Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 17 April 2026

C. Yu
Huawei Technologies
S. Belotti
Nokia
J.-F. Bouquier
Vodafone
F. Peruzzini
FiberCop
P. Bedard
Cisco

14 October 2025

A Base YANG Data Model for Network Inventory
draft-ietf-ivy-network-inventory-yang-11

Abstract

This document defines a base YANG data model for reporting network inventory. The scope of this base model is set to be application- and technology-agnostic. The base data model can be augmented with application- and technology-specific details.

About This Document

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

The latest revision of this draft can be found at https://ietf-ivy-wg.github.io/network-inventory-yang/draft-ietf-ivy-network-inventory-yang.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-ivy-network-inventory-yang/.

Source for this draft and an issue tracker can be found at https://github.com/ietf-ivy-wg/network-inventory-yang.

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/.

Copyright Notice

Table of Contents

1. Introduction

This document defines a base YANG data model for reporting network inventory that is application- and technology-agnostic. The base data model can be augmented to describe application- and technology-specific information. Please note that the usage of term "network inventory", in the context of this I-D, is to indicate that it is describing "network-wide" scope inventory information.

Network Inventory is a collection of data for network devices and their components managed by a specific management system.

Network inventory is a fundamental functional block in the overall network management which was specified many years ago. Network inventory management is a critical part for ensuring that the network remains healthy (e.g., auditing to identify faulty elements), well-planned (e.g., identify assets to upgrade or to decommission), and maintained appropriately to meet the performance objectives. Also, network inventory management allows operators to keep track of which devices are deployed in their networks, including relevant embedded software and hardware versions.

Exposing standard interfaces to retrieve network elements capabilities as maintained in an inventory are key enablers for many applications. For example, [I-D.ietf-teas-actn-poi-applicability] identifies a gap about the lack of YANG data models that could be used at Abstraction and Control of TE Networks (ACTN) Multi-Domain Service Coordinator-Provisioning Network Controller Interface (MPI) level to report whole or partial network hardware inventory information available at domain controller level towards upper layer systems (e.g., Multi-Domain Service Coordinator (MDSC) or Operations Support Systems (OSS) layers).

It is key for operators to coordinate with the industry towards the use of a standard YANG data model for Network Inventory data instead of using vendors proprietary APIs.

[RFC8348] defines a YANG data model for the management of the hardware on a single server and therefore it is more applicable to the domain controller towards the network elements rather than at the northbound interface of a network controller (e.g., toward an application or another hierarchical network controller). However, the YANG data model defined in [RFC8348] has been used as a reference for defining the YANG network inventory data model presented in this document.

Per the definition of [RFC8969], the YANG data model defined in [RFC8348] is a device model while the YANG data model defined in this document is a network model.

This document defines one YANG module "ietf-network-inventory" in Section 5.

This base data model is application- and technology-agnostic (that is, valid for IP/MPLS, optical, and microwave networks as well as optical local loops, access networks, core networks, data centers, etc.) and can be augmented to include required application- and technology-specific inventory details together with specific hardware or software component’s attributes.

The YANG data model defined in the document is scoped to cover the common use cases for Inventory but at network-wide level, covering both hardware and base software information.

The YANG data model defined in this document conforms to the Network Management Datastore Architecture [RFC8342].

1.1. Editorial Note (To be removed by RFC Editor)

Note to the RFC Editor: This section is to be removed prior to publication.

This document contains placeholder values that need to be replaced with finalized values at the time of publication. This note summarizes all of the substitutions that are needed.

Please apply the following replacements:

* XXXX --> the assigned RFC number for this I-D

2. Terminology and Notations

2.1. Requirements Notations

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

2.2. Terminology

The following terms are defined in [RFC7950] and are not redefined here:

* server
* augment
* data model
* data node

The following terms are defined in [RFC6241] and are not redefined here:

* configuration data
* state data

The following terms are defined in [RFC8342] and are not redefined here:

* applied configuration

The following terms are defined in the description statements of the corresponding YANG identities, defined in [IANA_HW_YANG], and are not redefined here:

* backplane
* battery
* cpu
* fan
* module

* power supply
* sensor
* stack
* storage device

Chassis: A field replaceable equipment with a particular structural format and dimensions. A chassis can, but does not need to, include spaces (called slots) to take cards.

Elsewhere, a chassis can be called shelf, sub-rack, stand-alone unit, etc.

Port: A component where networking traffic can be received and/or transmitted, e.g., by attaching networking cables.

In case of pluggable ports, the port may be empty when no pluggable module is plugged in.

Also, the document makes use of the following terms:

Network Inventory: A collection of data for network elements and their components with network-wide scope, managed by a specific management system.

Physical Network Element: An implementation or application specific group of components (e.g., hardware components).

Network Element: The generalization of the physical network element definition.

Hardware Component: The generalization of the hardware components defined in [IANA_HW_YANG] (e.g., backplane, battery, container, cpu, chassis, fan, module, port, power supply, sensor, stack, and storage device components).

The list of hardware components can be extended in future versions of [IANA_ENTITY_MIB] (and, consequently, of ([IANA_HW_YANG]).

Component: The generalization of the hardware component definition to include other inventory objects which can be managed, from an inventory perspective, like hardware components.

Card: A pluggable equipment with a particular structural format and

dimensions which can be inserted into one or more slots (or sub-slots). A card can have spaces (called sub-slots) to take other cards.

Elsewhere, a card can be called board, module, circuit pack, etc..

Slot: A space in a chassis that can be equipped with one card, which may be chosen from a limited range of types of cards. A slot can be subdivided into smaller spaces (called sub-slots).

Container: A hardware component class that is capable of containing one or more removable physical entities (e.g., a slot in a chassis is containing a board).

2.3. Tree Diagrams

The meanings of the symbols in the YANG tree diagrams are defined in [RFC8340].

2.4. YANG Prefixes

Table 1 list the prefixes of the modules that are used in this document.

Prefix	YANG Module	Reference
inet	ietf-inet-types	Section 4 of [RFC6991]
yang	ietf-yang-types	Section 3 of [RFC6991]
ianahw	iana-hardware	[IANA_HW_YANG]
nwi	ietf-network-inventory	RFC XXXX

Table 1: Prefixes and corresponding YANG modules

3. YANG Data Model for Network Inventory Overview

The base network inventory model, defined in this document, provides a list of network elements and of network element components:

The network-inventory top level container has been defined to support reporting other types of network inventory objects, besides the network elements and network element components.

These additional types of network inventory objects can be defined, together with the associated YANG data model and the rationale for managing them as part of the network inventory, in other documents providing application- and technology-specific companion augmentation data models, such as [I-D.ietf-ivy-network-inventory-location].

The network element definition is generalized to support physical network elements and other types of components' groups that can be managed as physical network elements from an inventory perspective.

Physical network elements are usually devices such as hosts, gateways, terminal servers, and the like, which have management agents responsible for performing the network management functions requested by the network management stations ([RFC1157]).

The "ne-type" is defined as a YANG identity to describe the type of the network element. This document defines only the "physical-network-element" identity.

Other types of network elements can be defined in other documents, together with the associated YANG identity and the rationale for managing them as network elements from an inventory perspective.

The component definition is also generalized to support any types of component inventory objects that can be managed as hardware components from an inventory perspective.

The data model for components defined in this document uses a list of components within each network element.

Different types of components can be distinguished by the class of component. The component "class" is defined as a union between the hardware class identity, defined in "iana-hardware", and the "non-hardware" identity, defined in this document.

Other types of components can be defined in other documents, together with the associated YANG identity and the rationale for managing them as components from an inventory perspective.

The identity definition of additional types of "ne-type" and "non-hardware" identity of component are outside the scope of this document and could be defined in application- and technology-specific companion augmentation data models, such as [I-D.ietf-ivy-network-inventory-software].

In [RFC8348], rack, chassis, slot, sub-slot, board and port are defined as components of network elements with generic attributes.

While [RFC8348] is used to manage the hardware of a single server (e.g., a network element), the Network Inventory YANG data model is used to retrieve the base inventory information that a controller discovers from all the network elements with network-wide scope under its control.

However, the YANG data model defined in [RFC8348] has been used as a reference for defining the YANG network inventory data model. This approach can simplify the implementation of this inventory model when the controller uses the YANG data model defined in [RFC8348] to retrieve the hardware from the network elements under its control.

3.1. Common attributes for inventory object

For all the inventory objects, there are some common attributes, such as:

uuid: The Universally Unique Identifier (UUID) of the inventory object, assigned by the server. Such identifiers are widely implemented with systems and guaranteed to be globally unique.

name: A human-readable label information of the inventory object, which could be assigned by the server. It could also be present on a Graphical User Interface (GUI).

alias: A human-readable label information of the inventory object, provided by a network operator. It could also be present on a GUI instead as well as the name.

description: A human-readable description of the inventory object, provided by a network operator or by the server. The description provides more detailed information to prompt users when performing maintenance operations etc.

3.1.1. Common attributes for network elements and components

To be consistent with the component definition, some of the attributes defined in [RFC8348] for components are reused for network elements, such as:

mfg-name: The name of the manufacturer of the entity (component or network element).

product-name: The vendor-specific and human-interpretable string describing the entity (component or network element) type.

It is expected that vendors assign unique product names to different entities within the scope of the vendor.

Other software-related attributes are defined in Section 3.3.2 and applicable to network elements and components.

3.2. Network Element

In addition to the common attributes defined for network elements and components in Section 3.1, the following attributes are defined for the network elements:

ne-id: The identifier that uniquely identifies the NE within the network, assigned by the server since the network elements cannot guaranteed that their local identifier is unique within the network.

The ne-id should be assigned such that the same network element will always be identified through the same identifier, even if the network elements get disconnected from the network controller. Mechanisms to ensure this (e.g., checking the mfg-name, product-name, management IP address, physical location) are implementation specific and outside the scope of standardization.

ne-type: The type of network element (e.g., physical network element). See Section 3 for the definition of NE types.

product-rev: A vendor-specific product revision string for the network-element.

3.3. Components

The YANG data model for network inventory mainly follows the same approach of [RFC8348] and reports the network hardware inventory as a list of components with different types (e.g., chassis, module, and port).

In addition to the common attributes defined for network elements and components in Section 3.1, the following attributes are defined for the network elements:

component-id: The identifier that uniquely identifies the component within the NE. It can be assigned by the NE or by the server.

class: The type of component (e.g., chassis, module, port). See Section 3 for the definition of component types.

hardware-rev: The vendor-specific hardware revision string for the component.

The preferred value is the hardware revision identifier actually

printed on the component itself (if present).

mfg-date: The date of manufacturing of the component.

part-number: The vendor-specific part number of the component type.

It is expected that vendors assign unique part numbers to different component types within the scope of the vendor.

serial-number: The vendor-specific serial number of the the component instance.

It is expected that vendors assign unique serial numbers to different component instances at least within the scope of the part-number.

asset-id: An asset tracking identifier for the component, provided by a network operator.

is-fru: Indicates whether or not a component is considered a 'field-replaceable unit' by the vendor.

For state data like "admin-state", "oper-state", and so on, this document considers that they are related to device hardware management, not network inventory. Therefore, they are outside of the scope of this document. Same for the sensor-data, they should be defined in some other performance monitoring data models instead of the inventory data model.

3.3.1. Hardware Components

Based on TMF classification in [TMF_SD2-20], hardware components can be divided into two groups, holder group and equipment group. The holder group contains rack, chassis, slot, sub-slot while the equipment group contains network-element, board and port.

Figure 1 describes the relationship between typical inventory objects in a physical network element.

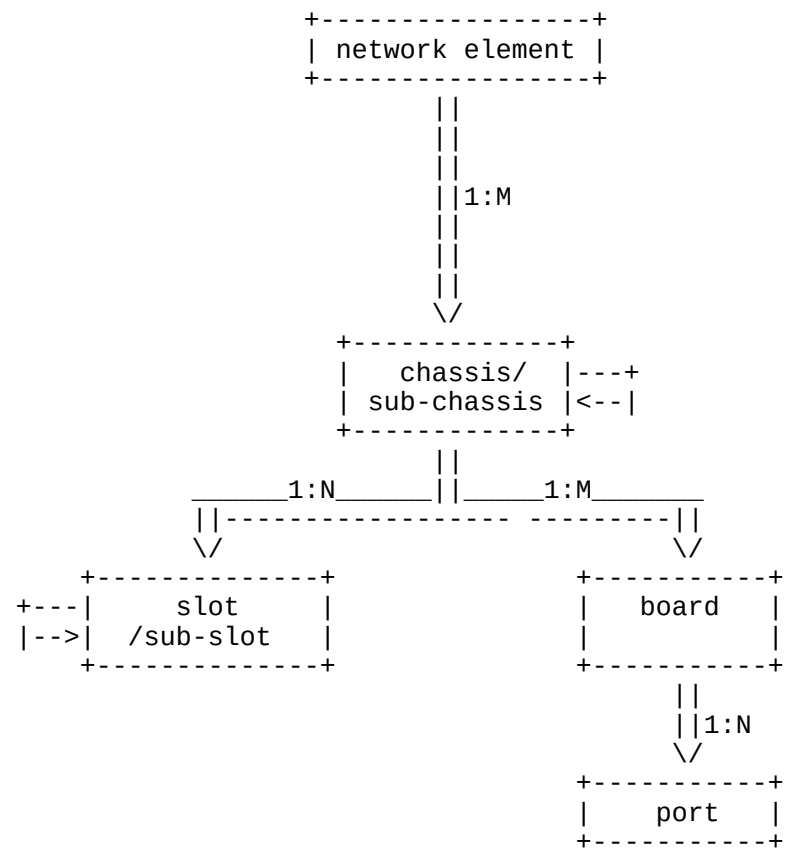

Figure 1: Relationship between typical inventory objects in physical network elements

The "iana-hardware" module [IANA_HW_YANG] defines YANG identities for the hardware component types in the IANA-maintained "IANA-ENTITY-MIB" registry.

Some of the definitions taken from [RFC8348] are based on the ENTITY-MIB [RFC6933].

Additional attributes of specific hardware, such as CPU, storage, port, or power supply are defined in the hardware extension.

3.3.2. Software Components

This document defines a "software-rev" list for NEs and components, which provide basic software attributes for network elements and components.

The scope of the list is to provide information about the software modules configured to be active on the related entity (network element or component).

The model supports scenarios where multiple software modules can be configured to be active on the entity. For example, on a network element an Operating System and an Application software modules can be configured to be active; in the same way, on a component like a circuit pack a boot-loader, a firmware and one or more FPGA software modules can be configured to be active.

For each software module, configured to be active, the name and version information is provided.

The management of inactive/standby software modules and of the software upgrade or downgrade life-cycle are outside the scope of the base inventory model and can be addressed in other models which augment the base inventory model such as the model under definition in [I-D.ietf-ivy-network-inventory-software].

The software and hardware components share the same attributes of the component and have similar replaceable requirements. Generally, the device also has other software data, for example, one or more software patch information.

The software components of other classes, such as platform software, BIOS, bootloader, and software patch information, are outside the scope of this document and defined other documents such as [I-D.ietf-ivy-network-inventory-software].

3.4. Changes Since RFC 8348

This document re-defines some attributes listed in [RFC8348], based on some integration experience for network inventory data.

3.4.1. Part Number

According to the description in [RFC8348], the attribute named "model-name" under the component, is preferred to have a customer-visible part number value. "Model-name" is not straightforward to understand and we suggest to rename it as "part-number" directly.

3.4.2. Component identifiers

There are some use cases where the name of the components are assigned and changed by the operator. In these cases, the assigned names are also not guaranteed to be always unique.

In order to support these use cases, this model is not aligned with [RFC8348] in defining the component name as the key for the component list.

Instead the name is defined as an optional attribute and the component-id is defined as the key for the component list (in alignment with the approach followed for the network-element list).

4. Network Inventory Tree Diagram

Figure 2 below shows the tree diagram of the YANG data model defined in module "ietf-network-inventory" (Section 5).

```
module: ietf-network-inventory
  +--ro network-inventory
     +--ro network-elements
        +--ro network-element* [ne-id]
           +--ro ne-id           string
           +--ro ne-type?        identityref
           +--ro uuid?           yang:uuid
           +--ro name?           string
           +--ro alias?          string
           +--ro description?    string
           +--ro software-rev* [name]
           |  +--ro name         string
           |  +--ro revision?    string
           |  +--ro patch* [revision]
           |     +--ro revision    string
           +--ro mfg-name?       string
           +--ro product-name?   string
           +--ro product-rev?    string
           +--ro components
              +--ro component* [component-id]
                 +--ro component-id      string
                 +--ro class             union
                 +--ro uuid?             yang:uuid
                 +--ro name?             string
                 +--ro alias?            string
                 +--ro description?      string
                 +--ro software-rev* [name]
                 |  +--ro name         string
                 |  +--ro revision?    string
                 |  +--ro patch* [revision]
                 |     +--ro revision    string
                 +--ro mfg-name?         string
                 +--ro product-name?     string
                 +--ro hardware-rev?     string
                 +--ro mfg-date?         yang:date-and-time
                 +--ro part-number?      string
                 +--ro serial-number?    string
                 +--ro asset-id?         string
                 +--ro is-fru?           boolean
                 +--ro uri*              inet:uri
                 +--ro parent-rel-pos?   int32
                 +--ro parent*
                 |       -> ../../component/component-id
                 +--ro is-main?          boolean
```

Figure 2: Network inventory tree diagram

5. YANG Data Model for Network Inventory

```
<CODE BEGINS> file "ietf-network-inventory@2025-10-07.yang"
module ietf-network-inventory {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-network-inventory";
  prefix nwi;

  import iana-hardware {
    prefix ianahw;
    reference
      "https://www.iana.org/assignments/yang-parameters";
  }
  import ietf-yang-types {
    prefix yang;
    reference
      "RFC 6991: Common YANG Data Types";
  }
  import ietf-inet-types {
    prefix inet;
    reference
      "RFC 6991: Common YANG Data Types";
  }

  organization
    "IETF IVY Working Group";
  contact
    "WG Web:   <https://datatracker.ietf.org/wg/ivy/>
     WG List:  <mailto:inventory-yang@ietf.org>

     Editor:   Chaode Yu
               <yuchaode@huawei.com>

     Editor:   Sergio Belotti
               <sergio.belotti@nokia.com>

     Editor:   Jean-Francois Bouquier
               <jeff.bouquier@vodafone.com>

     Editor:   Fabio Peruzzini
               <fabio.peruzzini@telecomitalia.it>

     Editor:   Phil Bedard
               <phbedard@cisco.com>";
  description
    "This module defines a base model for retrieving network
     inventory.
```

```
        The model fully conforms to the Network Management
        Datastore Architecture (NMDA).

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

        Redistribution and use in source and binary forms, with or
        without modification, is permitted pursuant to, and subject
        to the license terms contained in, the Revised BSD License
        set forth in Section 4.c of the IETF Trust's Legal Provisions
        Relating to IETF Documents
        (https://trustee.ietf.org/license-info).

        This version of this YANG module is part of RFC XXXX; see
        the RFC itself for full legal notices.

        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 (RFC 2119) (RFC 8174) when, and only when,
        they appear in all capitals, as shown here.";

    // RFC Ed.: update the date below with the date of RFC publication
    // and remove this note.

    revision 2025-10-09 {
      description
        "Initial version";
      reference
        "RFC XXXX: A YANG Data Model for Network Inventory.";
    }

    /*
     * Identities
     */

    identity non-hardware-component-class {
      description
        "Base identity for non hardware components (e.g., software
         components) in a managed device.";
    }

    identity ne-type {
      description
        "Base identity for Network Element (NE) types.";
    }

    identity ne-physical {
```

```
    base nwi:ne-type;
    description
      "A physical network element (NE). ";
  }

  /*
   * Types
   */

  typedef ne-ref {
    type leafref {
      path "/nwi:network-inventory/nwi:network-elements"
         + "/nwi:network-element/nwi:ne-id";
    }
    description
      "This type is intended to be used by data models that need to
       reference Network Element.";
  }

  /*
   * Groupings
   */

  grouping port-ref {
    description
      "This grouping is intended to be used by data models that need
       to reference a port component within a Network Element.";
    leaf ne-ref {
      type nwi:ne-ref;
      description
        "The reference to the Network Element which contains the
         port to be referenced.";
    }
    leaf port-ref {
      type leafref {
        path "/nwi:network-inventory/nwi:network-elements/"
           + "nwi:network-element[nwi:ne-id=current()/../ne-ref]"
           + "/nwi:components/nwi:component/nwi:component-id";
      }
      must "derived-from-or-self (/nwi:network-inventory/
            nwi:network-elements/nwi:network-element
            [nwi:ne-id=../ne-ref]/nwi:components/nwi:component
            [nwi:component-id=current()]/nwi:class, 'ianahw:port')";
      description
        "The reference to the port component.";
    }
  }
```

```
    grouping basic-common-entity-attributes {
      description
        "The set of basic attributes which are common to all the
         entities (e.g., component, network elements, location, passive
         entities) defined in this module and in other inventory
         modules.";
      leaf uuid {
        type yang:uuid;
        description
          "The Universally Unique Identifier of the entity
           (e.g., component).";
      }
      leaf name {
        type string;
        description
          "The name of the  entity (e.g., component), as specified by
           a network operator, that provides a non-volatile 'handle'
           for the entity and that can be modified anytime during the
           entity lifetime.

           If no configured value exists, the server MAY set the value
           of this node to a locally unique value in the operational
           state.";
      }
      leaf alias {
        type string;
        description
          "The alias name of the entity (e.g., component). This alias
           name can be specified by a network operator.";
      }
      leaf description {
        type string;
        description
          "The textual description of the entity (e.g., component).";
      }
    }

    grouping ne-component-common-entity-attributes {
      description
        "The set of attributes which are common to all the entities
         (e.g., component, network elements) defined in this module.";
      uses basic-common-entity-attributes;
      list software-rev {
        key "name";
        description
          "The list of the software modules configured to be active
           within the entity (e.g., component).";
        leaf name {
```

```
          type string;
          description
            "The vendor-specific name of the software module.";
        }
        leaf revision {
          type string;
          description
            "The vendor-specific revision string of the software
             module when not implicitly defined as part of the name of
             the software module.";
        }
        list patch {
          key "revision";
          description
            "The list of software patches configured to be active for
             the software module.";
          leaf revision {
            type string;
            description
              "The vendor-specific revision string of the software
               patch when not implicitly defined as part of the name or
               revision of the software module.";
          }
        }
      }
      leaf mfg-name {
        type string;
        description
          "The name of the manufacturer of this entity
           (e.g., component).";
      }
      leaf product-name {
        type string;
        description
          "The vendor-specific and human-interpretable string
           describing the entity (e.g., component) type. It is expected
           that vendors assign unique product names to different entity
           (e.g., component) types within the scope of the vendor.";
      }
    }

    grouping component-attributes {
      description
        "The set of common attributes of a component.

         This grouping is intended also to be re-used by data models
         that need to report the common attributes of a component.";
      leaf component-id {
```

```
          type string;
          description
            "An identifier that uniquely identifies the component
             in a node.";
        }
        leaf class {
          type union {
            type identityref {
              base ianahw:hardware-class;
            }
            type identityref {
              base non-hardware-component-class;
            }
          }
          mandatory true;
          description
            "The type of the component.";
        }
        uses ne-component-common-entity-attributes {
          refine "software-rev" {
            reference
              "RFC 6933: Entity MIB (Version 4) -
                         entPhysicalSoftwareRev";
          }
          refine "mfg-name" {
            description
              "The name of the manufacturer of this physical
               component.
               The preferred value is the manufacturer name
               string actually printed on the component itself
               (if present).

               Note that comparisons between instances of the
               'model-name', 'software-rev', and 'serial-number' nodes
               are only meaningful amongst components with the same value
               of 'mfg-name'.

               If the manufacturer name string associated with
               the physical component is unknown to the server,
               then this node is not instantiated.";
            reference
              "RFC 6933: Entity MIB (Version 4) -
                         entPhysicalMfgName";
          }
        }
        leaf hardware-rev {
          type string;
          description
```

```
          "The vendor-specific hardware revision string for the
           component.

           The preferred value is the hardware revision identifier
           actually printed on the component itself (if present).";
        reference
          "RFC 6933: Entity MIB (Version 4) - entPhysicalHardwareRev";
      }
      leaf mfg-date {
        type yang:date-and-time;
        description
          "The date of manufacturing of the component.";
        reference
          "RFC 6933: Entity MIB (Version 4) - entPhysicalMfgDate";
      }
      leaf part-number {
        type string;
        description
          "The vendor-specific part number of the component
           type. It is expected that vendors assign unique part
           numbers to different component types within the
           scope of the vendor.";
      }
      leaf serial-number {
        type string;
        description
          "The vendor-specific serial number of the the entity
           (e.g., component) instance. It is expected that vendors
           assign unique serial numbers to different network element
           instances within the scope of the product name.";
      }
      leaf asset-id {
        type string;
        description
          "This node is an asset tracking identifier for the component,
           as specified by a network operator.

           A server implementation MAY map this leaf to the
           entPhysicalAssetID MIB object.  Such an
           implementation needs to use some mechanism to handle
           the differences in size and characters allowed
           between this leaf and entPhysicalAssetID.

           The definition of such a mechanism is outside the
           scope of this document.";
        reference
          "RFC 6933: Entity MIB (Version 4) -
                     entPhysicalAssetID";
```

```
      }
      leaf is-fru {
        type boolean;
        description
          "This node indicates whether or not this component is
           considered a 'field-replaceable unit' by the vendor.
           If this node contains the value 'true', then this
           component identifies a field-replaceable unit.
           For all components that are permanently contained
           within a field-replaceable unit, the value 'false'
           should be returned for this node.";
        reference
          "RFC 6933: Entity MIB (Version 4) -
                     entPhysicalIsFRU";
      }
      leaf-list uri {
        type inet:uri;
        description
          "This node contains identification information about
           the component.";
        reference
          "RFC 6933: Entity MIB (Version 4) - entPhysicalUris";
      }
    }

    /*
     * Data Nodes
     */

    container network-inventory {
      config false;
      description
        "Top-level container for network inventory.";
      container network-elements {
        description
          "The top-level container for the list of network elements
           within the network.";
        list network-element {
          key "ne-id";
          description
            "The list of network elements within the network.";
          leaf ne-id {
            type string;
            description
              "An identifier that uniquely identifies the NE in
               a network.";
          }
          leaf ne-type {
```

```
        type identityref {
          base nwi:ne-type;
        }
        default "nwi:ne-physical";
        description
          "The NE type.";
      }
      uses ne-component-common-entity-attributes;
      leaf product-rev {
        type string;
        description
          "The vendor-specific product revision string for the
           network-element.";
      }
      container components {
        description
          "The top-level container for the list of components
           within a network element.";
        list component {
          key "component-id";
          description
            "The list of components within a network element.";
          uses component-attributes;
          leaf parent-rel-pos {
            when 'count(../parent) < 2' {
              description
                "This data node is applicable only when this
                 component is contained in the network-element or
                 in only one parent component.";
            }
            type int32 {
              range "0 .. 2147483647";
            }
            description
              "The relative position with respect to the parent
               component among all the sibling components.";
            reference
              "RFC 6933: Entity MIB (Version 4) -
                         entPhysicalParentRelPos";
          }
          leaf-list parent {
            type leafref {
              path "../../component/component-id";
              require-instance false;
            }
            description
              "The identifiers of all the components that
               physically contain this component.
```

```
                    If this list is empty, this component is not
                    contained in any other component but it is contained
                    in the network-element.";
                  reference
                    "RFC 6933: Entity MIB (Version 4) -
                               entPhysicalContainedIn";
                }
                leaf is-main {
                  when "derived-from-or-self(../nwi:class, "
                    + "'ianahw:chassis')";
                  type boolean;
                  description
                    "This node indicates whether the chassis is taking or
                     not the 'main' role.

                     This node is applicable only to scenarios where
                     there are chassis components which can take the
                     'main' role (e.g., multi-chassis network elements),
                     otherwise it is omitted.";
                }
              }
            }
          }
        }
      }
    }
  }
  <CODE ENDS>
```

Figure 3: Network inventory YANG module

6. Operational Considerations

The network inventory YANG data model defined in the document is intended to report the actual inventory data that a network controller knows of the network elements and components actually installed within the network. Therefore, this data model provides a read-only perspective of the network inventory information.

As outlined in Section 1, per the definition of [RFC8969], the network inventory model is a network model.

This information can be provided by a network controller to an higher level hierarchical network controller, to an Inventory OSS or to any other type of application which needs to discover the network inventory information.

For example, in the context of ACTN, the network inventory YANG data model can be used at the MPI interfaces, as defined in [RFC8453], or on an interface, not defined in [RFC8453] between the MDSC and the Inventory OSS.

The information in the model is populated by controller by reading it from the devices using the device model supported by the devices. This model does not constraint the device models used on the device: the YANG data model defined in [RFC8348] is an option but other options (e.g., vendor specific interfaces or YANG data models) are also allowed. In case some information is not provided by the device, the network controller SHALL omit this information unless this information is known by other sources of information (e.g., through local configuration within the network controller).

In case of hierarchical controllers, a hierarchical network controller can also collect the network inventory information from its lower level network controllers using this YANG data model (or other mechanisms which are outside the scope of this document) and report the combined network inventory information to an an higher level network controller, to an Inventory OSS or to any other type of application which needs to discover the network inventory information.

7. Security Considerations

This section is modeled after the template described in Section 3.7 of [I-D.ietf-netmod-rfc8407bis].

The "ietf-network-inventory" YANG module defines a data model that is designed to be accessed via YANG-based management protocols, such as NETCONF [RFC6241] and RESTCONF [RFC8040]. These YANG-based management protocols (1) have to use a secure transport layer (e.g., SSH [RFC4252], TLS [RFC8446], and QUIC [RFC9000]) and (2) have to use mutual authentication.

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. Specifically, the following subtrees and data nodes have particular sensitivities/ vulnerabilities:

Specifically, the following subtrees and data nodes have particular sensitivities/vulnerabilities:

* "/nwi:network-elements"

  This subtree reports the inventory information for all the network elements and their hardware components deployed within the network as well as of the software modules being active on these network elements and components. Unauthorized access to this subtree can disclose this information. A malicious attacker can use this information to perform targeted attacks to network elements, hardware components or software modules with known vulnerabilities.

Modules that use the groupings that are defined in this document should identify the corresponding security considerations. For example, reusing the 'component-attributes' grouping may expose sensitive information.

8. IANA Considerations

IANA is requested to register the following URI in the "ns" registry within the "IETF XML Registry" group [RFC3688]:

```
URI: urn:ietf:params:xml:ns:ietf-network-inventory
Registrant Contact: The IESG
XML: N/A; the requested URI is an XML namespace.
```

IANA is requested to register the following YANG module in the "YANG Module Names" registry [RFC6020] within the "YANG Parameters" registry group.

```
name:         ietf-network-inventory
namespace:    urn:ietf:params:xml:ns:yang:ietf-network-inventory
prefix:       nwi
reference:    RFC XXXX
```

9. References

9.1. Normative References

[IANA_ENTITY_MIB]
IANA, "IANA-ENTITY-MIB", n.d.,
<https://www.iana.org/assignments/ianaentity-mib/ianaentity-mib.xhtml>.

[IANA_HW_YANG]
IANA, "iana-hardware YANG Module", n.d., <https://www.iana.org/assignments/iana-hardware/iana-hardware.xhtml>.

[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>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/rfc/rfc6020>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/rfc/rfc6241>.

[RFC6933] Bierman, A., Romascanu, D., Quittek, J., and M. Chandramouli, "Entity MIB (Version 4)", RFC 6933, DOI 10.17487/RFC6933, May 2013, <https://www.rfc-editor.org/rfc/rfc6933>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/rfc/rfc6991>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/rfc/rfc7950>.

[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>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/rfc/rfc8341>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/rfc/rfc8342>.

[RFC8348] Bierman, A., Bjorklund, M., Dong, J., and D. Romascanu, "A YANG Data Model for Hardware Management", RFC 8348, DOI 10.17487/RFC8348, March 2018, <https://www.rfc-editor.org/rfc/rfc8348>.

[TMF_SD2-20] TM Forum, "SD2-20_Equipment Model", TMF MTOSI 4.0, Network Resource Fulfilment (NRF), SD2-20 , May 2008, <https://www.tmforum.org/resources/suite/mtosi-4-0/>.

9.2. Informative References

[I-D.ietf-ivy-network-inventory-location] Wu, B., Belotti, S., Bouquier, J., Peruzzini, F., and P. Bedard, "A YANG Data Model for Network Inventory Location", Work in Progress, Internet-Draft, draft-ietf-ivy-network-inventory-location-03, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-ivy-network-inventory-location-03>.

[I-D.ietf-ivy-network-inventory-software] Wu, B., Zhou, C., Wu, Q., and M. Boucadair, "A YANG Network Data Model of Network Inventory Software Extensions", Work in Progress, Internet-Draft, draft-ietf-ivy-network-inventory-software-01, 10 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-ivy-network-inventory-software-01>.

[I-D.ietf-netmod-rfc8407bis] Bierman, A., Boucadair, M., and Q. Wu, "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", Work in Progress, Internet-Draft, draft-ietf-netmod-rfc8407bis-28, 5 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-netmod-rfc8407bis-28>.

[I-D.ietf-teas-actn-poi-applicability] Peruzzini, F., Bouquier, J., Busi, I., King, D., and D. Ceccarelli, "Applicability of Abstraction and Control of Traffic Engineered Networks (ACTN) to Packet Optical Integration (POI)", Work in Progress, Internet-Draft, draft-ietf-teas-actn-poi-applicability-16, 9 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-teas-actn-poi-applicability-16>.

[I-D.ygb-ivy-passive-network-inventory] Yu, C., Guo, A., Busi, I., Boroon, M., Belotti, S., van caenegem, T., S., S. 1., B., S., Davis, N., Tilocca, M.,

Peters, B., Yoon, B. Y., LIUYUCONG, Zhao, Y., and A. Sakalabhaktula, "A YANG Data Model for Passive Network Inventory", Work in Progress, Internet-Draft, draft-ygb-ivy-passive-network-inventory-02, 7 July 2025, <https://datatracker.ietf.org/doc/html/draft-ygb-ivy-passive-network-inventory-02>.

[RFC1157] Case, J., Fedor, M., Schoffstall, M., and J. Davin, "Simple Network Management Protocol (SNMP)", RFC 1157, DOI 10.17487/RFC1157, May 1990, <https://www.rfc-editor.org/rfc/rfc1157>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/rfc/rfc3688>.

[RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol", RFC 4252, DOI 10.17487/RFC4252, January 2006, <https://www.rfc-editor.org/rfc/rfc4252>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/rfc/rfc7951>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/rfc/rfc8040>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/rfc/rfc8340>.

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

[RFC8453] Ceccarelli, D., Ed. and Y. Lee, Ed., "Framework for Abstraction and Control of TE Networks (ACTN)", RFC 8453, DOI 10.17487/RFC8453, August 2018, <https://www.rfc-editor.org/rfc/rfc8453>.

[RFC8969] Wu, Q., Ed., Boucadair, M., Ed., Lopez, D., Xie, C., and L. Geng, "A Framework for Automating Service and Network Management with YANG", RFC 8969, DOI 10.17487/RFC8969, January 2021, <https://www.rfc-editor.org/rfc/rfc8969>.

[RFC9000] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

Appendix A. Comparison With Openconfig-platform Data Model

Since more and more devices can be managed by domain controller through OpenConfig, to ensure that our inventory data model can cover these devices' inventory data, we have compared our inventory data model with the "openconfig-platform" model which is the data model used to manage inventory information in OpenConfig.

Openconfig-platform data model is NE-level and uses a generic component concept to describe its inner devices and containers, which is similar to "ietf-hardware" model in [RFC8348]. Since we have also reused the component concept of [RFC8348] in our inventory data model, we can compare the component's attributes between "openconfig-platform" and our model directly , which is stated below:

Attributes in oc-platform	Attributes in our model	remark
name	name	
type	class	
id	uuid	
location	location	
description	description	
mfg-name	mfg-name	
mfg-date	mfg-date	
hardware-version	hardware-rev	
firmware-version	firmware-rev	
software-version	software-rev	
serial-no	serial-num	
part-no	part-number	

clei-code		TBD
removable	is-fru	
oper-status		state data
empty	contained-child?	If there is no contained child, it is empty.
parent	parent-references	
redundant-role		TBD
last-switchover-reason		state data
last-switchover-time		state data
last-reboot-reason		state data
last-reboot-time		state data
switchover-ready		state data
temperature		performance data
memory		performance data
allocated-power		TBD
used-power		TBD
pcie		alarm data
properties		TBD
subcomponents		
chassis		
port		
power-supply		TBD
fan		Fan is considered as a specific board. And no need to define as a

			single component
fabric			TBD
storage			For Optical and IP technology, no need to manage storage on network element
cpu			For Optical and IP technology, no need to manage CPU on network element
integrated-circuit			
backplane			Backplane is considered as a part of board. And no need to define as a single component
software-module			TBD
controller-card			Controller card is considered as a specific functional board. And no need to define as a single component

Table 2: Comparison between openconfig platform and inventory data models

As it mentioned in Section 3.3 that state data and performance data are out of scope of our data model, it is same for alarm data and it should be defined in some other alarm data models separately. And for some component specific structures in "openconfig-platform", we consider some of them can be contained by our existing structure, such as fan, backplane, and controller-card, while some others do not need to be included in this network inventory model like storage and cpu.

Mostly, our inventory data model can cover the attributes from OpenConfig.

Appendix B. Terminology of Container

Within this document , with the term "container" we consider an hardware component class capable of containing one or more removable physical entities, e.g. a slot in a chassis is containing a board.

terminology of IVY base model	terminology in other model
container	holder

Table 3: terminology mapping

Appendix C. Efficiency Issue

During the integration with OSS in some operators, some efficiency/ scalability concerns have been discovered when synchronizing network inventory data for big networks. More discussions are needed to address these concerns.

Considering that relational databases are widely used by traditional OSS systems and also by some network controllers, the inventory objects are most likely to be saved in different tables. With the model defined in current draft, when doing a full synchronization, network controller needs to convert all inventory objects of each NE into component objects and combine them together into a single list, and then construct a response and send to OSS or MDSC. The OSS or MDSC needs to classify the component list and divide them into different groups, in order to save them in different tables. The combining-regrouping steps are impacting the network controller & OSS/MDSC processing, which may result in efficiency/scalability limitations in large scale networks.

An alternative YANG model structure, which defines the inventory objects directly, instead of defining generic components, has also been analyzed. However, also with this model, there still could be some scalability limitations when synchronizing full inventory resources in large scale of networks. This scalability limitation is caused by the limited transmission capabilities of HTTP protocol. We think that this scalability limitation should be solved at protocol level rather than data model level.

The model proposed by this draft is designed to be as generic as possible so to cover future special types of inventory objects that could be used in other technologies, that have not been identified yet. If the inventory objects were to be defined directly with fixed hierarchical relationships in YANG model, this new type of inventory

objects needs to be manually defined, which is not a backward compatible change and therefore is not an acceptable approach for implementation. With a generic model, it is only needed to augment a new component class and extend some specific attributes for this new inventory component class, which is more flexible. We consider that this generic data model, enabling a flexible and backward compatible approach for other technologies, represents the main scope of this draft. Solution description to efficiency/scalability limitations mentioned above is considered as out-of-scope.

Appendix D. Examples of ports

This appendix provides some examples of port implementations and how they can be modelled using the "ietf-network-inventory" module defined in Section 5.

Figure 4 shows an example of a single board which contains three type of ports:

1. An integrated port (non pluggable);

2. An empty port;

3. A pluggable port

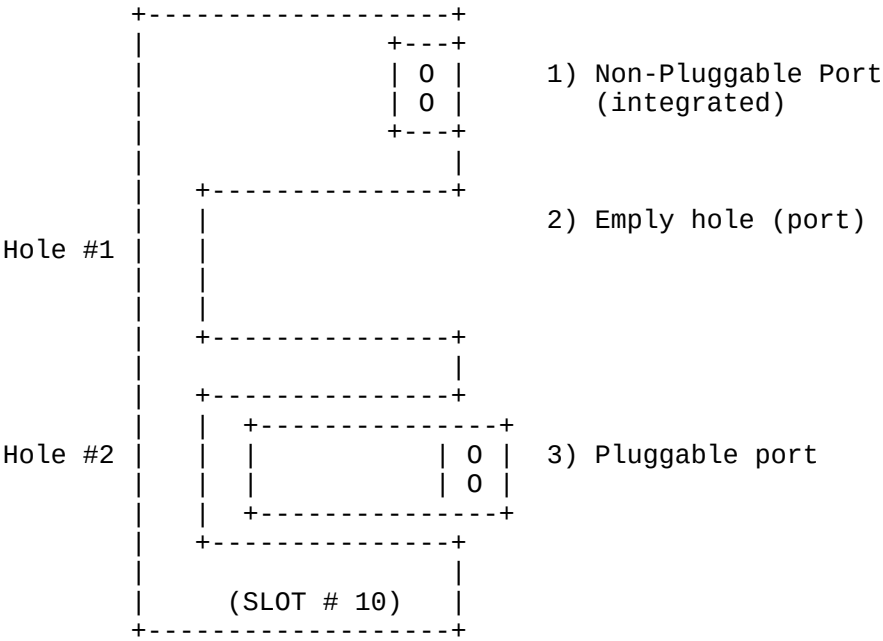

Figure 4: Example of a board with different types of ports

D.1. JSON Examples

This appendix contains an example of an instance data tree in JSON encoding [RFC7951], instantiating the "ietf-network-inventory" module to describe the three types of ports on a single board, as shown in Figure 4.


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

{
  "ietf-network-inventory:network-inventory": {
    "network-elements": {
      "network-element" : [
        {
          "ne-id": "NE-1",
          "description": "Network element example with ports and \
                                                          breakouts.",
          "components": {
            "component": [
              {
                "component-id": "board-1",
                "class": "iana-hardware:module",
                "description": "Network element example with ports \
                                                    and breakouts."
              },
              {
                "component-id": "port-1",
                "class": "iana-hardware:port",
                "description": "Example of an integrated (non-\
                                                 pluggable) port.",
                "parent": [
                  "board-1"
                ],
                "parent-rel-pos": 1
              },
              {
                "component-id": "port-2",
                "class": "iana-hardware:port",
                "description": "Example of an empty pluggable port.",
                "parent": [
                  "board-1"
                ],
                "parent-rel-pos": 2
              },
              {
                "component-id": "port-3",
```

```
                    "class": "iana-hardware:port",
                    "description": "Example of a non empty pluggable \
                                                                port.",
                    "parent": [
                      "board-1"
                    ],
                    "parent-rel-pos": 3
                  },
                  {
                    "component-id": "transceiver-module-3",
                    "class": "iana-hardware:module",
                    "description": "Example of a pluggable module \
                                    plugged within a pluggable port.",
                    "parent": [
                      "port-3"
                    ],
                    "is-fru": true
                  }
                ]
              }
            }
          ]
        }
      }
    }
```

Appendix E. Example of multi-chassis network elements

This appendix provides some examples of multi-chassis network elements and how they can be modelled using the "ietf-network-inventory" module defined in Section 5.

Multi-chassis network elements are network elements composed by two or more chassis interconnected, in principle, with any topology.

Stacked switches are an example of multi-chassis which consist of multiple standalone switches that are interconnected through dedicated stack ports and cables and managed as a single logical unit. Stacked switch: - are connected using a daisy-chain or a ring topology - are managed using a single IP Address - synchronized software-upgrade - use Priority/MAC-Addr(s) decide Master/Members selection and communication.

Figure 5 and Figure 6 describe two examples of stacked switch with three stacked switches (pizza boxes) connected in a daisy-chain or ring topology.

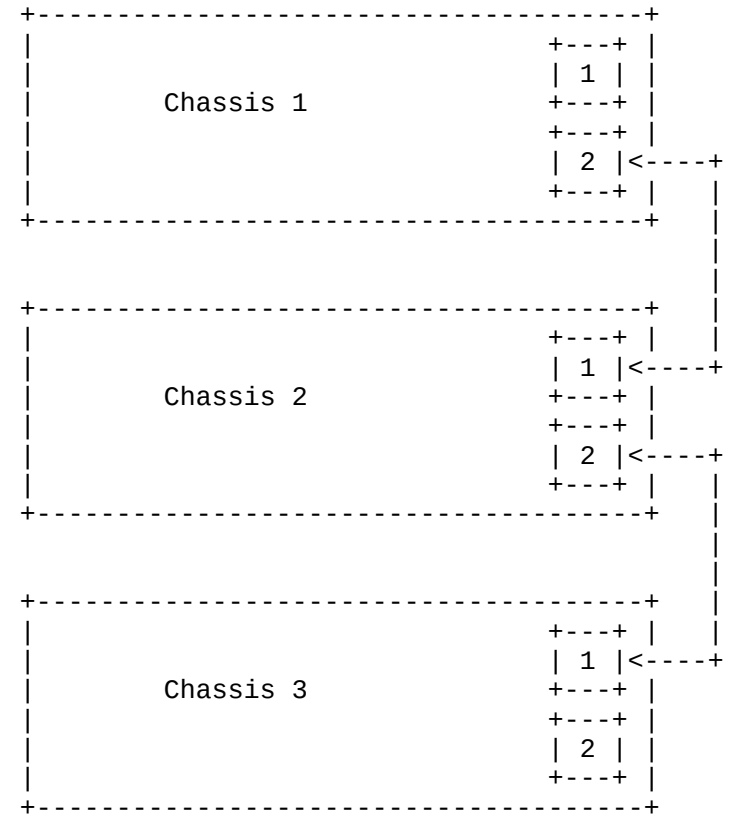

Figure 5: Example of a stacked switch in a daisy chain topology

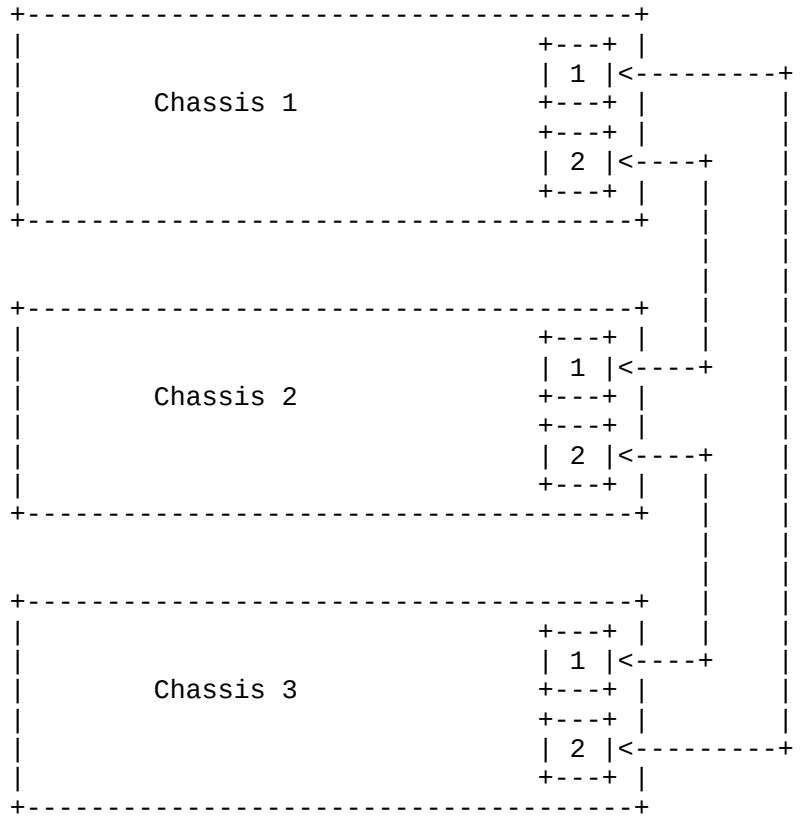

Figure 6: Example of a stacked switch in a ring topology

Using the base network inventory YANG data model, each stackable switch can be modelled as a chassis within the same network element, which models the stacked switch. The stack ports are modelled like other ports. The stack cables are not reported using the base network inventory YANG data model but can be reported using the passive network inventory YANG data model under definition in [I-D.ygb-ivy-passive-network-inventory].

Cascaded switches are another example of multi-chassis which consist of multiple standalone switches that are interconnected and managed as a single logical unit. Cascaded switch: - are usually connected in a tree topology - are managed using a single IP Address - the root of the tree is configured as Master.

Figure 7 describe an example of cascaded switch with three chassis connected in a tree topology.

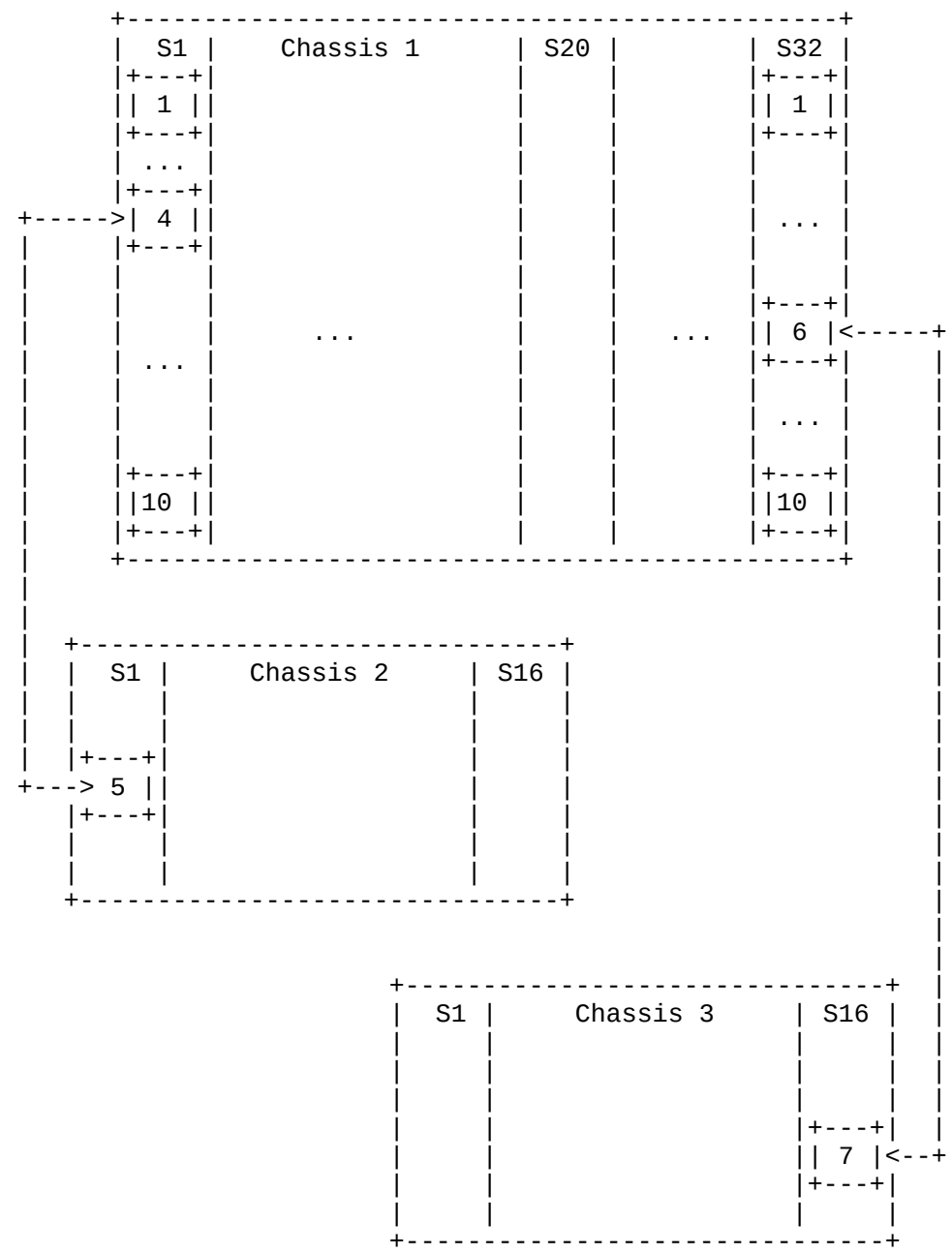

Figure 7: Example of a cascaded switch in a tree topology

Using the base network inventory YANG data model each interconnected switch can be modelled as a chassis within the same network element, which models the cascaded switch. The ports used to interconnect the different chassis are normal (traffic) ports and modelled like other ports. The interconnecting cables are not reported using the base network inventory YANG data model but can be reported using the passive network inventory model under definition in [I-D.ygb-ivy-passive-network-inventory].

E.1. JSON Examples

This appendix contains an example of an instance data tree in JSON encoding [RFC7951], instantiating the "ietf-network-inventory" model to describe the three examples of multi-chassis NEs, as shown in Figure 5, Figure 6 and Figure 7.

> Note: the base inventory model allows reporting only the chassis and ports configuration. Reporting the link between the chassis of the same NE is outside the scope of the base inventory model. The YANG data model under definition in [I-D.ygb-ivy-passive-network-inventory] as an augmentation of the base inventory YANG data model can be used to provide this additional information.


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

{
  "ietf-network-inventory:network-inventory": {
    "network-elements": {
      "network-element" : [
        {
          "ne-id": "NE-1",
          "description": "Stack Switch in a daisy chain topology.",
          "components": {
            "component": [
              {
                "component-id": "chassis-1",
                "class": "iana-hardware:chassis",
                "description": "First switch of the stack.",
                "parent-rel-pos": 1,
                "is-fru": true
              },
              {
                "component-id": "port-1-1",
                "class": "iana-hardware:port",
                "description": "First stack port of the first \
                                              switch in the stack.",
                "parent": [
```

```
        "chassis-1"
      ],
      "parent-rel-pos": 1
    },
    {
      "component-id": "port-1-2",
      "class": "iana-hardware:port",
      "description": "Second stack port of the first \
                                      switch in the stack.",
      "parent": [
        "chassis-1"
      ],
      "parent-rel-pos": 2
    },
    {
      "component-id": "transceiver-module-1-2",
      "class": "iana-hardware:module",
      "description": "Transceiver module plugged in the \
      second stack port of the first switch in the stack.",
      "parent": [
        "port-1-2"
      ],
      "is-fru": true
    },
    {
      "component-id": "chassis-2",
      "class": "iana-hardware:chassis",
      "description": "Second switch of the stack.",
      "parent-rel-pos": 2,
      "is-fru": true
    },
    {
      "component-id": "port-2-1",
      "class": "iana-hardware:port",
      "description": "First stack port of the second \
                                      switch in the stack.",
      "parent": [
        "chassis-2"
      ],
      "parent-rel-pos": 1
    },
    {
      "component-id": "transceiver-module-2-1",
      "class": "iana-hardware:module",
      "description": "Transceiver module plugged in the \
       first stack port of the first switch in the stack.",
      "parent": [
        "port-2-1"
```

```
        ],
        "is-fru": true
      },
      {
        "component-id": "port-2-2",
        "class": "iana-hardware:port",
        "description": "Second stack port of the second \
                                        switch in the stack.",
        "parent": [
          "chassis-2"
        ],
        "parent-rel-pos": 2
      },
      {
        "component-id": "transceiver-module-2-2",
        "class": "iana-hardware:module",
        "description": "Transceiver module plugged in the \
        second stack port of the first switch in the stack.",
        "parent": [
          "port-2-2"
        ],
        "is-fru": true
      },
      {
        "component-id": "chassis-3",
        "class": "iana-hardware:chassis",
        "description": "Third switch of the stack.",
        "parent-rel-pos": 3,
        "is-fru": true
      },
      {
        "component-id": "port-3-1",
        "class": "iana-hardware:port",
        "description": "First stack port of the third \
                                        switch in the stack.",
        "parent": [
          "chassis-3"
        ],
        "parent-rel-pos": 1
      },
      {
        "component-id": "transceiver-module-3-1",
        "class": "iana-hardware:module",
        "description": "Transceiver module plugged in the \
         first stack port of the third switch in the stack.",
        "parent": [
          "port-3-1"
        ],
```

```
                "is-fru": true
              },
              {
                "component-id": "port-3-2",
                "class": "iana-hardware:port",
                "description": "Second stack port of the second \
                                              switch in the stack.",
                "parent": [
                  "chassis-3"
                ],
                "parent-rel-pos": 2
              }
            ]
          }
        },
        {
          "ne-id": "NE-2",
          "description": "Stack Switch in a ring topology.",
          "components": {
            "component": [
              {
                "component-id": "chassis-1",
                "class": "iana-hardware:chassis",
                "description": "First switch of the stack.",
                "parent-rel-pos": 1,
                "is-fru": true
              },
              {
                "component-id": "port-1-1",
                "class": "iana-hardware:port",
                "description": "First stack port of the first \
                                              switch in the stack.",
                "parent": [
                  "chassis-1"
                ],
                "parent-rel-pos": 1
              },
              {
                "component-id": "transceiver-module-1-1",
                "class": "iana-hardware:module",
                "description": "Transceiver module plugged in the \
                 first stack port of the first switch in the stack.",
                "parent": [
                  "port-1-1"
                ],
                "is-fru": true
              },
              {
```

```
                  "component-id": "port-1-2",
                  "class": "iana-hardware:port",
                  "description": "Second stack port of the first \
                                                  switch in the stack.",
                  "parent": [
                    "chassis-1"
                  ],
                  "parent-rel-pos": 2
                },
                {
                  "component-id": "transceiver-module-1-2",
                  "class": "iana-hardware:module",
                  "description": "Transceiver module plugged in the \
                  second stack port of the first switch in the stack.",
                  "parent": [
                    "port-1-2"
                  ],
                  "is-fru": true
                },
                {
                  "component-id": "chassis-2",
                  "class": "iana-hardware:chassis",
                  "description": "Second switch of the stack.",
                  "parent-rel-pos": 2,
                  "is-fru": true
                },
                {
                  "component-id": "port-2-1",
                  "class": "iana-hardware:port",
                  "description": "First stack port of the second \
                                                  switch in the stack.",
                  "parent": [
                    "chassis-2"
                  ],
                  "parent-rel-pos": 1
                },
                {
                  "component-id": "transceiver-module-2-1",
                  "class": "iana-hardware:module",
                  "description": "Transceiver module plugged in the \
                   first stack port of the first switch in the stack.",
                  "parent": [
                    "port-2-1"
                  ],
                  "is-fru": true
                },
                {
                  "component-id": "port-2-2",
```

```
      "class": "iana-hardware:port",
      "description": "Second stack port of the second \
                                      switch in the stack.",
      "parent": [
        "chassis-2"
      ],
      "parent-rel-pos": 2
    },
    {
      "component-id": "transceiver-module-2-2",
      "class": "iana-hardware:module",
      "description": "Transceiver module plugged in the \
      second stack port of the first switch in the stack.",
      "parent": [
        "port-2-2"
      ],
      "is-fru": true
    },
    {
      "component-id": "chassis-3",
      "class": "iana-hardware:chassis",
      "description": "Third switch of the stack.",
      "parent-rel-pos": 3,
      "is-fru": true
    },
    {
      "component-id": "port-3-1",
      "class": "iana-hardware:port",
      "description": "First stack port of the third \
                                      switch in the stack.",
      "parent": [
        "chassis-3"
      ],
      "parent-rel-pos": 1
    },
    {
      "component-id": "transceiver-module-3-1",
      "class": "iana-hardware:module",
      "description": "Transceiver module plugged in the \
       first stack port of the third switch in the stack.",
      "parent": [
        "port-3-1"
      ],
      "is-fru": true
    },
    {
      "component-id": "port-3-2",
      "class": "iana-hardware:port",
```

```
                "description": "Second stack port of the second \
                                                switch in the stack.",
                "parent": [
                  "chassis-3"
                ],
                "parent-rel-pos": 2
              },
              {
                "component-id": "transceiver-module-3-2",
                "class": "iana-hardware:module",
                "description": "Transceiver module plugged in the \
                second stack port of the third switch in the stack.",
                "parent": [
                  "port-3-2"
                ],
                "is-fru": true
              }
            ]
          }
        },
        {
          "ne-id": "NE-3",
          "description": "Cascaded Switch in a tree topology.",
          "components": {
            "component": [
              {
                "component-id": "chassis-1",
                "class": "iana-hardware:chassis",
                "description": "First chassis of the cascaded switch\
                                                                    .",
                "parent-rel-pos": 1,
                "is-fru": true
              },
              {
                "component-id": "slot-1-1",
                "class": "iana-hardware:container",
                "description": "Slot 1 of the first chassis of the \
                                                cascaded switch.",
                "parent": [
                  "chassis-1"
                ],
                "parent-rel-pos": 1
              },
              {
                "component-id": "card-1-1",
                "class": "iana-hardware:module",
                "description": "Card plugged into slot 1 of the \
                              first chassis of the cascaded switch.",
```

```
                  "parent": [
                    "slot-1-1"
                  ],
                  "is-fru": true
                },
                {
                  "component-id": "port-1-1-1",
                  "class": "iana-hardware:port",
                  "description": "Empty port 1 on the card plugged \
             into slot 1 of the first chassis of the cascaded switch.",
                  "parent": [
                    "card-1-1"
                  ],
                  "parent-rel-pos": 1
                },
                {
                  "component-id": "port-1-1-4",
                  "class": "iana-hardware:port",
                  "description": "Pluggable port 4 on the card \
       plugged into slot 1 of the first chassis of the cascaded switch.",
                  "parent": [
                    "card-1-1"
                  ],
                  "parent-rel-pos": 4
                },
                {
                  "component-id": "transceiver-module-1-1-4",
                  "class": "iana-hardware:module",
                  "description": "Transceiver module plugged in port \
     4 on the card plugged into slot 1 of the first chassis of the \
                                                    cascaded switch.",
                  "parent": [
                    "port-1-1-4"
                  ],
                  "is-fru": true
                },
                {
                  "component-id": "port-1-1-10",
                  "class": "iana-hardware:port",
                  "description": "Empty port 10 on the card plugged \
             into slot 1 of the first chassis of the cascaded switch.",
                  "parent": [
                    "card-1-1"
                  ],
                  "parent-rel-pos": 10
                },
                {
                  "component-id": "slot-1-20",
```

```
                  "class": "iana-hardware:container",
                  "description": "Empty slot 20 of the first chassis \
                                                of the cascaded switch.",
                  "parent": [
                    "chassis-1"
                  ],
                  "parent-rel-pos": 20
                },
                {
                  "component-id": "slot-1-32",
                  "class": "iana-hardware:container",
                  "description": "Slot 32 of the first chassis of the \
                                                    cascaded switch.",
                  "parent": [
                    "chassis-1"
                  ],
                  "parent-rel-pos": 32
                },
                {
                  "component-id": "card-1-32",
                  "class": "iana-hardware:module",
                  "description": "Card plugged into slot 32 of the \
                                  first chassis of the cascaded switch.",
                  "parent": [
                    "slot-1-32"
                  ],
                  "is-fru": true
                },
                {
                  "component-id": "port-1-32-1",
                  "class": "iana-hardware:port",
                  "description": "Empty port 1 on the card plugged \
          into slot 32 of the first chassis of the cascaded switch.",
                  "parent": [
                    "card-1-32"
                  ],
                  "parent-rel-pos": 1
                },
                {
                  "component-id": "port-1-32-6",
                  "class": "iana-hardware:port",
                  "description": "Pluggable port 6 on the card \
    plugged into slot 32 of the first chassis of the cascaded switch.",
                  "parent": [
                    "card-1-32"
                  ],
                  "parent-rel-pos": 6
                },
```

```
                {
                  "component-id": "transceiver-module-1-32-6",
                  "class": "iana-hardware:module",
                  "description": "Transceiver module plugged in port \
   6 on the card plugged into slot 32 of the first chassis of the \
                                                  cascaded switch.",
                  "parent": [
                    "port-1-32-6"
                  ],
                  "is-fru": true
                },
                {
                  "component-id": "port-1-32-10",
                  "class": "iana-hardware:port",
                  "description": "Empty port 10 on the card plugged \
           into slot 32 of the first chassis of the cascaded switch.",
                  "parent": [
                    "card-1-32"
                  ],
                  "parent-rel-pos": 10
                },
                {
                  "component-id": "chassis-2",
                  "class": "iana-hardware:chassis",
                  "description": "Second chassis of the cascaded \
                                                          switch.",
                  "parent-rel-pos": 2,
                  "is-fru": true
                },
                {
                  "component-id": "slot-2-1",
                  "class": "iana-hardware:container",
                  "description": "Slot 1 of the second chassis of the \
                                                  cascaded switch.",
                  "parent": [
                    "chassis-2"
                  ],
                  "parent-rel-pos": 1
                },
                {
                  "component-id": "card-2-1",
                  "class": "iana-hardware:module",
                  "description": "Card plugged into slot 1 of the \
                                  second chassis of the cascaded switch.",
                  "parent": [
                    "slot-2-1"
                  ],
                  "is-fru": true
```

```
                },
                {
                  "component-id": "port-2-1-5",
                  "class": "iana-hardware:port",
                  "description": "Pluggable port 5 on the card \
     plugged into slot 1 of the second chassis of the cascaded switch.",
                  "parent": [
                    "card-2-1"
                  ],
                  "parent-rel-pos": 4
                },
                {
                  "component-id": "transceiver-module-2-1-5",
                  "class": "iana-hardware:module",
                  "description": "Transceiver module plugged in port \
   5 on the card plugged into slot 1 of the second chassis of the \
                                                  cascaded switch.",
                  "parent": [
                    "port-2-1-5"
                  ],
                  "is-fru": true
                },
                {
                  "component-id": "chassis-3",
                  "class": "iana-hardware:chassis",
                  "description": "Third chassis of the cascaded switch\
                                                                  .",
                  "parent-rel-pos": 3,
                  "is-fru": true
                },
                {
                  "component-id": "slot-3-1",
                  "class": "iana-hardware:container",
                  "description": "Empty slot 1 of the third chassis \
                                       of the cascaded switch.",
                  "parent": [
                    "chassis-3"
                  ],
                  "parent-rel-pos": 1
                },
                {
                  "component-id": "slot-3-16",
                  "class": "iana-hardware:container",
                  "description": "Slot 16 of the third chassis of the \
                                                  cascaded switch.",
                  "parent": [
                    "chassis-3"
                  ],
```

```
                  "parent-rel-pos": 16
                },
                {
                  "component-id": "card-3-16",
                  "class": "iana-hardware:module",
                  "description": "Card plugged into slot 16 of the \
                                  third chassis of the cascaded switch.",
                  "parent": [
                    "slot-3-16"
                  ],
                  "is-fru": true
                },
                {
                  "component-id": "port-3-16-7",
                  "class": "iana-hardware:port",
                  "description": "Pluggable port 7 on the card \
    plugged into slot 16 of the third chassis of the cascaded switch.",
                  "parent": [
                    "card-3-16"
                  ],
                  "parent-rel-pos": 7
                },
                {
                  "component-id": "transceiver-module-3-16-7",
                  "class": "iana-hardware:module",
                  "description": "Transceiver module plugged in port \
   7 on the card plugged into slot 16 of the third chassis of the \
                                                  cascaded switch.",
                  "parent": [
                    "port-3-16-7"
                  ],
                  "is-fru": true
                }
              ]
            }
          }
        ]
      }
    }
  }
```

Appendix F. Example of non-modular network elements

This appendix provides some examples of non-modular network elements and how they can be modelled using the "ietf-network-inventory" module defined in Section 5.

Non-modular network elements (also known as "pizza boxes") are network elements composed by a single chassis (with a 1U horizontal oriented rectangular shape) as a self-contained system. A non-modular network element does not have any slots to take cards so it cannot take any non-field replaceable modules other than pluggable ports.

Using the base network inventory YANG data model a non-modular network element can be modelled as a network element containing only one chassis and ports (as child components of the chassis).

Reporting the single chassis component within a non-modular network element is required because the chassis component is the type of component which provides the physical characteristics of the network element chassis (the network element is defined just as an assembly of components) and its location, using the network inventory YANG data model under definition in [I-D.ietf-ivy-network-inventory-location].

F.1. JSON Examples

This appendix contains an example of an instance data tree in JSON encoding [RFC7951], instantiating the "ietf-network-inventory" module to describe an example of a non-modular network element.

```
{
  "ietf-network-inventory:network-inventory": {
    "network-elements": {
      "network-element" : [
        {
          "ne-id": "Pizza-box-NE",
          "description": "Example of a pizza box NE.",
          "components": {
            "component": [
              {
                "component-id": "pizza-chassis",
                "class": "iana-hardware:chassis",
                "description": "Pizza box chassis.",
                "is-fru": true
              },
              {
                "component-id": "port-1",
                "class": "iana-hardware:port",
                "description": "Port 1 of the pizza box.",
                "parent": [
                  "pizza-chassis"
                ],
                "parent-rel-pos": 1
```

```
      },
      {
        "component-id": "port-2",
        "class": "iana-hardware:port",
        "description": "Port 2 of the pizza box.",
        "parent": [
          "pizza-chassis"
        ],
        "parent-rel-pos": 2
      },
      {
        "component-id": "port-3",
        "class": "iana-hardware:port",
        "description": "Port 3 of the pizza box.",
        "parent": [
          "pizza-chassis"
        ],
        "parent-rel-pos": 3
      },
      {
        "component-id": "port-4",
        "class": "iana-hardware:port",
        "description": "Port 4 of the pizza box.",
        "parent": [
          "pizza-chassis"
        ],
        "parent-rel-pos": 4
      },
      {
        "component-id": "port-5",
        "class": "iana-hardware:port",
        "description": "Port 5 of the pizza box.",
        "parent": [
          "pizza-chassis"
        ],
        "parent-rel-pos": 5
      },
      {
        "component-id": "port-6",
        "class": "iana-hardware:port",
        "description": "Port 6 of the pizza box.",
        "parent": [
          "pizza-chassis"
        ],
        "parent-rel-pos": 6
      },
      {
        "component-id": "port-7",
```

```
                    "class": "iana-hardware:port",
                    "description": "Port 7 of the pizza box.",
                    "parent": [
                      "pizza-chassis"
                    ],
                    "parent-rel-pos": 7
                  },
                  {
                    "component-id": "port-8",
                    "class": "iana-hardware:port",
                    "description": "Port 8 of the pizza box.",
                    "parent": [
                      "pizza-chassis"
                    ],
                    "parent-rel-pos": 8
                  },
                  {
                    "component-id": "port-9",
                    "class": "iana-hardware:port",
                    "description": "Port 9 of the pizza box.",
                    "parent": [
                      "pizza-chassis"
                    ],
                    "parent-rel-pos": 9
                  },
                  {
                    "component-id": "port-10",
                    "class": "iana-hardware:port",
                    "description": "Port 10 of the pizza box.",
                    "parent": [
                      "pizza-chassis"
                    ],
                    "parent-rel-pos": 10
                  }
                ]
              }
            }
          ]
        }
      }
    }
```

Acknowledgments

The authors of this document would like to thank the authors of [I-D.ietf-teas-actn-poi-applicability] for having identified the gap and requirements to trigger this work.

This document was prepared using kramdown.

Contributors

Italo Busi
Huawei Technologies
Email: italo.busi@huawei.com

Aihua Guo
Futurewei Technologies
Email: aihuaguo.ietf@gmail.com

Victor Lopez
Nokia
Email: victor.lopez@nokia.com

Bo Wu
Huawei Technologies
Email: lana.wubo@huawei.com

Chenfang Zhang
China Unicom
Email: zhangcf80@chinaunicom.cn

Oscar Gonzalez de Dios
Telefonica
Email: oscar.gonzalezdedios@telefonica.com

Nigel Davis
Ciena
Email: ndavis@ciena.com

Authors' Addresses

Chaode Yu
Huawei Technologies
Email: yuchaode@huawei.com

Sergio Belotti
Nokia

Email: sergio.belotti@nokia.com

Jean-Francois Bouquier
Vodafone
Email: jeff.bouquier@vodafone.com

Fabio Peruzzini
FiberCop
Email: fabio.peruzzini@fibercop.com

Phil Bedard
Cisco
Email: phbedard@cisco.com