Host, Uri-Port, Uri-Path, and Uri-Query options as described in [RFC7252].
2. CoAP Response codes of the form X.YZ are expressed in the OpenAPI as XYZ (the actual response code is scaled by 100 in the OpenAPI rendering).
3. Placeholder objects are defined in OpenAPI to represent the Protocol Buffer binary objects which will be transported in CoAP messaging payloads. The actual binary objects are defined in a separate Protocol Buffer file Section 3.3.2.3.
4. CoAP supports the NON messaging pattern. OpenAPI syntax always requires a response be defined. The CSMP interface definitions will note when a response is for a COAP NON request and not an actual CoAP response (no response is sent).

3.2.1. CSMP Device Interface

A CSMP device MUST implement the interface described below. The interface definition is also available at [CSMPDEV] which is directly importable into development tooling.

Various forms of the GET method are used for retrieving registration information, device information, and monitoring information from the devices. Various forms of the POST method are used to deliver configuration and commands to devices. Usage of this interface is detailed in the sections which follow.

```
openapi: 3.0.0
info:
  description: |
    CSMP Device Interface. RFC 2119 language is used to indicate
    normative statements.
  version: 1.0.1
  title: CSMP Device Interface.
servers:
  #
  # Placeholder for the actual device-url
  #
  - url: 'https://CsmDevice/1.0.1'

paths:
  /c:
    get:
      description: |
        Retrieve TlvIndex TLV or specific list of TLVs from a device.
      parameters:
        - $ref: '#/components/parameters/qOption'
        - $ref: '#/components/parameters/aOption'
        - $ref: '#/components/parameters/rOption'

      responses:
        '205':
          description: |
            Successful return of requested TLVs or TlvIndex.
            Device MUST return the requested specific TLVs when
            the "q" param is included (full tables are allowed).
            Device MUST return only TlvIndex when the "q" param
            is omitted.
          content:
            application/octet-stream:
              schema:
                oneOf:
                  - $ref: 'https://TLVsGETFromDevice'
                  - $ref: 'https://TlvIndex'
        '400':
          description: Bad Request.
        '401':
          description: Unauthorized.
        '402':
```

```
    description: Bad option.
  '403':
    description: TLV not found, response won't fit in MTU.
  '404':
    description: Not found.
  '405':
    description: Method not allowed.
```

post:

```
  description: Process TLV updates from the NMS.
  The q option MUST NOT be used.
  A device MUST process the following TLVs (if present, in order)
  1. Signature - verify the message signature.
  2. SignatureValidity - verify if the signature is still valid.
  3. GroupMatch - confirm device belongs indicated group.
  4. The TLVs to be updated.
```

parameters:

```
- $ref: '#/components/parameters/aOption'
- $ref: '#/components/parameters/rOption'
```

requestBody:

```
  description: One or more TLVs to be POSTed to the device.
  required: true
  content:
    application/octet-stream:
      schema:
        $ref: 'https://TLVsPOSTableToDevice'
```

responses:

```
  '201':
    description: TLVs have been updated.
  '400':
    description: Bad Request.
  '401':
    description: Unauthorized.
  '402':
    description: Bad option.
  '403':
    description: Forbidden.
  '404':
    description: Not found.
  '405':
    description: Method not allowed.
```

/c/{tlvIdPath}:

get:

```
  description: |
    Retrieve single TLV from a device.
  parameters:
```

- \$ref: '#/components/parameters/tlvIdPath'
- \$ref: '#/components/parameters/aOption'
- \$ref: '#/components/parameters/rOption'

responses:

'205':
 description: Return the requested TLV. Full table retrieval is allowed.
 content:
 application/octet-stream:
 schema:
 \$ref: 'https://TLVsGETFromDevice'
'400':
 description: Bad Request.
'401':
 description: Unauthorized.
'402':
 description: Bad option.
'403':
 description: TLV not found, response won't fit in MTU.
'404':
 description: Not found.
'405':
 description: Method not allowed.

Note that POST is not permitted on this URL.
#

Query parameters used for device requests.
#

components:

parameters:

qOption:
 description: Q option for request to device for one or more specific TLVs. When present in a request, the response payload MUST contain the requested TLVs. If a TLV does not exist on the device, it MUST NOT be returned in the response and no error should be indicated.

in: query

name: q

required: false

schema:

type: string

description: Format is <tlvID1>+<tlvID2>+<tlvID3>+...

aOption:

description: |
 A option. When omitted, the request MUST be CON.
 When present, request MUST be NON and the device

MUST wait random interval from time of receipt up to "a" seconds before responding with a NON POST to <nms-url>/c. This form is often used with multicast requests to devices, but may also be used for unicast requests to devices.

in: query

name: a

required: false

schema:

type: integer

description: The device will wait random interval from receipt up to "a" seconds before sending.

rOption:

description: |

R option MAY be used with the A option to indicate device MUST respond with a NON POST to specified URL (overriding <nms-url>/c).

in: query

name: r

required: false

schema:

type: string

tlvIdPath:

description: Path param specifying TLV ID.

in: path

name: tlvIdPath

required: true

schema:

type: integer

3.2.2. CSMP NMS Interface

A CSMP NMS MUST implement the interface described below. The interface definition is also available at [CSMPNMS] which is directly importable into development tooling.

Various forms of the POST method are used for device registration, device metrics reporting, and asynchronous GET responses (resulting from a request including the "a" option and default response URL). Usage of this interface is detailed in the sections which follow.

openapi: 3.0.0

info:

description: CSMP NMS Interface. RFC 2119 language is used to indicate normative statements.

version: 1.0.1

title: CSMP NMS Interface

```
servers:
  #
  # Placeholder for the actual nms-url.
  #
  - url: 'https://CsmpNms/1.0.0'

paths:
  /r:
    post:
      description: Device registration POST to the NMS.
        RequestBody MAY include any of the RegistrationRequest
        TLVs (see CsmpComponents). RequestBody MUST include
        DeviceID and CurrentTime TLVs. POSTs to this address MUST
        be CON (acknowledged).
      requestBody:
        required: true
        content:
          application/octet-stream:
            schema:
              $ref: 'https://RegistrationRequestTLVs'
      responses:
        '203':
          description: Successful registration response to device.
            The response MAY include any of the RegistrationResponse
            TLVs (see CsmpComponents).
          content:
            application/octet-stream:
              schema:
                $ref: 'https://RegistrationResponseTLVs'
        '403':
          description: Duplicate DeviceID was received or DeviceID
            was not recognized or authorized by the NMS.
            There is no payload in this response.
        '404':
          description: Multiple Session IDs were found for the device.
            There is no payload for this response.
    /c:
      post:
        description: Devices POST periodic metrics, command responses,
          events, and asynchronous GET responses to the NMS. A POST to
          this address MUST be NON (non acknowledged... to avoid
          excessive acknowledgement, 4.03 or 4.04 conditions are handled
          silently by the NMS). RequestBody MAY include any of
          TLVsPOSTableToNms TLVs (see CsmpComponents). The RequestBody
          MUST include the CurrentTime and SessionID TLVs
          (see CsmpComponents).
        requestBody:
          required: true
```

```
    content:
      application/octet-stream:
        schema:
          $ref: 'https://TLVsPostAbleToNMS'
    responses:
      '100':
        description: There is no response to this (NON) request
          (100 is used to indicate no response).
```

3.2.3. CSMP Common Components.

This section contains definitions common to both the CSMP Device and NMS interfaces. The definitions are also available at [CSMPCOMP] which is directly importable into development tooling.

```
# OpenAPI version identifier - required for OpenAPI 3.0 domains
openapi: 3.0.0
```

```
info:
  title: CSMP Components
  description: Common components for CSMP APIs
  version: '1.0'
```

```
components:
  schemas:
    TLVsGetAbleFromDevice:
      description: |
        TLVs which are GET-able from a <device-url>/c.
        Metrics and settings.
      anyOf:
        - $ref: '#/components/schemas/TlvIndex'
        - $ref: '#/components/schemas/HardwareDesc'
        - $ref: '#/components/schemas/InterfaceDesc'
        - $ref: '#/components/schemas/ReportSubscribe'
        - $ref: '#/components/schemas/IPAddress'
        - $ref: '#/components/schemas/IPRoute'
        - $ref: '#/components/schemas/CurrentTime'
        - $ref: '#/components/schemas/RPLSettings'
        - $ref: '#/components/schemas/Uptime'
        - $ref: '#/components/schemas/InterfaceMetrics'
        - $ref: '#/components/schemas/IPRouterRPLMetrics'
        - $ref: '#/components/schemas/Ieee8021xStatus'
        - $ref: '#/components/schemas/Ieee80211iStatus'
        - $ref: '#/components/schemas/WPANStatus'
        - $ref: '#/components/schemas/DHCP6ClientStatus'
        - $ref: '#/components/schemas/NMSSettings'
        - $ref: '#/components/schemas/NMSStatus'
        - $ref: '#/components/schemas/Ieee8021xSettings'
```

- \$ref: '#/components/schemas/Ieee802154BeaconStats'
- \$ref: '#/components/schemas/RPLInstance'
- \$ref: '#/components/schemas/FirmwareImageInfo'
- \$ref: '#/components/schemas/SysResetStats'
- \$ref: '#/components/schemas/MplStats'
- \$ref: '#/components/schemas/RPLStats'
- \$ref: '#/components/schemas/DHCP6Stats'
- \$ref: '#/components/schemas/EventReport'
- \$ref: '#/components/schemas/EventIndex'
- \$ref: '#/components/schemas/EventSubscribe'
- \$ref: '#/components/schemas/EventStats'
- \$ref: '#/components/schemas/NetworkRole'
- \$ref: '#/components/schemas/NetStat'

RegistrationRequestTLVs:

- description: |
TLVs which may be contained in a registration POST request to the NMS.
- anyOf:
- \$ref: '#/components/schemas/DeviceID'
 - \$ref: '#/components/schemas/CurrentTime'
 - \$ref: '#/components/schemas/HardwareDesc'
 - \$ref: '#/components/schemas/InterfaceDesc'
 - \$ref: '#/components/schemas/IPAddress'
 - \$ref: '#/components/schemas/NMSStatus'
 - \$ref: '#/components/schemas/WPANStatus'
 - \$ref: '#/components/schemas/RPLSettings'
 - \$ref: '#/components/schemas/SessionID'
 - \$ref: '#/components/schemas/GroupInfo'
 - \$ref: '#/components/schemas/ReportSubscribe'

RegistrationResponseTLVs:

- description: |
TLVs which may be contained in a registration response from the NMS.
- anyOf:
- \$ref: '#/components/schemas/SessionID'
 - \$ref: '#/components/schemas/GroupAssign'
 - \$ref: '#/components/schemas/ReportSubscribe'
 - \$ref: '#/components/schemas/DescriptionRequest'
 - \$ref: '#/components/schemas/NMSRedirectRequest'

TLVsPostAbleToNMS:

- description: |
TLVs which may be POSTed to <nms-url>/c
There are several situations under which this can occur ...

1. Periodic metrics reports from a device.
2. Command responses from a device.
3. Asynchronous events reported by a device.
4. Asynchronous GET responses from a device.

anyOf:

- \$ref: '#/components/schemas/TlvIndex'
- \$ref: '#/components/schemas/DeviceID'
- \$ref: '#/components/schemas/NMSRedirectRequest'
- \$ref: '#/components/schemas/SessionID'
- \$ref: '#/components/schemas/HardwareDesc'
- \$ref: '#/components/schemas/InterfaceDesc'
- \$ref: '#/components/schemas/ReportSubscribe'
- \$ref: '#/components/schemas/IPAddress'
- \$ref: '#/components/schemas/IPRoute'
- \$ref: '#/components/schemas/CurrentTime'
- \$ref: '#/components/schemas/RPLSettings'
- \$ref: '#/components/schemas/Uptime'
- \$ref: '#/components/schemas/InterfaceMetrics'
- \$ref: '#/components/schemas/IPRouterRPLMetrics'
- \$ref: '#/components/schemas/PingResponse'
- \$ref: '#/components/schemas/Ieee8021xStatus'
- \$ref: '#/components/schemas/Ieee80211iStatus'
- \$ref: '#/components/schemas/WPANStatus'
- \$ref: '#/components/schemas/DHCP6ClientStatus'
- \$ref: '#/components/schemas/NMSSettings'
- \$ref: '#/components/schemas/NMSStatus'
- \$ref: '#/components/schemas/Ieee8021xSettings'
- \$ref: '#/components/schemas/Ieee802154BeaconStats'
- \$ref: '#/components/schemas/RPLInstance'
- \$ref: '#/components/schemas/GroupInfo'
- \$ref: '#/components/schemas/LowpanMacStats'
- \$ref: '#/components/schemas/LowpanPhySettings'
- \$ref: '#/components/schemas/TransferResponse'
- \$ref: '#/components/schemas/LoadResponse'
- \$ref: '#/components/schemas/CancelLoadResponse'
- \$ref: '#/components/schemas/SetBackupResponse'
- \$ref: '#/components/schemas/FirmwareImageInfo'
- \$ref: '#/components/schemas/SysResetStats'
- \$ref: '#/components/schemas/MplStats'
- \$ref: '#/components/schemas/RPLStats'
- \$ref: '#/components/schemas/DHCP6Stats'
- \$ref: '#/components/schemas/EventReport'
- \$ref: '#/components/schemas/EventIndex'
- \$ref: '#/components/schemas/EventStats'

TLVsPostAbleToDevice:

description: |

TLVs which may be POSTed to <device-url>/c.

Commands and settings.

anyOf:

- \$ref: '#/components/schemas/NMSRedirectRequest'
- \$ref: '#/components/schemas/RPLSettings'
- \$ref: '#/components/schemas/PingRequest'
- \$ref: '#/components/schemas/RebootRequest'
- \$ref: '#/components/schemas/NMSSettings'
- \$ref: '#/components/schemas/Ieee8021xSettings'
- \$ref: '#/components/schemas/GroupMatch'
- \$ref: '#/components/schemas/LowpanPhySettings'
- \$ref: '#/components/schemas/TransferRequest'
- \$ref: '#/components/schemas/ImageBlock'
- \$ref: '#/components/schemas/LoadRequest'
- \$ref: '#/components/schemas/CancelLoadRequest'
- \$ref: '#/components/schemas/SetBackupRequest'
- \$ref: '#/components/schemas/SignatureValidity'
- \$ref: '#/components/schemas/Signature'
- \$ref: '#/components/schemas/SignatureSettings'
- \$ref: '#/components/schemas/NetworkRole'
- \$ref: '#/components/schemas/MplReset'
- \$ref: '#/components/schemas/EventSubscribe'

TableTLVs:

description: |

Table TLVs.

anyOf:

- \$ref: '#/components/schemas/HardwareDesc'
- \$ref: '#/components/schemas/InterfaceDesc'
- \$ref: '#/components/schemas/IPAddress'
- \$ref: '#/components/schemas/IPRoute'
- \$ref: '#/components/schemas/InterfaceMetrics'
- \$ref: '#/components/schemas/IPRouterPLMetrics'
- \$ref: '#/components/schemas/Ieee802154BeaconStats'
- \$ref: '#/components/schemas/RPLInstance'
- \$ref: '#/components/schemas/FirmwareImageInfo'
- \$ref: '#/components/schemas/NetStat'
- \$ref: '#/components/schemas/CertBundle'

AllTLVs:

description: |

All CSMP defined TLVs.

This is used as an editorial starting point for the other TLV collections defined within.

anyOf:

- \$ref: '#/components/schemas/TlvIndex'
- \$ref: '#/components/schemas/DeviceID'
- \$ref: '#/components/schemas/NMSRedirectRequest'
- \$ref: '#/components/schemas/SessionID'

- \$ref: '#/components/schemas/DescriptionRequest'
- \$ref: '#/components/schemas/HardwareDesc'
- \$ref: '#/components/schemas/InterfaceDesc'
- \$ref: '#/components/schemas/ReportSubscribe'
- \$ref: '#/components/schemas/IpAddress'
- \$ref: '#/components/schemas/IPRoute'
- \$ref: '#/components/schemas/CurrentTime'
- \$ref: '#/components/schemas/RPLSettings'
- \$ref: '#/components/schemas/Uptime'
- \$ref: '#/components/schemas/InterfaceMetrics'
- \$ref: '#/components/schemas/IPRouterRPLMetrics'
- \$ref: '#/components/schemas/PingRequest'
- \$ref: '#/components/schemas/PingResponse'
- \$ref: '#/components/schemas/RebootRequest'
- \$ref: '#/components/schemas/Ieee8021xStatus'
- \$ref: '#/components/schemas/Ieee80211iStatus'
- \$ref: '#/components/schemas/WPANStatus'
- \$ref: '#/components/schemas/DHCP6ClientStatus'
- \$ref: '#/components/schemas/NMSSettings'
- \$ref: '#/components/schemas/NMSStatus'
- \$ref: '#/components/schemas/Ieee8021xSettings'
- \$ref: '#/components/schemas/Ieee802154BeaconStats'
- \$ref: '#/components/schemas/RPLInstance'
- \$ref: '#/components/schemas/GroupAssign'
- \$ref: '#/components/schemas/GroupEvict'
- \$ref: '#/components/schemas/GroupMatch'
- \$ref: '#/components/schemas/GroupInfo'
- \$ref: '#/components/schemas/LowpanMacStats'
- \$ref: '#/components/schemas/LowpanPhySettings'
- \$ref: '#/components/schemas/TransferRequest'
- \$ref: '#/components/schemas/ImageBlock'
- \$ref: '#/components/schemas/LoadRequest'
- \$ref: '#/components/schemas/CancelLoadRequest'
- \$ref: '#/components/schemas/SetBackupRequest'
- \$ref: '#/components/schemas/TransferResponse'
- \$ref: '#/components/schemas/LoadResponse'
- \$ref: '#/components/schemas/CancelLoadResponse'
- \$ref: '#/components/schemas/SetBackupResponse'
- \$ref: '#/components/schemas/FirmwareImageInfo'
- \$ref: '#/components/schemas/SignatureValidity'
- \$ref: '#/components/schemas/Signature'
- \$ref: '#/components/schemas/SignatureSettings'
- \$ref: '#/components/schemas/SysResetStats'
- \$ref: '#/components/schemas/NetStat'
- \$ref: '#/components/schemas/NetworkRole'
- \$ref: '#/components/schemas/CertBundle'
- \$ref: '#/components/schemas/MplStats'
- \$ref: '#/components/schemas/MplReset'

```
- $ref: '#/components/schemas/RPLStats'
- $ref: '#/components/schemas/DHCP6Stats'
- $ref: '#/components/schemas/EventReport'
- $ref: '#/components/schemas/EventIndex'
- $ref: '#/components/schemas/EventSubscribe'
- $ref: '#/components/schemas/EventStats'

#
# Object (TLV) placeholders to allow OpenAPI to compile.
# Each of these is authoritatively defined in
# the CSMP TLV definitions.
#
TlvIndex:
  type: object
DeviceID:
  type: object
NMSRedirectRequest:
  type: object
SessionID:
  type: object
DescriptionRequest:
  type: object
HardwareDesc:
  type: object
InterfaceDesc:
  type: object
ReportSubscribe:
  type: object
IPAddress:
  type: object
IPRoute:
  type: object
CurrentTime:
  type: object
RPLSettings:
  type: object
Uptime:
  type: object
InterfaceMetrics:
  type: object
IPRouterPLMetrics:
  type: object
PingRequest:
  type: object
PingResponse:
  type: object
RebootRequest:
  type: object
```


Ieee8021xStatus:
 type: object
Ieee80211iStatus:
 type: object
WPANStatus:
 type: object
DHCP6ClientStatus:
 type: object
NMSSettings:
 type: object
NMSStatus:
 type: object
Ieee8021xSettings:
 type: object
Ieee802154BeaconStats:
 type: object
RPLInstance:
 type: object
GroupAssign:
 type: object
GroupEvict:
 type: object
GroupMatch:
 type: object
GroupInfo:
 type: object
LowpanMacStats:
 type: object
LowpanPhySettings:
 type: object
TransferRequest:
 type: object
ImageBlock:
 type: object
LoadRequest:
 type: object
CancelLoadRequest:
 type: object
SetBackupRequest:
 type: object
TransferResponse:
 type: object
LoadResponse:
 type: object
CancelLoadResponse:
 type: object
SetBackupResponse:
 type: object

```
FirmwareImageInfo:
  type: object
SignatureValidity:
  type: object
Signature:
  type: object
SignatureSettings:
  type: object
SysResetStats:
  type: object
NetStat:
  type: object
NetworkRole:
  type: object
CertBundle:
  type: object
MplStats:
  type: object
MplReset:
  type: object
RPLStats:
  type: object
DHCP6Stats:
  type: object
EventReport:
  type: object
EventIndex:
  type: object
EventSubscribe:
  type: object
EventStats:
  type: object
```

3.3. Resources

CSMP devices and CSMP NMS use the CoAP GET, POST, and DELETE methods to manipulate the resources specified in Section 3.3.2.3, which are the Protocol Buffer object payloads contained in the various CoAP requests and responses. Usage of these payloads is detailed in the sections which follow.

3.3.1. Base URL

A CSMP server is located at a <base-url> of the form
coap://hostname:port/<base-path>

The <base-path> for all CSMP resources at a particular hostname:port MUST be identical.

It is RECOMMENDED that a default <base-path> of "/" be used.

Because an NMS CSMP request message may be multicast to a large number of devices, all CSMP devices within a multicast domain MUST have identical port and <base-path>.

The following <base-path> are reserved for future use:

1. /o

Full details of the Device and NMS service URLs are defined in Section 3.2.1 and Section 3.2.2.

3.3.2. Resource Encoding

The message payloads of CSMP requests and responses MUST be formatted as a sequence of Type-Length-Value objects.

3.3.2.1. Standard TLVs

Standard TLVs have the following format:

Type	Length	Value
------	--------	-------

The Type field is an unsigned integer identifying a specific CSMP TLV ID and MUST be encoded as a Protocol Buffers varint.

The Length field is an unsigned integer containing the number of octets occupied by the Value field. The Length field MUST be encoded as a Protocol Buffers varint.

The Value field MUST contain the Protocol Buffers encoded TLV corresponding to the indicated Type.

The set of objects defined by CSMP and their Type (TLV ID) are specified in Section 3.3.2.3.

3.3.2.2. Vendor Defined TLV

The Vendor Defined TLV follows the same format as a Standard TLV with the following important extensions:

The Type field MUST be set to 127.

The Value field contains one or more SubTLVs of the format

SubType	Length	Value
---------	--------	-------

The SubType field is an unsigned integer containing a vendor defined TLV ID and MUST be encoded as a Protocol Buffers varint. The SubTypes 0 through 15 (inclusive) are reserved for use by this specification. SubTypes 16 onward are available for vendor defined usage.

The Length field is an unsigned integer containing the number of octets occupied by the Value field of the SubTLV. The Length field MUST be encoded as a Protocol Buffers varint.

The Value field MUST contain the Protocol Buffer encoded SubTLV value defined by the vendor.

SubType 1 is reserved by this specification to indicate Vendor Private Enterprise Number (PEN). SubType 1 MUST be the first SubTLV present in the TLV. The corresponding Value field MUST be set to the vendor's PEN.

Registration Procedures for Private Enterprise Numbers are described in [PEN].

3.3.2.3. CSMP TLV Definitions

This section contains the CSMP TLV definitions (defined using [PB]). The definitions are also available at [CSMPMSG] which is directly importable into development tooling.

```
syntax = "proto3";
```

```
/*
```

```
The definitions for CSMP TLVs.
```

```
This file is directly consumable by the Protocol Buffer tool chain. Requirements are specified using the terminology and conventions as referenced in [RFC2119]. Requirements key words referenced in [RFC2119] must be capitalized. Full details re: Protocol Buffers are accessible at https://developers.google.com/protocol-buffers
```

```
*/
```

```
/*
```

```
CSMP TLV ID Mapping to Protocol Buffer messages.
```

A unique TLV ID MUST be assigned to each CSMP Protocol Buffer message (defined within). An unused TLV ID MAY be assigned to a new message (the new TLV ID assignment MUST be recorded within, along with the definition of the new message). A Reserved TLV ID MUST NOT be assigned to a message. An existing TLV ID assignment MUST NOT be re-assigned to a new message.

NOTE the legend.

- Unused: available for assignment.
- Reserved: not available for assignment.
- All other named messages are currently used by CSMP.

All TLV assignments MUST be recorded in the table below.
Take care to maintain the numbering.

TLVID	Message
1	TlvIndex
2	DeviceID
3	Reserved
4	Unused
5	Unused
6	NMSRedirectRequest
7	SessionID
8	DescriptionRequest
9	Unused
10	Reserved
11	HardwareDesc
12	InterfaceDesc
13	ReportSubscribe
14	Reserved
15	Reserved
16	IPAddress
17	IPRoute
18	CurrentTime
19	Reserved
20	Reserved
21	RPLSettings
22	Uptime
23	InterfaceMetrics
24	Reserved
25	IPRouteRPLMetrics
26	Unused
27-29	Reserved
30	PingRequest
31	PingResponse
32	RebootRequest
33	Ieee8021xStatus
34	Ieee80211iStatus
35	WPANStatus
36	DHCP6ClientStatus
37-41	Reserved
42	NMSSettings
43	NMSStatus
44-46	Reserved

47	Ieee8021xSettings
48	Ieee802154BeaconStats
49-52	Reserved
53	RPLInstance
54	Reserved
55	GroupAssign
56	GroupEvict
57	GroupMatch
58	GroupInfo
59	Unused
60	Unused
61	Reserved
62	LowpanMacStats
63	LowpanPhySettings
64	Unused
65	TransferRequest
66	Reserved
67	ImageBlock
68	LoadRequest
69	CancelLoadRequest
70	SetBackupRequest
71	TransferResponse
72	LoadResponse
73	CancelLoadResponse
74	SetBackupResponse
75	FirmwareImageInfo
76	SignatureValidity
77	Signature
78	Reserved
79	SignatureSettings
80	Reserved
81	Reserved
82	Unused
83	Unused
84	Reserved
85	Unused
86	SysResetStats
87	Unused
88	Reserved
89	Unused
90	Unused
91-97	Reserved
98	Unused
99	Unused
100	Reserved
101	Unused
102	Unused
103	Unused

104	Unused
105	Unused
106	Unused
107	Reserved
108	Reserved
109	Unused
110-112	Reserved
113	Unused
114	Unused
115-117	Reserved
118	Unused
119	Unused
120-122	Reserved
123	Unused
124	NetStat
125	Reserved
126	Reserved
127	Vendor Defined TLV
128-131	Reserved
132-139	Unused
140	Reserved
141	NetworkRole
142-151	Reserved
152-154	Unused
155-157	Reserved
158-159	Unused
160-165	Reserved
166-169	Unused
170-171	Reserved
172	CertBundle
173-179	Unused
180	Reserved
181-199	Unused
200-202	Reserved
203-209	Unused
210	Reserved
211	Reserved
212-216	Unused
217-220	Reserved
221-239	Unused
240	Reserved
241	MplStats
242	MplReset
243-299	Unused
301-303	Reserved
304	Unused
305-307	Reserved
308-309	Unused

310-312 Reserved
313 RPLStats
314 DHCP6Stats
315 Reserved
316 Reserved
317-324 Unused
325-337 Reserved
338-339 Unused
340-399 Reserved
400-499 Unused
500 Reserved
501 Reserved
502 Reserved
503-509 Unused
510 Reserved
511 Reserved
512-519 Unused
520 Reserved
521 Reserved
522-529 Unused
530 Reserved
531 Reserved
532-539 Unused
540 Reserved
541-599 Unused
600-607 Reserved
608 ... Unused

*/

/*

Message definitions follow.

Tag notation used within ...

Class:: designates class of device for which the TLV is relevant.

Generic (any IP addressable device)

Mesh (Wi-SUN mesh devices)

Others TBD.

*/

package csmp.tlvs;

option java_package = "com.cisco.cgms.protocols.csmp.tlvs";

// TLV 1

// A list of zero or more TLV IDs

// Class:: Generic

//


```
message TlvIndex {
    // list of TLV IDs (string encoded) supported by the device.
    repeated string tlvid = 1;
}

// TLV 2
// Primary identifier for a specific device.
// Class:: Generic
//

message DeviceID {
    oneof type_present {
        // Set to 1 to indicate EUI-64 format.
        uint32 type = 1;
    }
    oneof id_present {
        // The unique identifier of the device in EUI-64 format
        string id = 2;
    }
}

// TLV 6
// Used by NMS to force device registration to a specific NMS.
// Class:: Generic
//

message NMSRedirectRequest {
    oneof url_present {
        // NMS <base-url> to which the device registration will be directed.
        // MUST be formatted per section 6 of RFC 7252
        string url = 1;
    }
    oneof immediate_present {
        // True == device should immediately send registration request
        // to the specified NMS url.
        bool immediate = 2;
    }
}

// TLV 7
// Session ID used by NMS to track device CSMP messaging.
// Assigned by the NMS, used in all subsequent Device to NMS messaging.
// Class:: Generic
//

message SessionID {
    oneof id_present {
        string id = 1; // session ID
    }
}
```

```
    }  
  }  
  
  // TLV 8  
  // List of zero or more TLVs requested by the NMS from a Device.  
  // The requested TLV values will be sent to the NMS asynchronously.  
  // Class:: Generic  
  //  
  
message DescriptionRequest {  
  // list of TLV IDs in string format.  
  repeated string tlvid = 1;  
}  
  
// A list of hardware modules with their firmware versions.  
//  
message HardwareModule {  
  oneof moduleType_present {  
    // hardware module type. Rf Dsp=1, PLC Dsp=2, CPU=3, FPGA=4  
    uint32 moduleType = 1;  
  }  
  oneof firmwareRev_present {  
    // firmware version of the hardware module  
    string firmwareRev = 2;  
  }  
}  
  
// TLV 11  
// This TLV contains hardware description information for the device.  
// The contents of the fields are defined by the equivalently-named  
// fields in the entry of the SNMP MIB object entPhysicalTable.  
// Class:: Generic  
//  
  
message HardwareDesc {  
  oneof entPhysicalIndex_present {  
    // index of this hardware being described.  
    int32 entPhysicalIndex = 1;  
  }  
  oneof entPhysicalDescr_present {  
    // A textual description of physical entity.  
    string entPhysicalDescr = 2;  
  }  
  oneof entPhysicalVendorType_present {  
    // An indication of the vendor-specific hardware type of the  
    // physical entity.  
    bytes entPhysicalVendorType = 3;  
  }  
}
```

```
oneof entPhysicalContainedIn_present {
    // The value of entPhysicalIndex for the physical entity which
    // 'contains' this physical entity.
    int32 entPhysicalContainedIn = 4;
}
oneof entPhysicalClass_present {
    // An indication of the general hardware type of the physical
    // entity.
    int32 entPhysicalClass = 5;
}
oneof entPhysicalParentRelPos_present {
    // An indication of the relative position of this 'child' component
    // among all its 'sibling' components.
    int32 entPhysicalParentRelPos = 6;
}
oneof entPhysicalName_present {
    // The textual name of the physical entity.
    string entPhysicalName = 7;
}
oneof entPhysicalHardwareRev_present {
    // The vendor-specific hardware revision string for the physical
    // entity.
    string entPhysicalHardwareRev = 8;
}
oneof entPhysicalFirmwareRev_present {
    // The vendor-specific firmware revision string for the physical
    // entity.
    string entPhysicalFirmwareRev = 9;
}
oneof entPhysicalSoftwareRev_present {
    // The vendor-specific software revision string for the physical
    // entity.
    string entPhysicalSoftwareRev = 10;
}
oneof entPhysicalSerialNum_present {
    // The vendor-specific serial number string for the physical
    // entity.
    string entPhysicalSerialNum = 11;
}
oneof entPhysicalMfgName_present {
    // The name of the manufacturer of this physical component.
    string entPhysicalMfgName = 12;
}
oneof entPhysicalModelName_present {
    // The vendor-specific model name identifier string associated
    // with this physical component.
    string entPhysicalModelName = 13;
}
```

```
oneof entPhysicalAssetID_present {
    // This object is a user-assigned asset tracking identifier for
    // the physical entity and provides non-volatile storage of this
    // information.
    string entPhysicalAssetID = 14;
}
oneof entPhysicalMfgDate_present {
    // This object contains the date of manufacturing of the managed
    // entity.
    uint32 entPhysicalMfgDate = 15;
}
oneof entPhysicalURIs_present {
    // This object contains additional identification information about
    // the physical entity.
    string entPhysicalURIs = 16;
}
oneof entPhysicalFunction_present {
    // This field can take the following values: METER = 1,
    // RANGE_EXTENDER = 2, DA_GATEWAY = 3, CGE = 4, ROOT = 5,
    // CONTROLLER = 6, SENSOR = 7, NETWORKNODE = 8.
    uint32 entPhysicalFunction = 17;
}
oneof entPhysicalOUI_present {
    // uniquely identifies a vendor
    bytes entPhysicalOUI = 18;
}
// This defines a list of hardware modules with their
// firmware versions
repeated HardwareModule hwModule = 19;
}

// TLV 12
// This TLV contains description information for an interface on
// the device. The contents of these fields are defined by the
// equivalently-named fields in the SNMP MIB object ifTable.
// Class:: Generic
//

message InterfaceDesc {
    oneof ifIndex_present {
        // A unique value, greater than zero, for each interface.
        int32 ifIndex = 1;
    }
    oneof ifName_present {
        // The textual name of the interface.
        string ifName = 2;
    }
    oneof ifDescr_present {
```

```
// A textual string containing information about the interface.
string ifDescr = 3;
}
oneof ifType_present {
    // The type of interface.
    int32 ifType = 4;
}
oneof ifMtu_present {
    // The size of the largest packet which can be sent/received
    // on the interface, specified in octets.
    int32 ifMtu = 5;
}
oneof ifPhysAddress_present {
    // The interface's address at its protocol sub-layer.
    bytes ifPhysAddress = 6;
}
}

// TLV 13
// This TLV specifies the periodic reporting of a set of TLVs.
// Class:: Generic
//

message ReportSubscribe {
    oneof interval_present {
        // The periodic time interval (in seconds) at which the device
        // sends the tlvid set of tlvs.
        uint32 interval = 1;
    }
    // The tlvs to be sent on the interval.
    repeated string tlvid = 2;

    oneof intervalHeartBeat_present {
        // The periodic time interval at which the device sends the
        // tlvidHeartBeat set of tlvs.
        uint32 intervalHeartBeat = 3;
    }
    // The tlvs to be sent on the heartbeat interval.
    repeated string tlvidHeartBeat = 4;
}

// TLV 16
// Describes a particular IP address (identified by the index) attached
// to an interface.
// Class:: Generic
//

message IPAddress {
```

```

oneof ipAddressIndex_present {
    // A unique value, greater than zero, for each IP address
    int32 ipAddressIndex = 1;
}
oneof ipAddressAddrType_present {
    // Address type defined as integers :
    // ipv4=1, ipv6=2, ipv4z=3, ipv6z=4, ipv6am=5
    uint32 ipAddressAddrType = 2;
}
oneof ipAddressAddr_present {
    // The IP address
    bytes ipAddressAddr = 3;
}
oneof ipAddressIfIndex_present {
    // Index of the associated interface
    int32 ipAddressIfIndex = 4;
}
oneof ipAddressType_present {
    // IP type defined as integers :
    // unicast=1, anycast=2, broadcast=3
    uint32 ipAddressType = 5;
}
oneof ipAddressOrigin_present {
    // Address origin defined as integers:
    // other=1, manual=2, dhcp=4, linklayer=5, random=6
    uint32 ipAddressOrigin = 6;
}
oneof ipAddressStatus_present {
    // status defined as integers:
    // preferred=1, deprecated=2, invalid=3, inaccessible=4, unknown=5,
    // tentative=6, duplicate=7, optimistic=8
    uint32 ipAddressStatus = 7;
}
reserved 8;
reserved 9;
oneof ipAddressPfxLen_present {
    // The prefix length associated with the IP address.
    uint32 ipAddressPfxLen = 10;
}
}

// TLV 17
// Describes a particular IP route (identified by the index) attached
// to an interface.
// Class:: Generic
//

```

```

message IPRoute {

```

```
oneof inetCidrRouteIndex_present {
    // A unique value, greater than zero, for each route.
    uint32 inetCidrRouteIndex = 1;
}
oneof inetCidrRouteDestType_present {
    // Destination Addresss type defined as integers:
    // ipv4=1, ipv6=2, ipv4z=3, ipv6z=4, ipv6am=5.
    uint32 inetCidrRouteDestType = 2;
}
oneof inetCidrRouteDest_present {
    // IP address of the destination of the route.
    bytes inetCidrRouteDest = 3;
}
oneof inetCidrRoutePfxLen_present {
    // Associated prefix length of the route destination.
    uint32 inetCidrRoutePfxLen = 4;
}
oneof inetCidrRouteNextHopType_present {
    // Next hop Addresss type defined as integers:
    // ipv4=1, ipv6=2, ipv4z=3, ipv6z=4, ipv6am=5.
    uint32 inetCidrRouteNextHopType = 5;
}
oneof inetCidrRouteNextHop_present {
    // IP address of the next hop of the route (device parent).
    bytes inetCidrRouteNextHop = 6;
}
oneof inetCidrRouteIfIndex_present {
    // Index of the associated interface.
    uint32 inetCidrRouteIfIndex = 7;
}
reserved 8;
reserved 9;
reserved 10;
}

// TLV 18
// Contains the current time as maintained on the device.
// For time stamping purposes, this tlv MUST also be sent along with
// every periodic metric report. It MAY contain a POSIX timestamp
// or an ISO 8601 timestamp.
// Class:: Generic
//

message CurrentTime {
    oneof posix_present {
        // posix timestamp.
        uint32 posix = 1;
    }
}
```

```
    oneof iso8601_present {
        // iso 8601 timestamp.
        string iso8601 = 2;
    }
    oneof source_present {
        // time service from:
        // local=1, admin=2, network=3.
        uint32 source = 3;
    }
}

// TLV 21
// For retrieving the RPL Settings on the device.
// Class:: Mesh
//

message RPLSettings {
    oneof ifIndex_present {
        // interface id
        int32 ifIndex = 1;
    }
    oneof enabled_present {
        // whether RPL feature is enabled.
        bool enabled = 2;
    }
    oneof dioIntervalMin_present {
        // min interval of DIO trickle timer in milliseconds.
        uint32 dioIntervalMin = 3;
    }
    oneof dioIntervalMax_present {
        // max interval of DIO trickle timer in milliseconds.
        uint32 dioIntervalMax = 4;
    }
    oneof daoIntervalMin_present {
        // min interval of DAO trickle timer in milliseconds.
        uint32 daoIntervalMin = 5;
    }
    oneof daoIntervalMax_present {
        // max interval of DAO trickle timer in milliseconds.
        uint32 daoIntervalMax = 6;
    }
    oneof mopType_present {
        // mode of operation for RPL. 1: non-storing mode; 2: storing mode.
        uint32 mopType = 7;
    }
}

// TLV 22
```



```
// Contains the total system uptime of the device (seconds).
// Class:: Generic
//

message Uptime {
  oneof sysUpTime_present {
    // uptime info in seconds.
    uint32 sysUpTime = 1;
  }
}

// TLV 23
// The statistics of an interface
// Class:: Generic
//

message InterfaceMetrics {
  oneof ifIndex_present {
    // A unique value, greater than zero, for each interface.
    // Is same as in InterfaceDesc's ifIndex for the same interface.
    int32 ifIndex = 1;
  }
  oneof ifInSpeed_present {
    // The speed at which the incoming packets are received on the
    // interface.
    uint32 ifInSpeed = 2;
  }
  oneof ifOutSpeed_present {
    // The speed at which the outgoing packets are transmitted on
    // the interface.
    uint32 ifOutSpeed = 3;
  }
  oneof ifAdminStatus_present {
    // The desired state of the interface.
    uint32 ifAdminStatus = 4;
  }
  oneof ifOperStatus_present {
    // The current operational state of the interface.
    uint32 ifOperStatus = 5;
  }
  oneof ifLastChange_present {
    // The value of sysUpTime at the time the interface entered its
    // current operational state.
    uint32 ifLastChange = 6;
  }
  oneof ifInOctets_present {
    // The total number of octets received on the interface,
    // including framing characters.
  }
}
```

```
    uint32 ifInOctets = 7;
}
oneof ifOutOctets_present {
    // The total number of octets transmitted out of the interface,
    // including framing characters.
    uint32 ifOutOctets = 8;
}
oneof ifInDiscards_present {
    // The number of inbound packets which were chosen to be discarded
    // even though no errors had been detected to prevent their being
    // deliverable to a higher-layer protocol (application dependant).
    uint32 ifInDiscards = 9;
}
oneof ifInErrors_present {
    // For packet-oriented interfaces, the number of inbound packets
    // that contained errors preventing them from being deliverable to
    // a higher-layer protocol (subset of ifInDiscards).
    uint32 ifInErrors = 10;
}
oneof ifOutDiscards_present {
    // The number of outbound packets which were chosen to be discarded
    // even though no errors had been detected to prevent their being
    // transmitted.
    uint32 ifOutDiscards = 11;
}
oneof ifOutErrors_present {
    // For packet-oriented interfaces, the number of outbound packets
    // that could not be transmitted because of errors.
    uint32 ifOutErrors = 12;
}
}

// TLV 25
// Describes status of each RPL router
// Class:: Mesh
//

message IPRouteRPLMetrics {
    oneof inetCidrRouteIndex_present {
        // refers to a particular index in the IPRoute table.
        int32 inetCidrRouteIndex = 1;
    }
    oneof instanceIndex_present {
        // Corresponding RPL instance of this route.
        int32 instanceIndex = 2;
    }
    oneof rank_present {
        // advertised rank.
    }
}
```

```
    int32 rank = 3;
}
oneof hops_present {
    // Not currently used, future use of hops metric.
    int32 hops = 4;
}
oneof pathEtx_present {
    // advertised path ethx.
    int32 pathEtx = 5;
}
oneof linkEtx_present {
    // next-hop link ETX.
    int32 linkEtx = 6;
}
oneof rssiForward_present {
    // forward RSSI value (relative to the device).
    sint32 rssiForward = 7;
}
oneof rssiReverse_present {
    // reverse RSSI value (relative to the device).
    sint32 rssiReverse = 8;
}
oneof lqiForward_present {
    // forward LQI value.
    int32 lqiForward = 9;
}
oneof lqiReverse_present {
    // reverse LQI value.
    int32 lqiReverse = 10;
}
oneof dagSize_present {
    // nodes count of this pan (number of joined devices).
    uint32 dagSize = 11;
}
reserved 12 to 17;
// forward phy mode value.
PhyModeInfo phyModeForward = 18;
// reverse phy mode value.
PhyModeInfo phyModeReverse = 19;
}

// TLV 30
// Request the device to perform a ping operation to a
// destination address.
// Class:: Generic
//
```

```
message PingRequest {
```

```
    oneof dest_present {
        // IP address to be pinged from the device.
        string dest = 1;
    }
    oneof count_present {
        // number of times to ping.
        uint32 count = 2;
    }
    oneof delay_present {
        // delay between ping in seconds.
        uint32 delay = 3;
    }
}

// TLV 31
// Acquire the current status of the last PingRequest.
// Class:: Generic
//

message PingResponse {
    oneof sent_present {
        // number of packets sent
        uint32 sent = 1;
    }
    oneof received_present {
        // number of packets received
        uint32 received = 2;
    }
    oneof minRtt_present {
        // min round trip time
        uint32 minRtt = 3;
    }
    oneof meanRtt_present {
        // mean round trip time
        uint32 meanRtt = 4;
    }
    oneof maxRtt_present {
        // max round trip time
        uint32 maxRtt = 5;
    }
    oneof stdevRtt_present {
        // standard deviation of the round trip time
        uint32 stdevRtt = 6;
    }
    oneof src_present {
        // source IP address for the ping
        string src = 7;
    }
}
```

```
}

// TLV 32
// Request a device to reboot.
// Class:: Generic
//

message RebootRequest {
  oneof flag_present {
    // 0 : reboot and transfer to designated running image.
    // 1 : reboot and stop at bootloader CLI.
    uint32 flag = 1;
  }
}

// TLV 33
// 802.1x status
// Class:: Mesh
//

message Ieee8021xStatus {
  oneof ifIndex_present {
    // It is RECOMMENDED this be set to 2 for the 6LowPAN interface.
    int32 ifIndex = 1;
  }
  oneof enabled_present {
    // 802.1x enabled or not?
    bool enabled = 2;
  }
  oneof identity_present {
    // subject name of certificate, max len 32
    string identity = 3;
  }
  oneof state_present {
    // state of tls handshake
    uint32 state = 4;
  }
  oneof pmKeyId_present {
    // hash value of pmk, len 16
    bytes pmkId = 5;
  }
  oneof clientCertValid_present {
    // whether client cert is valid
    bool clientCertValid = 6;
  }
  oneof caCertValid_present {
    // whether ca cert is valid
    bool caCertValid = 7;
  }
}
```

```
}
oneof privateKeyValid_present {
    // whether private key of client cert is valid
    bool privateKeyValid = 8;
}
oneof rlyPanid_present {
    // panid of relay node
    uint32 rlyPanid = 9;
}
oneof rlyAddress_present {
    // eui64 address of relay node, len 8
    bytes rlyAddress = 10;
}
oneof rlyLastHeard_present {
    // last heard from relay node in seconds
    uint32 rlyLastHeard = 11;
}
}

// TLV 34
// 802.11i status
// Class:: Mesh
//

message Ieee80211iStatus {
    oneof ifIndex_present {
        // It is RECOMMENDED this be set to 2 for the 6LowPAN interface.
        int32 ifIndex = 1;
    }
    oneof enabled_present {
        // 802.11i is eabled or not
        bool enabled = 2;
    }
    oneof pmkId_present {
        // hash value of pmk, len 16
        bytes pmkId = 3;
    }
    oneof ptkId_present {
        // hash value of ptk, len 16
        bytes ptkId = 4;
    }
    oneof gtkIndex_present {
        // index of gtk
        int32 gtkIndex = 5;
    }
    oneof gtkAllFresh_present {
        // whether all gtk are fresh
        bool gtkAllFresh = 6;
    }
}
```

```
}
// list of hash value for each gtk, hash len 8, max repeat 4
repeated bytes gtkList = 7;
// list of lifetime for each gtk, hash len 8, max repeat 4
repeated uint32 gtkLifetimes = 8;
oneof authAddress_present {
    // eui64 address of authenticate node
    bytes authAddress = 9;
}
}

message PhyModeInfo {
    oneof phyMode_present {
        // phy operating mode value (as defined in section 5.2 of
        // PHYWG Wi-SUN PHY Technical Specification - Amendment 1VA8)
        uint32 phyMode = 1;
    }
    oneof txPower_present {
        // transmit power value in dbm.
        int32 txPower = 2;
    }
}

// TLV 35
// Configuration of WPAN-specific interface settings
// Class:: Mesh
//

message WPANStatus {
    oneof ifIndex_present {
        // It is RECOMMENDED this be set to 2 for the 6LowPAN interface.
        int32 ifIndex = 1;
    }
    oneof SSID_present {
        // Max len 32 (Wi-SUN NetName).
        bytes SSID = 2;
    }
    oneof panid_present {
        uint32 panid = 3;
    }
    reserved 4;

    oneof dot1xEnabled_present {
        // Is dot1x enabled?
        bool dot1xEnabled = 5;
    }
    oneof securityLevel_present {
        // Security level

```

```
    uint32 securityLevel = 6;
}
oneof rank_present {
    uint32 rank = 7;
}
oneof beaconValid_present {
    // Is beacon valid (where invalid means receipt has
    // timed-out/contact with PAN was lost)? (PC frame for Wi-SUN)
    bool beaconValid = 8;
}
oneof beaconVersion_present {
    // Beacon version (Wi-SUN PAN version).
    uint32 beaconVersion = 9;
}
oneof beaconAge_present {
    // Last heard beacon message in seconds (PC frame for Wi-SUN).
    uint32 beaconAge = 10;
}
oneof txPower_present {
    // Transmit power value in dbm
    int32 txPower = 11;
}
oneof dagSize_present {
    // Count of nodes joined to this PAN
    uint32 dagSize = 12;
}
oneof metric_present {
    // Metric to border router (Wi-SUN Routing Cost)
    uint32 metric = 13;
}
oneof lastChanged_present {
    // seconds since last PAN change (PAN ID change).
    uint32 lastChanged = 14;
}
oneof lastChangedReason_present {
    // Reason for last PAN change:
    // -1 == unknown,
    // 0 == mesh initializing,
    // 1 == mesh connectivity lost,
    // 2 == GTK timeout,
    // 3 == default route lost,
    // 4 == migrated to better PAN,
    // 5 == reserved.
    uint32 lastChangedReason = 15;
}
oneof demoModeEnabled_present {
    // Is demo mode enabled?
    bool demoModeEnabled = 16;
}
```



```
}
oneof txFec_present {
    // Is FEC enabled?
    bool txFec = 17;
}
oneof phyMode_present {
    // Phy operating mode value (as defined in section 5.2 of
    // PHYWG Wi-SUN PHY Technical Specification - Amendment 1VA8)
    uint32 phyMode = 18;
}
reserved 19;
// Multi phy mode and transmit power value for adaptive modulation.
repeated PhyModeInfo phyModeList = 20;
}

// TLV 36
// Status of DHCP6 client
// Class:: Generic
//

message DHCP6ClientStatus {
    oneof ifIndex_present {
        // It is RECOMMENDED this be set to 2 for the 6LowPAN interface
        int32 ifIndex = 1;
    }
    oneof ianaIAID_present {
        // TA ID value.
        uint32 ianaIAID = 2;
    }
    oneof ianaT1_present {
        // T1 value.
        uint32 ianaT1 = 3;
    }
    oneof ianaT2_present {
        // T2 value.
        uint32 ianaT2 = 4;
    }
}

// TLV 42
// Configure device reporting settings.
// Class:: Generic
//

message NMSSettings {
    oneof regIntervalMin_present {
        // Min interval of register trickle timer in seconds.
        uint32 regIntervalMin = 1;
    }
}
```

```
    }
    oneof regIntervalMax_present {
        // Max interval of register trickle timer in seconds.
        uint32 regIntervalMax = 2;
    }
    reserved 3 to 6;
}

// TLV 43
// Registration status to NMS.
// NMS uses this TLV to record the reason for the registration
// operation as recorded in the lastRegReason field of the TLV.
// Class:: Generic
//

message NMSStatus {
    oneof registered_present {
        // True if device is registerd with NMS.
        bool registered = 1;
    }
    oneof NMSAddr_present {
        // IPv6 address of NMS.
        bytes NMSAddr = 2;
    }
    oneof NMSAddrOrigin_present {
        // Mechanism used to get NMS's IPv6 address.
        // (fixed/DHCPv6/redirect by TLV6 ... values).
        uint32 NMSAddrOrigin = 3;
    }
    oneof lastReg_present {
        // Time since last succesful registration in seconds.
        uint32 lastReg = 4;
    }
    oneof lastRegReason_present {
        // Reason for last registration:
        // 1: coldstart,
        // 2: administrative,
        // 3: IP address changed,
        // 4: NMS changed,
        // 5: NMS redirect,
        // 6: NMS error,
        // 7: IDevID, LDevID, or NMS certificate changed,
        // 8: outage recovery.
        uint32 lastRegReason = 5;
    }
    oneof nextReg_present {
        // Time to next registration attempt in seconds.
        uint32 nextReg = 6;
    }
}
```

```
    }
    oneof NMSCertValid_present {
        // True if NMS certificate is valid.
        bool NMSCertValid = 7;
    }
}

// TLV 47
// Device settings for 802.1x
// Class:: Mesh
//

message Ieee8021xSettings {
    oneof ifIndex_present {
        // It is RECOMMENDED this be set to 2 for the 6LowPAN interface.
        int32 ifIndex = 1;
    }
    oneof secMode_present {
        // Security mode, non-security or security (Security Level from
        // Aux Security Header of IEEE 802.15.4-2020).
        uint32 secMode = 2;
    }
    oneof authIntervalMin_present {
        // Min interval of 802.1x trickle timer in seconds.
        uint32 authIntervalMin = 3;
    }
    oneof authIntervalMax_present {
        // Max interval of 802.1x trickle timer in seconds.
        uint32 authIntervalMax = 4;
    }
    oneof immediate_present {
        // True == do 802.1x authentication immediately,
        // False == do 802.1x authentication at next authentication
        // interval.
        bool immediate = 5;
    }
}

// TLV 48
// Statistic of 802.15.4 beacon packets
// Class:: Mesh
//

message Ieee802154BeaconStats {
    oneof ifIndex_present {
        // It is RECOMMENDED this be set to 2 for the 6LowPAN interface.
        int32 ifIndex = 1;
    }
}
```

```
oneof inFrames_present {
    // Count of received beacon.
    uint32 inFrames = 10;
}
oneof inFramesBeaconPAS_present {
    // Count of received PAS beacon.
    uint32 inFramesBeaconPAS = 11;
}
oneof inFramesBeaconPA_present {
    // Count of received PA beacon.
    uint32 inFramesBeaconPA = 12;
}
oneof inFramesBeaconPCS_present {
    // Count of received PCS beacon.
    uint32 inFramesBeaconPCS = 13;
}
oneof inFramesBeaconPC_present {
    // Count of received PC beacon.
    uint32 inFramesBeaconPC = 14;
}
oneof outFrames_present {
    // Count of all sent out beacon.
    uint32 outFrames = 20;
}
oneof outFramesBeaconPAS_present {
    // Count of sent out PAS beacon.
    uint32 outFramesBeaconPAS = 21;
}
oneof outFramesBeaconPA_present {
    // Count of sent out PA beacon.
    uint32 outFramesBeaconPA = 22;
}
oneof outFramesBeaconPCS_present {
    // Count of sent out PCS beacon.
    uint32 outFramesBeaconPCS = 23;
}
oneof outFramesBeaconPC_present {
    // Count of sent out PC beacon.
    uint32 outFramesBeaconPC = 24;
}
}

// TLV 53
// Indicates RPL instance information
// Class:: Mesh
//
```

```
message RPLInstance {
```

```
oneof instanceIndex_present {
    // Index for instance.
    int32 instanceIndex = 1;
}
oneof instanceId_present {
    // Instance id.
    int32 instanceId = 2;
}
oneof doDagId_present {
    // DODAG id, len 16.
    bytes doDagId = 3;
}
oneof doDagVersionNumber_present {
    // DODAG version number of instance.
    int32 doDagVersionNumber = 4;
}
oneof rank_present {
    // Rank value.
    int32 rank = 5;
}
oneof parentCount_present {
    // Count of available parents (Wi-SUN candidate parent set).
    int32 parentCount = 6;
}
oneof dagSize_present {
    // Node count of this DODAG.
    uint32 dagSize = 7;
}
// Max repeat 3.
repeated RPLParent parents = 8;
// Max repeat 3.
repeated RPLParent candidates = 9;
}

message RPLParent {
    oneof parentIndex_present {
        // This parent's index in the RPLParent table.
        int32 parentIndex = 1;
    }
    oneof instanceIndex_present {
        // A particular index in the RPLInstance table that this
        // parent underlies.
        int32 instanceIndex = 2;
    }
    oneof routeIndex_present {
        // A particular index in the IPRoute table that this
        // parent underlies.
        int32 routeIndex = 3;
    }
}
```

```
}
oneof ipv6AddressLocal_present {
    // IPv6 linklocal address.
    bytes ipv6AddressLocal = 4;
}
oneof ipv6AddressGlobal_present {
    // IPv6 global address.
    bytes ipv6AddressGlobal = 5;
}
oneof doDagVersionNumber_present {
    // DODAG version number if RPL parent.
    uint32 doDagVersionNumber = 6;
}
oneof pathEtx_present {
    // The parent's ETX back to the root.
    int32 pathEtx = 7;
}
oneof linkEtx_present {
    // The node's ETX to its next-hop.
    int32 linkEtx = 8;
}
oneof rssiForward_present {
    // Forward RSSI value.
    sint32 rssiForward = 9;
}
oneof rssiReverse_present {
    // Reverse RSSI value.
    sint32 rssiReverse = 10;
}
oneof lqiForward_present {
    // Forward LQI value.
    int32 lqiForward = 11;
}
oneof lqiReverse_present {
    // Reverse LQI value.
    int32 lqiReverse = 12;
}
oneof hops_present {
    // parent's hop value.
    int32 hops = 13;
}
}

// Groups in CSMP provide a mechanism to realize application
// layer multicast in the network. A group is uniquely defined by
// a type, id pair. Membership within a group type is exclusive,
// i.e., a device can be a member of only one group-id within a
// group-type. However, a device can be a member of more than one
```

```
// group of different group-types.

// A device is informed about its membership to a group using the
// GroupAssign TLV. On their very first boot, devices do not
// belong to any group. A device is added to a group by sending a
// GroupAssign TLV to the device. Receipt of a GroupAssign TLV
// replaces any existing group assignments. GroupAssign may occur
// either by direct unicast to a device or in the registration
// response from the NMS to the device. Note that a GroupAssign
// should not be sent over multicast, because it would possibly
// cause some group members to change and some group members not
// to change.

// Group membership information MUST be stored in persistent
// storage so that once a device has been assigned any group it is
// remembered across reboots. A device will only process multicast
// messages containing a GroupMatch TLV if the device belongs to a
// group specified by the GroupMatch TLV.

// TLV 55
// Class:: Generic
//

message GroupAssign {
    oneof type_present {
        uint32 type = 1;
    }
    oneof id_present {
        uint32 id = 2;
    }
}

// TLV 57
// Class:: Generic
//

message GroupMatch {
    oneof type_present {
        uint32 type = 1;
    }
    oneof id_present {
        uint32 id = 2;
    }
}

// TLV 58
// Class:: Generic
// GET to a device for this TLV MAY illicit a response with one or
```

```
// more GroupInfo TLVs.
//

message GroupInfo {
  oneof type_present {
    uint32 type = 1;
  }
  oneof id_present {
    uint32 id = 2;
  }
}

message LowpanMacCounters {
  oneof inFrames_present {
    // Count of all received frames.
    uint32 inFrames = 1;
  }
  oneof inFramesBeacon_present {
    // Count of received beacon frames (Note: Wi-SUN does
    // not use Beacon frames).
    uint32 inFramesBeacon = 2;
  }
  oneof inFramesData_present {
    // Count of received data frames.
    uint32 inFramesData = 3;
  }
  oneof inFramesAck_present {
    // Count of received ack frames.
    uint32 inFramesAck = 4;
  }
  oneof inFramesCmd_present {
    // Count of received command frames.
    uint32 inFramesCmd = 5;
  }
  oneof inFramesAsync_present {
    // Count of received async frames.
    uint32 inFramesAsync = 6;
  }
  oneof inFramesBcast_present {
    // Count of received broadcast frames.
    uint32 inFramesBcast = 7;
  }
  oneof inFramesUcast_present {
    // Count of received unicast frames.
    uint32 inFramesUcast = 8;
  }
  oneof outFrames_present {
    // Count of all sent out frames.
  }
}
```



```
    uint32 outFrames = 9;
}
oneof outFramesBeacon_present {
    // Count of sent out beacon frames.
    uint32 outFramesBeacon = 10;
}
oneof outFramesData_present {
    // Count of sent out data frames.
    uint32 outFramesData = 11;
}
oneof outFramesAck_present {
    // Count of sent out ack frames.
    uint32 outFramesAck = 12;
}
oneof outFramesCmd_present {
    // Count of sent out command frames.
    uint32 outFramesCmd = 13;
}
oneof outFramesAsync_present {
    // Count of sent out async frames.
    uint32 outFramesAsync = 14;
}
oneof outFramesBcast_present {
    // Count of recesent outived broadcast frames.
    uint32 outFramesBcast = 15;
}
oneof outFramesUcast_present {
    // Count of sent out unicast frames.
    uint32 outFramesUcast = 16;
}
}

// TLV 62
// Statistic of lowpan mac layer
// Class:: Mesh
//

message LowpanMacStats {
    // Total counter of lowpan mac layer.
    LowpanMacCounters total = 1;
    // RF counter of lowpan mac layer.
    LowpanMacCounters rf = 2;
    reserved 3;
}

// TLV 63
// Configuration of RF PHY
// Class:: Mesh
```

```
//  
  
message LowpanPhySettings {  
  oneof lowpanRF_present {  
    // 1 == enable lowpan RF, 0 == disable.  
    uint32 lowpanRF = 1;  
  }  
  reserved 2;  
}  
  
// TLV 65 - 75 are for firmware upgrade.  
// Usage is detailed in main body of the CSMP specification.  
//  
message HardwareInfo {  
  oneof hwId_present {  
    // Hardware information, max len 32.  
    string hwId = 1;  
  }  
  oneof vendorHwId_present {  
    // Sub hardware information, max len 32.  
    string vendorHwId = 2;  
  }  
}  
  
// TLV 65  
// Start to transfer firmware  
// Class:: Generic  
//  
  
message TransferRequest {  
  // Hardware information.  
  HardwareInfo hwInfo = 1;  
  oneof fileHash_present {  
    // SHA256 hash value of image file, len 32.  
    bytes fileHash = 2;  
  }  
  oneof fileName_present {  
    // Name of image file, max len 128.  
    string fileName = 3;  
  }  
  oneof version_present {  
    // Version number, max len 32.  
    string version = 4;  
  }  
  oneof fileSize_present {  
    // Total size of image file.  
    uint32 fileSize = 5;  
  }  
}
```

```
    oneof blockSize_present {
        // Block size of image file.
        uint32 blockSize = 6;
    }
    reserved 7 to 11;
}

// TLV 67
// Class:: Generic
//

message ImageBlock {
    oneof fileHash_present {
        // SHA256 hash value of image file, len 32.
        bytes fileHash = 1;
    }
    oneof blockNum_present {
        // Block number 0, 1, 2, etc.
        uint32 blockNum = 2;
    }
    oneof blockData_present {
        // Block context, max len 1024.
        bytes blockData = 4;
    }
}

// TLV 68
// Firmware load request
// Class:: Generic
//

message LoadRequest {
    oneof fileHash_present {
        // SHA256 hash value of image file, len 32.
        bytes fileHash = 1;
    }
    oneof loadTime_present {
        // UTC time to load image, set to 1 to load immediately.
        uint32 loadTime = 2;
    }
}

// TLV 69
// Firmware cancel load request
// Class:: Generic
//

message CancelLoadRequest {
```

```
    oneof fileHash_present {
        // SHA256 hash value of image file, len 32.
        bytes fileHash = 1;
    }
}

// TLV 70
// Set firmware to backup slot
// Class:: Generic
//

message SetBackupRequest {
    oneof fileHash_present {
        // SHA256 hash value of image file, len 32.
        bytes fileHash = 1;
    }
}

/* ResponseCodes

enum {
    // The request was successfully processed.
    OK = 0;
    // Device hardware type does not match
    // the request's hardware type.
    INCOMPATIBLE_HW = 1;
    // Image operation cannot be completed,
    // device only has partial image.
    IMAGE_INCOMPLETE = 2;
    // File hash does not match any image
    // available on the device.
    UNKNOWN_HASH = 3;
    // Image download is denied, filesize
    // of the new image is too large.
    FILE_SIZE_TOO_BIG = 4;
    // Image signature check failed.
    SIGNATURE_FAILED = 5;
    // Invalid request.
    INVALID_REQ = 6;
    // Invalid image block size.
    INVALID_BLOCK_SIZE = 7;
    // Request cannot be processed,
    // conflict with a pending device reboot.
    PENDING_REBOOT = 8;
    // Cancel reboot request cannot be processed,
    // image is already running.
    IMAGE_RUNNING = 9;
}
```

```
*/

// TLV 71
// Response for TLV 65 TransferRequest
// Class:: Generic
//

message TransferResponse {
  oneof fileHash_present {
    // SHA256 hash value of image file, len 32.
    bytes fileHash = 1;
  }
  oneof response_present {
    // Refer to ResponseCodes.
    uint32 response = 2;
  }
}

// TLV 72
// Response for TLV 68 LoadRequest
// Class:: Generic
//

message LoadResponse {
  oneof fileHash_present {
    // SHA256 hash value of image file, len 32.
    bytes fileHash = 1;
  }
  oneof response_present {
    // refer to ResponseCodes.
    uint32 response = 2;
  }
  oneof loadTime_present {
    // UTC time to load image.
    uint32 loadTime = 3;
  }
}

// TLV 73
// Response for TLV 69 CancelLoadRequest
// Class:: Generic
//

message CancelLoadResponse {
  oneof fileHash_present {
    // SHA256 hash value of image file, len 32.
    bytes fileHash = 1;
  }
}
```

```
    oneof response_present {
        // Refer to ResponseCodes.
        uint32 response = 2;
    }
}

// TLV 74
// Class:: Generic
//

message SetBackupResponse {
    oneof fileHash_present {
        // SHA256 hash value of image file, len 32.
        bytes fileHash = 1;
    }
    oneof response_present {
        // Refer to ResponseCodes.
        uint32 response = 2;
    }
}

// TLV 75
// Image information
// Class:: Generic
//

message FirmwareImageInfo {
    oneof index_present {
        uint32 index = 1;
    }
    oneof fileHash_present {
        // SHA256 hash value of image file, len 32.
        bytes fileHash = 2;
    }
    oneof fileName_present {
        // name of image file, max len 128.
        string fileName = 3;
    }
    oneof version_present {
        // version number, max len 32.
        string version = 4;
    }
    oneof fileSize_present {
        // total size of image file.
        uint32 fileSize = 5;
    }
    oneof blockSize_present {
        // block size of image file.

```

```
    uint32 blockSize = 6;
}
oneof bitmap_present {
    // bitmap of image file, max len 128. Big endian.
    // 1 means block was received, 0 means block was not received.
    bytes bitmap = 7;
}
oneof isDefault_present {
    // True if default image.
    bool isDefault = 8;
}
oneof isRunning_present {
    // True if running image.
    bool isRunning = 9;
}
oneof loadTime_present {
    // UTC time to load. 1 means load immediately.
    uint32 loadTime = 10;
}
// hardware information
HardwareInfo hwInfo = 11;
oneof bitmapOffset_present {
    // When present, MUST be set to indicate the start block
    // number of bitmap. Block numbering is 0 based.
    uint32 bitmapOffset = 12;
}
reserved 13 to 15;
}

// TLV 76
// Class:: Generic
//

message SignatureValidity {
    oneof notBefore_present {
        // Posix time.
        uint32 notBefore = 1;
    }
    oneof notAfter_present {
        // Posix time.
        uint32 notAfter = 2;
    }
}

// TLV 77
// Class:: Generic
//
```

```
message Signature {
  oneof value_present {
    bytes value = 1;
  }
}

// TLV 79
// Configuration of signature check settings about message from NMS
// Class:: Generic
//

message SignatureSettings {
  oneof reqSignedPost_present {
    // Check signature in POST message from NMS?
    bool reqSignedPost = 1;
  }
  oneof reqValidCheckPost_present {
    // Time valid check in POST message from NMS?
    bool reqValidCheckPost = 2;
  }
  oneof reqTimeSyncPost_present {
    // Return fail when node doesn't have global time in
    // POST message from NMS?
    bool reqTimeSyncPost = 3;
  }
  oneof reqSecLocalPost_present {
    // Check signature in POST message from console?
    bool reqSecLocalPost = 4;
  }
  oneof reqSignedResp_present {
    // Check signature in response message from NMS
    // during registration?
    bool reqSignedResp = 5;
  }
  oneof reqValidCheckResp_present {
    // Time valid check in response message from NMS
    // during registration?
    bool reqValidCheckResp = 6;
  }
  oneof reqTimeSyncResp_present {
    // Return fail when node doesn't have global time in
    // response message from NMS during registration?
    bool reqTimeSyncResp = 7;
  }
  oneof reqSecLocalResp_present {
    // Check signature in response message from console
    // during registration?
    bool reqSecLocalResp = 8;
  }
}
```



```
    }  
    oneof cert_present {  
        // Certificate context, used to change NMS's  
        // certificate, max len 512.  
        bytes cert = 9;  
    }  
}
```

```
message HardwareResetCount {  
    oneof total_present {  
        uint32 total = 1;  
    }  
    oneof externalReset_present {  
        // External reset reason.  
        uint32 externalReset = 2;  
    }  
    oneof powerOnReset_present {  
        // Power on reset reason.  
        uint32 powerOnReset = 3;  
    }  
}
```

```
message SoftwareResetCount {  
    oneof total_present {  
        uint32 total = 1;  
    }  
    oneof FWLoadReset_present {  
        // Firmware reload.  
        uint32 FWLoadReset = 2;  
    }  
    oneof CSMPRebootReset_present {  
        // Reload forced by CSMP TLV.  
        uint32 CSMPRebootReset = 3;  
    }  
    oneof vendorProgramReset_present {  
        // Reload forced by vendor's APP.  
        uint32 vendorProgramReset = 4;  
    }  
    oneof cfgLoadReset_present {  
        // Reload config file.  
        uint32 cfgLoadReset = 5;  
    }  
}
```

```
message ExceptionResetCount {  
    oneof total_present {  
        uint32 total = 1;  
    }  
}
```

```
    oneof IWDGReset_present {
        // Watchdog forced reset.
        uint32 IWDGReset = 2;
    }
    oneof cstackOverflowReset_present {
        // Stack over flow reset.
        uint32 cstackOverflowReset = 3;
    }
    oneof EPFReset_present {
        // GPIO reset.
        uint32 EPFReset = 4;
    }
}

// TLV 86
// Count of all types of reset in the system, include hardware reset,
// software reset and exception reset.
// Class:: Generic
//

message SysResetStats {
    oneof total_present {
        // Reset counters.
        uint32 total = 1;
    }
    HardwareResetCount hardwareReset = 2;
    SoftwareResetCount softwareReset = 3;
    ExceptionResetCount exceptionReset = 4;
}

// TLV 124
// Status of device network module, similar as netstat command in linux
// Class:: Generic
//

message NetStat {
    oneof sessionIndex_present {
        // Session index.
        int32 sessionIndex = 1;
    }
    oneof protocol_present {
        // 6 TCP,
        // 17 UDP.
        uint32 protocol = 2;
    }
    oneof localAddress_present {
        //IPv4 or IPv6 local address.
        bytes localAddress = 3;
    }
}
```

```
}
oneof localPort_present {
    // Local port number.
    uint32 localPort = 4;
}
oneof peerAddress_present {
    // IPv4 or IPv6 peer address.
    bytes peerAddress = 5;
}
oneof peerPort_present {
    // Peer port number.
    uint32 peerPort = 6;
}
oneof state_present {
    uint32 state = 7;
}
oneof role_present {
    // 1 - server/incoming,
    // 2 - client/outgoing.
    uint32 role = 8;
}
}

// TLV 141
// Indicate the network role of device
// Class:: Generic
//

message NetworkRole {
    // 0 - SYSTEM_DEFAULT,
    // 1 - TRANSIT_NODE,
    // 2 - LEAF_NODE.
    uint32 preference = 1;
}

message CertInfoEntry {
    oneof type_present {
        // 1 - FND public key,
        // 2 - 802.1x CA,
        // 3 - 802.1x public key,
        // 4 - iDevID public key.
        uint32 type = 1;
    }
    oneof certSubj_present {
        // Certificate subject name, max length 128.
        string certSubj = 2;
    }
    oneof certValidNotBefore_present {
```

```
    // Not before of valid time, max length 16.
    string certValidNotBefore = 3;
  }
  oneof certValidNotAfter_present {
    // Not after of valid time, max length 16.
    string certValidNotAfter = 4;
  }
  oneof certFingerprint_present {
    // Certificate finger print, max length 20.
    bytes certFingerprint = 5;
  }
}

// TLV 172
// Device Certificate Bundle TLV.
//
message CertBundle {
  // Max repeat is 5.
  repeated CertInfoEntry certInfo = 1;
}

// TLV 241
// Statistic of MPL packet
// Class:: Mesh
//

message MplStats {
  oneof dataSent_present {
    // Count of sent data packets.
    uint32 dataSent = 1;
  }
  oneof dataReceived_present {
    // Count of received data packets.
    uint32 dataReceived = 2;
  }
  oneof dataError_present {
    // Count of error data packets.
    uint32 dataError = 3;
  }
  oneof dataSentDuplicate_present {
    // Count of duplicate sent data packets.
    uint32 dataSentDuplicate = 4;
  }
  oneof dataReceivedDuplicate_present {
    // Count of duplicate received data packets.
    uint32 dataReceivedDuplicate = 5;
  }
  oneof controlSent_present {
```

```
    // Count of send control packets.
    uint32 controlSent = 6;
}
oneof controlReceived_present {
    // Count of received control packets.
    uint32 controlReceived = 7;
}
oneof controlError_present {
    // Count of error control packets.
    uint32 controlError = 8;
}
}

// TLV 242
// Reset statistic of MPL packet
// Class:: Mesh
//
```

```
message MplReset {
    oneof stats_present {
        // True means reset MPL statistics.
        bool stats = 8;
    }
}
```

```
// TLV 313
// Statistics for RPL messages
// Class:: Mesh
//
```

```
message RPLStats {
    oneof inFramesDIS_present {
        // Count of received DIS packets.
        uint32 inFramesDIS = 1;
    }
    oneof inFramesDIO_present {
        // Count of received DIO packets.
        uint32 inFramesDIO = 2;
    }
    oneof inFramesDAO_present {
        // Count of received DAO packets.
        uint32 inFramesDAO = 3;
    }
    oneof outFramesDIS_present {
        // Count of sent DIS packets.
        uint32 outFramesDIS = 4;
    }
    oneof outFramesDIO_present {
```

```
    // Count of sent DIO packets.
    uint32 outFramesDIO = 5;
}
oneof outFramesDAO_present {
    // Count of sent DAO packets.
    uint32 outFramesDAO = 6;
}
oneof outFramesNoPathDAO_present {
    // Count of sent no-path DAO packets.
    uint32 outFramesNoPathDAO = 7;
}
oneof outFramesNS_present {
    // Count of sent neighbor solicit packets.
    uint32 outFramesNS = 8;
}
}

// TLV 314
// Statistics for DHCPv6 messages
// Class:: Generic
//

message DHCP6Stats {
    oneof clientFramesSolicit_present {
        // Count of sent solicit packets.
        uint32 clientFramesSolicit = 1;
    }
    oneof clientFramesAdvertise_present {
        // Count of sent advertise packets.
        uint32 clientFramesAdvertise = 2;
    }
    oneof clientFramesRequest_present {
        // Count of sent request packets.
        uint32 clientFramesRequest = 3;
    }
    oneof clientFramesReply_present {
        // Count of sent reply packets.
        uint32 clientFramesReply = 4;
    }
    oneof relayFramesForward_present {
        // Count of DHCP relay packets forwarded by node.
        uint32 relayFramesForward = 5;
    }
    oneof relayFramesReply_present {
        // Count of DHCP relay packets relayed by node.
        uint32 relayFramesReply = 6;
    }
}
```

3.3.3. Large Requests

A single CSMP TLV MUST NOT be larger than the space available in a single CoAP request message payload, minus the space occupied by mandatory TLVs. CSMP requests containing large TLVs or many TLVs may exceed available space within a CoAP request / UDP datagram.

If a POST request is larger than the UDP MTU, the request MUST be split into multiple POST requests with the TLVs spread across the message bodies. The server MUST be prepared to handle the TLVs in any order.

If a GET request exceeds the UDP MTU because the max length of the "q" option is exceeded, the request MUST be split into multiple GET requests, each with a subset of the query option.

The GET response from a server may not be able to fit all requested TLVs into the response. The server will respond with only the TLVs it is able to fit within the message body. A client SHOULD issue additional GET requests to obtain the missing TLVs.

Recommended network MTU will be deployment / technology dependent. For example, an MTU of 1024 is often used for large scale IEEE 802.15.4 mesh networks.

3.4. CSMP Security Model

The NMS signs outgoing device messaging. Devices verify the signature to confirm source and integrity of incoming NMS messages. NMS-Device trust is established with an NMS certificate/public key programmed into the device at time of manufacture. Signing TLVs included in the message payload enable signature verification by a device. The Signing TLVs are:

1. Signature TLV. When included, the Signature TLV MUST be the last TLV in a message payload. The signature is calculated over the first byte of the message payload up to but not including the Signature TLV itself. Unless otherwise specified, the signature MUST be calculated as ECDSA with SHA-256 signature cipher using the signer's (NMS) private key. If the message signature is incorrect, the device MUST ignore the message.
2. SignatureValidity TLV. The SignatureValidity TLV defines the validity period for the message. The SignatureValidity TLV MUST be included when the Signature TLV is included. If the message is received outside the defined validity period, the device MUST ignore the message.

If either of the Signing TLVs are missing from a message payload, the device MUST ignore the message.

Additional layer 2, 3, or 4 security mechanisms may be utilized to meet the requirements of specific deployment models (Wi-SUN layer 2 security, VPN at layer 3, DTLS at layer 4, etc.). Details of these additional security mechanisms are out of scope of this specification.

3.4.1. Signature Exemption

For situations in which a request payload signature adds overhead without improving security, the Signing TLVs may be elided for certain payloads. For example, the overhead of signing each block of a firmware update may be unnecessary as a full image integrity check is performed over the entire file and reported to the NMS.

The signature exemptible TLVs are:

1. ImageBlock
2. DescriptionRequest

The NMS MAY elide the Signing TLVs provided the request body contains only exemptible TLVs. Otherwise, the Signing TLVs MUST be included.

A device MAY accept incoming message payloads without Signing TLVs provided the payload contains only exemptible TLVs.

3.5. Device Groups

CSMP groups are used to support multicast messaging to devices.

A group is uniquely defined by a group-type/group-id pair. A device MAY be a member of multiple group-types, but MUST be a member of only one group-id within a group-type. A device MUST support membership in at least two group types.

The NMS assigns a device to a group using the GroupAssign TLV. On initial boot, a device has no group assignments. To be assigned to a device group, a GroupAssign TLV MUST be sent to the device either by a POST request from the NMS or within the response to the device's registration request to the NMS.

The NMS removes a device from a group by POST-ing a GroupEvict TLV to the device.

If a device's group assignment is changed at the NMS, upon receipt of the next metrics report from the device, the NMS MUST POST a new GroupAssign TLV to the device.

Group assignments are not additive. Assignments MUST be replaced upon receipt of a subsequent GroupAssign TLV.

A GroupAssign TLV MUST NOT be sent within a multicast message.

A GroupEvict TLV MUST NOT be sent within a multicast message.

Devices MUST maintain group assignments in durable storage (across power cyclings / reboots).

CSMP multicast messages MUST contain a GroupMatch TLV. Upon receipt of a multicast CSMP message:

1. A device MUST process the message if the contained GroupMatch TLV matches a group to which the device is assigned.
2. A device MUST ignore the message if the message does not contain a GroupMatch TLV.
3. A device MUST ignore the message if the GroupMatch TLV does not match a device group assignment.

3.5.1. Reserved Group Types

Group type 1 is reserved for configuration.

Group type 2 is reserved for firmware.

3.6. Device TLV Processing Order

A device processes message payload TLVs in the following order:

1. If present, the Signature and SignatureValidity TLVs MUST be processed first.
2. If present, the GroupMatch TLV MUST be processed next.
3. The remaining payload TLVs MUST be processed in the order they appear in the payload.
4. TLVs within a payload SHOULD NOT be duplicated. In the case of a duplicate TLV, the last payload instance of TLV MUST be used.

5. The index field of a TLV table entry is used to determine uniqueness of the TLV. TLVs with identical index values MUST be considered to be duplicates (table entry TLVs are identified in Section 3.2.3).
6. TLV specific error handling is described in the OpenAPI definitions.

4. Functional Description

This section describes the major operational flows of the CSMP protocol.

4.1. Device Lifecycle States

For understanding of CSMP device behavior, it is helpful to consider the NMS' view of device states and state transitions (presented below).

The NMS views a device as transitioning through the following states:

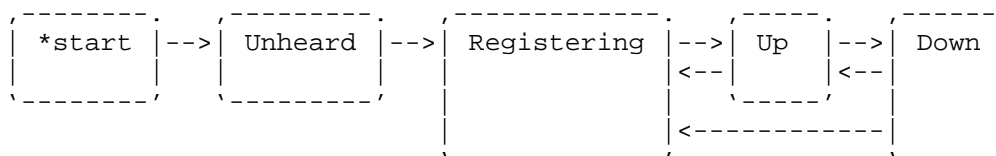


Figure 1: NMS View of Device State

1. A device pre-populated into the NMS prior to deployment exists in the Unheard state.
2. Upon receipt of a device registration request, the NMS records the device's presence on the network and the device enters the Registering state. The NMS responds with a Registration ACK payload containing configuration for the device.
3. The device transitions to the Up state upon NMS receipt of a device metrics report (indicating device configuration was successful). This is the normal operating state of a healthy device.
4. If subsequent metrics reports are lost (poor network conditions, device failure, etc.), the device transitions to the Down state. NMS receipt of a new metrics report transitions the device back to the Up state.

5. From the Up or Down state, a device may issue new registration requests due to a variety of reasons discussed below.

4.2. NMS Discovery

A device requires the <nms-base-url> of its NMS. Acquisition of the NMS URL may be accomplished via a variety of means including a DHCP option, pre-deployment administrative configuration setting, etc. The specific mechanism to be used is beyond the scope of this specification.

For devices using DHCPv6 address assignment, a device MAY request DHCPv6 option 26484 sub-option 1 to obtain the URL of its NMS.

4.3. Device Registration and Configuration

Registration is the messaging flow via which a device announces its entry onto the network and provides a means for the NMS to push configuration information to the device.

A device registers with an NMS by issuing a registration request to an NMS. The NMS subsequently responds to reject or accept the registration, with device configuration included in a successful registration response. A device issues a registration request for a variety of reasons:

1. A device MUST register when the device reboots (power cycled or receipt of RebootRequest TLV from the NMS).
2. A device MUST register when its IP address has changed (usually indicating a network re-join).
3. A device MUST register if its mesh parent has changed (mesh networks only).
4. A device MUST register if it detects the NMS IP address has changed.
5. A device MUST register upon receipt of NMSRedirectRequest TLV from the NMS. This can be caused by the removal of a device from the NMS inventory but the device continues to communicate with a Session ID now unknown to the NMS.

A device and NMS implement the registration messaging flow depicted in Figure 2.

Figure only available as SVG (PDF and HTML)

Figure 2: Device Registration, Configuration, and Metrics

4.3.1. Device Registration Request

A device MUST implement two configurable parameters used to control the registration process, initially set at manufacture time, and MUST be maintained in durable storage.

1. `tIntervalMin` defaults to 300 seconds (5 minutes). Also configurable via TLV 42/`regIntervalMin` field.
2. `tIntervalMax` defaults to 3600 seconds (1 hour). Also configurable via TLV 42/`regIntervalMax` field.

A device MUST issue registration requests using the following algorithm.

```
Set tInterval = tIntervalMin.  
Wait an initial period between 0 and tInterval seconds.  
Do {  
  1. Wait tBackoff seconds, where tBackoff is a random  
     interval between tInterval/2 and tInterval seconds.  
     A tBackoff value in the latter half of the interval  
     ensures a minimum time between successive registration  
     attempts.  
  2. Send new CoAP CON POST request to NMS <nms-base-url>/r.  
     The message payload MUST contain the registration TLVs  
     described in section 3.3.1.2.  
  3. Wait the remaining (tInterval - tBackoff) seconds.  
  4. Set tInterval to 2 * tInterval.  
     If tInterval is greater than tIntervalMax,  
     set tInterval to tIntervalMax.  
} While the device has not received ACK to its registration POST.
```

An example execution of a full registration POST retry sequence is presented in Section 8.

If the device receives an ACK message with CoAP response code 2.03 (valid) from the NMS at any time before the device's next registration POST, the TLVs within the ACK message MUST be processed.

4.3.2. Registration POST Payload

The following TLVs MUST be included in the device registration POST to the NMS:

1. `DeviceID` (primary identifier of the device).

2. CurrentTime (used to validate device local time).

Previously registered devices SHOULD already have (durably stored) values for the Session ID, GroupInfo, and ReportSubscribe TLVs and MUST include these TLVs in the device registration POST to the NMS:

1. SessionID
2. GroupInfo (one per group)
3. ReportSubscribe

The following Device Information TLVs MUST be included in the registration POST:

1. HardwareDesc
2. InterfaceDesc (one per interface)
3. IPAddress (one per address)
4. NMSStatus (reason for the registration operation)
5. WPANStatus
6. RPLSettings

Note that the SessionID, GroupAssign, and ReportSubscribe TLV set is considered to be generic device Configuration. The Configuration TLV set is technology specific and MAY be extended with additional technology specific TLVs (beyond the scope of this specification).

4.3.3. NMS Registration Response

Upon receipt of a registration request, the NMS looks up the information for the device identified by DeviceID to confirm correctness of the request. See Section 3.2.2 for details of DeviceID and SessionID validation and related error response codes.

If the device is found in inventory, authorized to register, and all other registration request content is confirmed to be valid, the NMS MUST send an ACK response message with response code 2.03(Valid) to the device. The ACK takes one of two forms, depending upon whether or not the NMS will redirect the device to another NMS.

In the case where the NMS is not redirecting, the response body to a valid registration request contains the following TLVs:

1. SessionID MUST be elided from the ACK if the SessionID contained in the registration request is correct, otherwise the correct SessionID TLV MUST be included in the ACK.
2. GroupAssign MUST be elided from the ACK if the GroupAssign contained in the registration request is correct, otherwise the correct GroupAssign TLV MUST be included in the ACK.
3. ReportSubscribe MUST be elided from the ACK if the ReportSubscribe contained in the registration request is correct, otherwise the correct ReportSubscribe TLV MUST be included in the ACK.
4. Signing TLVs MUST be included.

In the case where the NMS is redirecting, the response body to a valid registration request contains the following TLVs:

1. NMSRedirectRequest MUST be included.
2. Signing TLVs MUST be included.

When the response contains a SessionID TLV, the device MUST durably store this TLV and SessionID TLV MUST be included in all future CSMP requests to the NMS.

When the response contains a GroupAssign TLV, the device MUST durably store the group-id (overwriting any other stored group-id for the same group-type) and MUST use the new GroupAssign values for comparison with all future receipt of GroupMatch TLVs.

When the response contains a ReportSubscribe TLV, the device MUST begin reporting the indicated metrics TLVs (ignoring any TLVs requested by the ReportSubscribe which are unknown to the device).

The NMS includes the NMSRedirectRequest TLV in a registration response to request the device register with an alternate NMS instance (load balancing, etc.). Upon receipt of this TLV, the device MUST cease registration attempts with the original NMS and start the registration process with the NMS indicated in the NMSRedirectRequest TLV. If registration succeeds with this new NMS, all subsequent device CSMP messaging MUST be directed to this new NMS. Note that device receipt of NMSRedirectRequest TLV is a one-time redirect and not persisted across device restarts.

4.3.4. Registration Complete

The NMS considers device registration to be complete when all of the following conditions are met:

1. The registration ACK to the device indicates code 2.03 (Valid) and message ID match is confirmed as described in CoAP.
2. The ACK response body does not contain an NMSRedirectRequest TLV.
3. The first metrics report from the device is received by the NMS.

4.4. Device Metrics Reporting

Upon receipt of a ReportSubscribe TLV, a device configures as many as two metrics reports:

1. A primary metrics report using the interval and TLV ID set.
2. A secondary metrics report using the heartbeat interval and heartbeat TLV ID set.

The primary metrics report **MUST** be used for mains powered devices (with the secondary report disabled). To conserve power, metrics reporting for low power devices **MAY** be split across primary and secondary reports, with the primary report configured to provide TLVs needed at more frequent interval and the secondary configured for TLVs required at a more relaxed interval.

A device configures metrics parameters to control the device's primary metrics report as follows:

1. `tMetricsInterval` (how often a device **MUST** report its metrics) is typically set between 5 minutes and 8 hours (depends on application requirements). Designated by the `interval` field of the ReportSubscribe TLV.
2. `tlvList` (the list of TLVs the device **MUST** report) is designated by the `tlvId` field of the ReportSubscribe TLV.

A device configures metrics parameters to control a device's secondary metrics report as follows:

1. `tMetricsInterval` is designated by the `intervalHeartBeat` field of the ReportSubscribe TLV.
2. `tlvList` is designated by the `tlvIdHeartBeat` field of the ReportSubscribe TLV.

The TLV content of the primary and secondary metrics reports are deployment and application specific. For example, the primary metrics report for a 6LoWPAN, mains-powered mesh node might be configured as:

1. InterfaceMetrics
2. GroupInfo
3. FirmwareImageInfo
4. Uptime
5. LowpanPhyStats
6. DfServMetrics
7. ReportSubscribe

TLV content of the secondary metrics report is similarly application specific and beyond the scope of this specification.

For each configured metrics report, a device MUST commence reporting immediately after receipt of a successful registration ACK by sending a NON POST to <nms-url>/c containing the following TLVs:

1. SessionID
2. CurrentTime
3. The TLVs from tlvList. The entirety of all table entries MUST be included.

Following the initial metrics report, the device MUST implement the following algorithm for subsequent metric reports (for both primary and secondary report):

Send a new CoAP NON POST to <nms-url>/c with required metrics TLVs. Wait initial random interval between 0 and tMetricsInterval seconds.

Do {

1. Wait tMetricsBackoff seconds, where tMetricsBackoff is random value between tMetricsInterval/2 and tMetricsInterval seconds.
2. Send a new CoAP NON POST message to <nms-url>/c with the required metrics TLVs
3. Wait the remaining (tMetricsInterval - tMetricsBackoff) seconds so that the full tMetricsInterval has expired.

} While the device has a valid IP address.

4.5. Device Firmware Update

CSMP defines a device firmware update process optimized for LPWANS. A key aspect of this process is the separation of image placement on a device from activation (execution) of the image on the device.

The device firmware update process consists of three sub-flows:

1. Firmware download. A new image is placed on one or more members of a device group.
2. Image load. Activate an image on one or more members of a device group at a scheduled time.
3. Set backup image. Optional designation of an image to be used when load of all other images fails.

A device should implement the following mechanisms in support of firmware update:

1. It is RECOMMENDED that vendors implement digital signing of images prior to image release to production.
2. It is RECOMMENDED that a device implement a secure bootloader which (upon device receipt of a LoadRequest TLV or at device power-up) validates the activated image's signature, loads the image from durable storage into operating memory, and transfers execution to the image. Specific details of image signing and the secure bootloader are left to the vendor beyond the scope of this specification.
3. A device MUST be capable of storing at least two firmware images: the running image and at least one additional image.
4. It is RECOMMENDED that a device also support a backup image. A device boots into the backup image when the device is unable to boot into any other image.
5. A device MUST be capable of scheduling an image load (activation) at a specific future time (i.e. the device must maintain a time source).

4.5.1. Firmware Image Format

A CSMP firmware image file consists of three main parts: a CSMP defined image header, the vendor defined image binary, and the vendor defined image signature (as depicted below).

Field	Size (octets)	Description
		Begin Header
Header Version	4	32 bit unsigned integer which MUST be set to 2.
Header Length	4	32 bit unsigned integer which MUST be set to 256.
App Rev Major	4	Vendor specific 32 bit unsigned integer which is set to indicate the major revision number of the application image.
App Rev Minor	4	Vendor specific 32 bit unsigned integer which is set to indicate the minor revision number of the application image.
App Build	4	Vendor specific 32 bit unsigned integer which is set to indicate the build of the application image.
App Length	4	32 bit unsigned integer which MUST be set to the octet length of the Header + Image Binary field.
App Name	32	Vendor specific 32 octet string which is set to indicate the name of the application.
App SCC Branch	32	Vendor specific 32 octet string which is set to indicate the source code control system branch ID.
App SCC Commit	8	Vendor specific 8 octet string which is set to indicate the source code control system commit ID.
App SCC Flags	4	Vendor specific 32 bit unsigned integer which is set to indicate

		the source code control system build flags.
App Build Date	16	Vendor specific 16 octet string which is set to indicate the build date and time of the application image.
hwid	32	32 octet field which is RECOMMENDED to be set to a concatenation of a unique manufacturer ID and product model identifier.
sub_hwid	32	32 octet field which MAY be set for a manufacturer specific purpose, or functionally elided by filling with 0x20 (ASCII space character).
kernel_rev	16	16 octet field which MAY be set for a manufacturer specific purpose, or functionally elided by filling with 0x20 (ASCII space character).
sub_kernel_rev	16	16 octet field which MAY be set for a manufacturer specific purpose, or functionally elided by filling with 0x20 (ASCII space character).
Reserved	44	Pad to 256 octets, octets MUST be set to 0.
		End Header, Begin Image
Image Binary	Variable	Vendor specific image data.
		End Image, Begin Signature
Signature Variable	Vendor specific image signature.	Calculated over entire content of this structure except the signature and pad fields.
		End Signature, Begin Pad

Pad	Variable	Optional pad field to enable image to fill vendor specific flash memory boundary. When present, octets MUST be set to 0xFF.
-----	----------	---

Table 1: Firmware Image Format

All multi-octet fields are encoded as little-endian.

4.5.2. Firmware Download

An NMS implements the messaging flow depicted in Figure 3 and Figure 4 to download an image to a group of devices. Unicast distribution is also supported, with the difference being use of unicast addresses, omission of the GroupMatch TLV and omission of the "a" query option.

Figure only available as SVG (PDF and HTML)

Figure 3: Firmware Download

Figure only available as SVG (PDF and HTML)

Figure 4: Firmware Download (cont)

A firmware download begins with the NMS informing the device group of the meta-data of an image the NMS wishes to download, and the devices subsequently informing the NMS of the images already loaded on the devices.

The NMS MUST issue a NON POST to <device-url>/c configured as follows:

1. A GroupMatch TLV MUST be included for the desired group.
2. The "a" option MUST be specified to randomize subsequent device responses.
3. The request MAY contain an "r" option to redirect the subsequent TransferResponse.
4. The request MUST contain a TransferRequest TLV (meta-data for the file to be downloaded)
5. The request MUST contain the Signing TLVs.

Devices receiving a TransferRequest message:

1. MUST process the Signing TLVs and the GroupMatch TLV as described in section 2.6.
2. MUST wait the specified period when the "a" option is included.
3. MUST issue a NON POST request to <nms-url>/c which MUST contain the TransferResponse TLV. The <nms-url> MUST be overridden by the "r" option if specified.

The NMS MUST proceed with image block transfer when at least one member of the target device group indicates Response Code of OK in the TransferResponse TLV. Otherwise, the transfer MUST be aborted.

Images are often multiple 100s of Kilobytes in size and likely require fragmentation into multiple blocks (N) to be transferred to a device.

For the transfer of each image block, the NMS MUST issue a NON POST to <device-url>/c configured as follows:

1. The request MUST contain the GroupMatch TLV for the desired group.
2. The request MUST contain an ImageBlock TLV
3. The request MAY contain the Signing TLVs.

To determine image download progress, the NMS MUST periodically include a DescriptionRequest TLV, requesting the FirmwareImageInfo TLV, in the image block request. It is RECOMMENDED that the NMS request the FirmwareImageInfo TLV at 10% increments of the image download. The NMS MUST request the FirmwareImageInfo TLV with the transfer of the last block of the image.

Devices receiving the image block request message:

1. MUST process the Signing TLVs and GroupMatch TLVs as described in section 2.6.
2. MUST cache the image block for final image assembly.
3. When the FirmwareImageInfo TLV is requested, the device MUST wait the specified period when the 'a' option is included and MUST issue a NON POST request to <nms-url>/c which MUST contain the FirmwareImageInfo TLV (meta-data for files loaded on the device). The <nms-url> MUST be overridden by the "r" option if specified.

Receipt of the final FirmwareImageInfo TLV enables the NMS to confirm the integrity of the completely downloaded image.

4.5.3. Image Activation

The messaging flow for scheduling image activation across a group of devices and cancelling a scheduled image activation are depicted in Figure 5 and described below. Unicast activation is also supported, with the difference being use of unicast addresses, omission of the GroupMatch TLV and omission of the "a" query option.

Figure only available as SVG (PDF and HTML)

Figure 5: Image Activation

4.5.3.1. Image Load

An NMS implements the following message flow to command devices to designate an image as the active executable and the time at which the image is to be activated.

The NMS MUST issue a NON POST to <device-url>/c configured as follows:

1. A GroupMatch TLV MUST be included for the desired group.
2. The "a" option MUST be specified to randomize subsequent device responses.
3. The request MAY contain an "r" option to redirect the subsequent LoadResponse.
4. The request MUST contain a LoadRequest TLV (designating the image and time at which the image is to be activated).
5. The request MUST contain the Signing TLVs.

Devices receiving a LoadRequest message:

1. MUST process the Signing TLVs and GroupMatch TLVs as described in section 2.6.
2. MUST wait the specified period when the "a" option is included.
3. MUST issue a NON POST request to <nms-url>/c which MUST contain the LoadResponse TLV (indicating success or reason for failure). The <nms-url> MUST be overridden by the "r" option if specified.

4.5.3.2. Cancel Image Load

An NMS implements the following message flow to cancel a previously scheduled image activation.

The NMS MUST issue a NON POST to <device-url>/c configured as follows:

1. A GroupMatch TLV MUST be included for the desired group.
2. The "a" option MUST be specified to randomize subsequent device responses.
3. The request MAY contain an "r" option to redirect the subsequent CancelLoadResponse.
4. The request MUST contain a CancelLoadRequest TLV (designating the image load to be cancelled).
5. The request MUST contain the Signing TLVs.

Devices receiving a CancelLoadRequest message:

1. MUST process the Signing TLVs and GroupMatch TLVs as described in section 2.6.
2. MUST wait the specified period when the "a" option is included.
3. MUST issue a NON POST request to <nms-url>/c which MUST contain the CancelLoadResponse TLV (indicating success or reason for failure). The <nms-url> MUST be overridden by the "r" option if specified.

4.5.4. Set Backup Image

An NMS implements the following message flow to command a device to designate a stored image as the backup image.

Figure only available as SVG (PDF and HTML)

Figure 6: Set Backup Image

The NMS MUST issue a NON POST to <device-url>/c configured as follows:

1. A GroupMatch TLV MUST be included for the desired group.

2. The "a" option MUST be specified to randomize subsequent device responses.
3. The request MAY contain an "r" option to redirect the subsequent SetBackupResponse.
4. The request MUST contain a SetBackupRequest TLV (designating the image load to be cancelled).
5. The request MUST contain the Signing TLVs.

Devices receiving a SetBackupRequest message:

1. MUST process the Signing TLVs and GroupMatch TLVs as described in section 2.6.
2. MUST wait the specified period when the "a" option is included.
3. MUST issue a NON POST request to <nms-url>/c which MUST contain the SetBackupResponse TLV (indicating success or reason for failure). The <nms-url> MUST be overridden by the "r" option if specified.

4.6. Device Commands

Many TLVs served by a device are used by the NMS to interrogate the device for configuration state and operational status. There are, however, several TLVs which direct a device to execute internal actions. Examples of these TLVs are PingRequest and RebootRequest.

Usage of a command TLVs is illustrated with the following PingRequest example directed at a single device.

The NMS MUST issue a NON POST to <device-url>/c configured as follows:

1. A GroupMatch TLV MUST NOT be included for the designed device.
2. The request MAY contain an "r" option to redirect the subsequent PingResponse.
3. The request MUST contain a PingRequest TLV (describing the Ping action to be performed).
4. The request MUST contain the Signing TLVs.

Devices receiving a PingRequest message:

1. MUST process the Signing TLVs as described in section 2.6.
2. MUST begin the requested Ping operation.

The NMS will subsequently interrogate the device with one or more GET requests to the device for the PingResponse TLV.

Note the specific messaging exchanges vary per the definition of each commands. Details are provided within Section 3.3.2.3.

5. Security Considerations

Threat models and appropriate mitigations are highly specific to individual CSMP/LPWAN deployments.

CSMP defines only the NMS signing of outgoing Device messaging using an NMS private key (for Device commands and firmware load). Signing TLVs included in the message payload enable signature verification by a Device using an NMS signing certificate\public key. The verified signature provides source authentication integrity check of the message incoming to the Device. Lifecycle management of the public/private keypair is out of scope of the specification.

Where additional security mechanisms are needed (source and integrity checking of Device to NMS communication, confidentiality of CSMP messaging, authentication and authorization of CSMP actors, replay protection), additional layer 2, 3, or 4 security mechanisms are utilized to meet security requirements of a specific deployment. Examples include:

1. Layer 2 802.1X/EAP-TLS is used to provide mutual authentication of Device and NMS as well as distribution of key material to be used with IEEE 802.15.4 frame security (providing confidentiality, source authentication, and replay protection).
2. Layer 3 VPN may be used to provide confidentiality, source authentication, and message integrity for messaging between Device and NMS.
3. Layer 4 DTLS may be used to provide confidentiality, source authentication, and message integrity for messaging between Device and NMS.

Specific usage profile details for these additional security mechanisms are out of scope of this specification.

6. IANA Considerations

This document requires no IANA actions.

7. Contributors

This work is based upon the considerable efforts of CSMP's original designer Gilman Tolle (with contributions from Phil Buonadonna, and Sumit Rangwala). The Editor further acknowledges the contributions made to the content of this document by the following individuals:

1. Jasvinder Bhasin, Cisco Systems, Inc. (jassi@cisco.com)
2. Chris Hett, Landis+Gyr (Chris.Hett@landisgyr.com)
3. Johannes van der Horst, MMB Networks
(johannes.vanderhorst@mmbresearch.com)
4. Klaus Hueske, Renesas (Klaus.Hueske@renesas.com).
5. Hideyuki Kuribayashi, ROHM Semiconductor
(Hideyuki.Kuribayashi@mnf.rohm.co.jp)
6. Kit-Mui Leung, Cisco Systems, Inc. (kml@cisco.com)
7. Huimin She, Cisco Systems, Inc. (hushe@cisco.com)
8. Li Zhao, Cisco Systems, Inc. (liz3@cisco.com)

8. Appendix A Registration Retry Example

1. Device powers up.
2. Device sets interval for 0 to 5 minutes.
3. Device wait a random time between 2.5 and 5 minutes.
4. Device sends a confirmable registration POST message.
5. Device waits the rest of the interval until 5 minutes have passed.
6. Device waits a random time between 5 and 10 minutes.
7. Device sends the registration message again, with a new CoAP message ID.

8. Device waits the rest of the interval until 10 minutes have passed.
 9. Device waits a random time between 10 and 20 minutes.
 10. Device sends the registration message again, with a new CoAP message ID.
 11. Device waits the rest of the interval until 20 minutes have passed.
 12. Device waits a random time between 20 and 40 minutes.
 13. Device sends the registration message again, with a new CoAP message ID.
 14. Device waits the rest of the interval until 40 minutes have passed.
 15. Device waits a random time between 40 and 60 minutes.
 16. Device sends the registration message again, with a new CoAP message ID.
 17. Device waits the rest of the interval until 60 minutes have passed.
 18. Repeat steps 15, 16, and 17 forever.
9. Appendix B Change Log
 - 9.1. draft 00 to 01
 1. Abstract reworded to clarify multi-vendor aspects of CSMP.
 2. Introduction reworded to clarify multi-vendor aspects of CSMP.
 3. OpenAPI definitions are placed in-line.
 4. Protocol Buffer Definitions are placed in-line.
 5. Due to IETF author limits, all authors except the Editor have been credited in the Contributors section.
 - 9.2. draft 01 to 02
 1. "Cisco" added to title as requested.

2. There are no IP disclosures to be made for this work. This point has been confirmed by Cisco legal.

9.3. draft 02 to 03

1. Trial attempt to convert Figures 2 and 6 to ASCII Art using PlantUML.

9.4. draft 03 to 04

1. Formatted OpenAPI and Protocol Buffer content to fit 72 char margin. NOTE due to limits on URL length, the OpenAPI content no longer will compile without modification. Reader should refer to source URLs for actual working OpenAPI definitions.
2. Removed -03 Figures 2 and 6 sequence diagram ASCII Art (generated from PlantUML ... still too wide) and restored the SVG.
3. Converted Figure 1 state machine SVG to ASCII art (gag).

9.5. draft 04 to 05

1. Keepalive revision while associated code base work progresses.

9.6. draft 05 to 06

1. Keepalive revision while associated code base work progresses.

9.7. draft 06 to 07

1. Keepalive revision while open source code base integrates FreeRTOS contributions.

9.8. draft 07 to 08

1. Added Vendor Defined TLV details.

9.9. draft 08 to 09

1. Keepalive revision while open source code base integrates FW update capabilities.

9.10. draft 09 to 10

1. Keepalive revision as work continues on the open source code base.

10. Normative References

- [CSMPCOMP] "CSMP Components", n.d., <<https://github.com/woobagooba/draft-ietf-is-csmp/blob/e0be5a31906eb3e9983e7cc28b24ed9482543784/CsmpComponents-1.0.yaml>>.
- [CSMPDEV] "CSMP Device Interface", n.d., <<https://github.com/woobagooba/draft-ietf-is-csmp/blob/e0be5a31906eb3e9983e7cc28b24ed9482543784/CsmpDevice-1.0.1.yaml>>.
- [CSMPMSG] "CSMP Payload Definitions", n.d., <<https://github.com/woobagooba/draft-ietf-is-csmp/blob/e0be5a31906eb3e9983e7cc28b24ed9482543784/CsmpTLVsPublic.proto>>.
- [CSMPNMS] "CSMP NMS Interface", n.d., <<https://github.com/woobagooba/draft-ietf-is-csmp/blob/e0be5a31906eb3e9983e7cc28b24ed9482543784/CsmpNms-1.0.1.yaml>>.
- [OPENAPI] "OpenAPI Specification v3.0.0", n.d., <<https://spec.openapis.org/oas/v3.0.0>>.
- [PB] "Protocol Buffers Version 3", n.d., <<https://developers.google.com/protocol-buffers/docs/proto3>>.
- [PEN] "Application for a Private Enterprise Number", n.d., <<https://www.iana.org/assignments/enterprise-numbers/assignment/apply/>>.
- [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/rfc/rfc2119>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/rfc/rfc7252>>.
- [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/rfc/rfc8174>>.

Author's Address

Paul Duffy
Cisco Systems, Inc.
Email: paduffy@cisco.com