

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 26, 2018

A. Bierman
YumaWorks
M. Bjorklund
Tail-f Systems
J. Dong
Huawei Technologies
D. Romascanu
January 22, 2018

**A YANG Data Model for Hardware Management
draft-ietf-netmod-entity-08**

Abstract

This document defines a YANG data model for the management of hardware on a single server.

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 <http://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 July 26, 2018.

Copyright Notice

Copyright (c) 2018 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in [Section 4](#).e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	2
1.1.	Terminology	2
1.2.	Tree Diagrams	3
2.	Objectives	3
3.	Hardware Data Model	3
3.1.	The Components Lists	5
4.	Relationship to ENTITY-MIB	5
5.	Relationship to ENTITY-SENSOR-MIB	6
6.	Relationship to ENTITY-STATE-MIB	7
7.	Hardware YANG Module	7
8.	IANA Considerations	35
8.1.	URI Registrations	35
8.2.	YANG Module Registrations	36
9.	Security Considerations	36
10.	Acknowledgments	37
11.	References	37
11.1.	Normative References	37
11.2.	Informative References	39
Appendix A.	Hardware State Data Model	39
A.1.	Hardware State YANG Module	40
	Authors' Addresses	55

[1.](#) Introduction

This document defines a YANG [[RFC7950](#)] data model for the management of hardware on a single server.

The data model includes configuration and system state (status information and counters for the collection of statistics).

The data model in this document is designed to be compliant with the Network Management Datastore Architecture (NMDA) [[I-D.ietf-netmod-revised-datastores](#)]. For implementations that do not yet support NMDA, a temporary module with system state data only is defined in [Appendix A](#).

[1.1.](#) Terminology

The keywords "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.

The following terms are defined in [\[I-D.ietf-netmod-revised-datastores\]](#) and are not redefined here:

- o client
- o server
- o configuration
- o system state
- o operational state
- o intended configuration

[1.2.](#) Tree Diagrams

Tree diagrams used in this document follow the notation defined in [\[I-D.ietf-netmod-yang-tree-diagrams\]](#).

[2.](#) Objectives

This section describes some of the design objectives for the hardware model.

- o There are many common properties used to identify hardware components, which need to be supported in the hardware data model.
- o There are many important information and states about the components, which needs to be collected from the devices which support the hardware data model.
- o The hardware data model should be suitable for new implementations to use as is.
- o The hardware data model defined in this document can be implemented on a system that also implements ENTITY-MIB, thus the mapping between the hardware data model and ENTITY-MIB should be clear.
- o The data model should support pre-provisioning of hardware components.

[3.](#) Hardware Data Model

This document defines the YANG module "ietf-hardware", which has the following structure:


```
module: ietf-hardware
  +--rw hardware
    +--ro last-change?   yang:date-and-time
    +--rw component* [name]
      +--rw name          string
      +--rw class          identityref
      +--ro physical-index? int32 {entity-mib}?
      +--ro description?   string
      +--rw parent?        -> ../../component/name
      +--rw parent-rel-pos? int32
      +--ro contains-child* -> ../../component/name
      +--ro hardware-rev?   string
      +--ro firmware-rev?   string
      +--ro software-rev?   string
      +--ro serial-num?     string
      +--ro mfg-name?       string
      +--ro model-name?     string
      +--rw alias?         string
      +--rw asset-id?       string
      +--ro is-fru?         boolean
      +--ro mfg-date?       yang:date-and-time
      +--rw uri*           inet:uri
      +--ro uuid?          yang:uuid
      +--rw state {hardware-state}?
        | +--ro state-last-changed? yang:date-and-time
        | +--rw admin-state?         admin-state
        | +--ro oper-state?          oper-state
        | +--ro usage-state?         usage-state
        | +--ro alarm-state?         alarm-state
        | +--ro standby-state?       standby-state
      +--ro sensor-data {hardware-sensor}?
        +--ro value?                sensor-value
        +--ro value-type?            sensor-value-type
        +--ro value-scale?           sensor-value-scale
        +--ro value-precision?       sensor-value-precision
        +--ro oper-status?           sensor-status
        +--ro units-display?         string
        +--ro value-timestamp?       yang:date-and-time
        +--ro value-update-rate?     uint32

notifications:
  +---n hardware-state-change
  +---n hardware-state-oper-enabled {hardware-state}?
    | +--ro name?          -> /hardware/component/name
    | +--ro admin-state?   -> /hardware/component/state/admin-state
    | +--ro alarm-state?   -> /hardware/component/state/alarm-state
  +---n hardware-state-oper-disabled {hardware-state}?
    +--ro name?           -> /hardware/component/name
```



```
+-ro admin-state?    -> /hardware/component/state/admin-state
+-ro alarm-state?    -> /hardware/component/state/alarm-state
```

3.1. The Components Lists

The data model for hardware presented in this document uses a flat list of components. Each component in the list is identified by its name. Furthermore, each component has a mandatory "class" leaf.

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

The "class" leaf is a YANG identity that describes the type of the hardware. Vendors are encouraged to either directly use one of the common IANA-defined identities, or derive a more specific identity from one of them.

4. Relationship to ENTITY-MIB

If the device implements the ENTITY-MIB [[RFC6933](#)], each entry in the "/hardware/component" list in the operational state is mapped to one `EntPhysicalEntry`. Objects that are writable in the MIB are mapped to "config true" nodes in the "/hardware/component" list, except "entPhysicalSerialNum" which is writable in the MIB, but "config false" in the YANG module.

The "physical-index" leaf MUST contain the value of the corresponding `entPhysicalEntry`'s `entPhysicalIndex`.

The "class" leaf is mapped to both `entPhysicalClass` and `entPhysicalVendorType`. If the value of the "class" leaf is an identity that is either derived from or is one of the identities in the "iana-hardware" module, then `entPhysicalClass` contains the corresponding `IANAPhysicalClass` enumeration value. Otherwise, `entPhysicalClass` contains the `IANAPhysicalClass` value "other(1)". Vendors are encouraged to define an identity (derived from an identity in "iana-hardware" if possible) for each enterprise-specific registration identifier used for `entPhysicalVendorType`, and use that identity for the "class" leaf.

The following tables list the YANG data nodes with corresponding objects in the ENTITY-MIB.

YANG data node in	ENTITY-MIB object
/hardware/component	
name	entPhysicalName
class	entPhysicalClass
	entPhysicalVendorType
physical-index	entPhysicalIndex
description	entPhysicalDescr
parent	entPhysicalContainedIn
parent-rel-pos	entPhysicalParentRelPos
contains-child	entPhysicalChildIndex
hardware-rev	entPhysicalHardwareRev
firmware-rev	entPhysicalFirmwareRev
software-rev	entPhysicalSoftwareRev
serial-num	entPhysicalSerialNum
mfg-name	entPhysicalMfgName
model-name	entPhysicalModelName
alias	entPhysicalAlias
asset-id	entPhysicalAssetID
is-fru	entPhysicalIsFRU
mfg-date	entPhysicalMfgDate
uri	entPhysicalUris
uuid	entPhysicalUUID

YANG Data Nodes and Related ENTITY-MIB Objects

5. Relationship to ENTITY-SENSOR-MIB

If the device implements the ENTITY-SENSOR-MIB [[RFC3433](#)], each entry in the "/hardware/component" list where the container "sensor-data" exists is mapped to one EntPhySensorEntry.

YANG data node in	ENTITY-SENSOR-MIB object
/hardware/component/sensor-data	
value	entPhySensorValue
value-type	entPhySensorType
value-scale	entPhySensorScale
value-precision	entPhySensorPrecision
oper-status	entPhySensorOperStatus
units-display	entPhySensorUnitsDisplay
value-timestamp	entPhySensorValueTimeStamp
value-update-rate	entPhySensorValueUpdateRate

YANG Data Nodes and Related ENTITY-SENSOR-MIB Objects

6. Relationship to ENTITY-STATE-MIB

If the device implements the ENTITY-STATE-MIB [[RFC4268](#)], each entry in the "/hardware/component" list where the container "state" exists is mapped to one EntStateEntry.

YANG data node in	ENTITY-STATE-MIB object
/hardware/component/state	
state-last-changed	entStateLastChanged
admin-state	entStateAdmin
oper-state	entStateOper
usage-state	entStateUsage
alarm-state	entStateAlarm
standby-state	entStateStandby

YANG Data Nodes and Related ENTITY-STATE-MIB Objects

7. Hardware YANG Module

```
<CODE BEGINS> file "ietf-hardware@2018-01-15.yang"
```

```
module ietf-hardware {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-hardware";
  prefix hw;

  import ietf-inet-types {
    prefix inet;
  }
}
```



```
import ietf-yang-types {
  prefix yang;
}
import iana-hardware {
  prefix ianahw;
}

organization
  "IETF NETMOD (Network Modeling) Working Group";

contact
  "WG Web:  <http://tools.ietf.org/wg/netmod/>
  WG List:  <mailto:netmod@ietf.org>

  Editor:   Andy Bierman
            <mailto:andy@yumaworks.com>

  Editor:   Martin Bjorklund
            <mailto:mbj@tail-f.com>

  Editor:   Jie Dong
            <mailto:jie.dong@huawei.com>

  Editor:   Dan Romascanu
            <mailto:dromasca@gmail.com>";

// RFC Ed.: replace XXXX and YYYY with actual RFC numbers and
// remove this note.

description
  "This module contains a collection of YANG definitions for
  managing hardware.

  This data model is designed for the Network Management Datastore
  Architecture defined in RFC YYYY.

  Copyright (c) 2018 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 Simplified BSD License
  set forth in Section 4.c of the IETF Trust's Legal Provisions
  Relating to IETF Documents
  (http://trustee.ietf.org/license-info).

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



```
// RFC Ed.: update the date below with the date of RFC publication
// and remove this note.
revision 2018-01-15 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: A YANG Data Model for Hardware Management";
}

/*
 * Features
 */

feature entity-mib {
  description
    "This feature indicates that the device implements
    the ENTITY-MIB.";
  reference "RFC 6933: Entity MIB (Version 4)";
}

feature hardware-state {
  description
    "Indicates the ENTITY-STATE-MIB objects are supported";
  reference "RFC 4268: Entity State MIB";
}

feature hardware-sensor {
  description
    "Indicates the ENTITY-SENSOR-MIB objects are supported";
  reference "RFC 3433: Entity Sensor MIB";
}

/*
 * Typedefs
 */

typedef admin-state {
  type enumeration {
    enum unknown {
      value 1;
      description
        "The resource is unable to report administrative state.";
    }
    enum locked {
      value 2;
      description
        "The resource is administratively prohibited from use.";
    }
  }
}
```



```
enum shutting-down {
    value 3;
    description
        "The resource usage is administratively limited to current
        instances of use.";
}
enum unlocked {
    value 4;
    description
        "The resource is not administratively prohibited from
        use.";
}
}
description
    "Represents the various possible administrative states.";
reference "RFC 4268: EntityState";
}

typedef oper-state {
    type enumeration {
        enum unknown {
            value 1;
            description
                "The resource is unable to report its operational state.";
        }
        enum disabled {
            value 2;
            description
                "The resource is totally inoperable.";
        }
        enum enabled {
            value 3;
            description
                "The resource is partially or fully operable.";
        }
        enum testing {
            value 4;
            description
                "The resource is currently being tested and cannot
                therefore report whether it is operational or not.";
        }
    }
}
description
    "Represents the possible values of operational states.";
reference "RFC 4268: EntityOperState";
}

typedef usage-state {
```



```
type enumeration {
  enum unknown {
    value 1;
    description
      "The resource is unable to report usage state.";
  }
  enum idle {
    value 2;
    description
      "The resource is servicing no users.";
  }
  enum active {
    value 3;
    description
      "The resource is currently in use and it has sufficient
        spare capacity to provide for additional users.";
  }
  enum busy {
    value 4;
    description
      "The resource is currently in use, but it currently has no
        spare capacity to provide for additional users.";
  }
}
description
  "Represents the possible values of usage states.";
reference "RFC 4268, EntityUsageState";
}

typedef alarm-state {
  type bits {
    bit unknown {
      position 0;
      description
        "The resource is unable to report alarm state.";
    }
    bit under-repair {
      position 1;
      description
        "The resource is currently being repaired, which, depending
          on the implementation, may make the other values in this
          bit string not meaningful.";
    }
    bit critical {
      position 2;
      description
        "One or more critical alarms are active against the
          resource.";
    }
  }
}
```



```
    }
    bit major {
        position 3;
        description
            "One or more major alarms are active against the
             resource.";
    }
    bit minor {
        position 4;
        description
            "One or more minor alarms are active against the
             resource.";
    }
    bit warning {
        position 5;
        description
            "One or more warning alarms are active against the
             resource.";
    }
    bit indeterminate {
        position 6;
        description
            "One or more alarms of whose perceived severity cannot be
             determined are active against this resource.";
    }
}
description
    "Represents the possible values of alarm states. An alarm is a
     persistent indication of an error or warning condition.

     When no bits of this attribute are set, then no active alarms
     are known against this component and it is not under repair.";
reference "RFC 4268: EntityAlarmStatus";
}

typedef standby-state {
    type enumeration {
        enum unknown {
            value 1;
            description
                "The resource is unable to report standby state.";
        }
        enum hot-standby {
            value 2;
            description
                "The resource is not providing service, but it will be
                 immediately able to take over the role of the resource to
                 be backed up, without the need for initialization
```



```
        activity, and will contain the same information as the
        resource to be backed up.";
    }
    enum cold-standby {
        value 3;
        description
            "The resource is to back up another resource, but will not
            be immediately able to take over the role of a resource to
            be backed up, and will require some initialization
            activity.";
    }
    enum providing-service {
        value 4;
        description
            "The resource is providing service.";
    }
}
description
    "Represents the possible values of standby states.";
reference "RFC 4268: EntityStateStatus";
}

typedef sensor-value-type {
    type enumeration {
        enum other {
            value 1;
            description
                "A measure other than those listed below.";
        }
        enum unknown {
            value 2;
            description
                "An unknown measurement, or arbitrary, relative numbers";
        }
        enum volts-AC {
            value 3;
            description
                "A measure of electric potential (alternating current).";
        }
        enum volts-DC {
            value 4;
            description
                "A measure of electric potential (direct current).";
        }
        enum amperes {
            value 5;
            description
                "A measure of electric current.";
```



```
    }
    enum watts {
        value 6;
        description
            "A measure of power.";
    }
    enum hertz {
        value 7;
        description
            "A measure of frequency.";
    }
    enum celsius {
        value 8;
        description
            "A measure of temperature.";
    }
    enum percent-RH {
        value 9;
        description
            "A measure of percent relative humidity.";
    }
    enum rpm {
        value 10;
        description
            "A measure of shaft revolutions per minute.";
    }
    enum cmm {
        value 11;
        description
            "A measure of cubic meters per minute (airflow).";
    }
    enum truth-value {
        value 12;
        description
            "Value is one of 1 (true) or 2 (false)";
    }
}
description
    "A node using this data type represents the sensor measurement
    data type associated with a physical sensor value. The actual
    data units are determined by examining a node of this type
    together with the associated sensor-value-scale node.

    A node of this type SHOULD be defined together with nodes of
    type sensor-value-scale and sensor-value-precision. These
    three types are used to identify the semantics of a node of
    type sensor-value.";
reference "RFC 3433: EntitySensorDataType";
```



```
}

typedef sensor-value-scale {
  type enumeration {
    enum yocto {
      value 1;
      description
        "Data scaling factor of 10-24.";
    }
    enum zepto {
      value 2;
      description
        "Data scaling factor of 10-21.";
    }
    enum atto {
      value 3;
      description
        "Data scaling factor of 10-18.";
    }
    enum femto {
      value 4;
      description
        "Data scaling factor of 10-15.";
    }
    enum pico {
      value 5;
      description
        "Data scaling factor of 10-12.";
    }
    enum nano {
      value 6;
      description
        "Data scaling factor of 10-9.";
    }
    enum micro {
      value 7;
      description
        "Data scaling factor of 10-6.";
    }
    enum milli {
      value 8;
      description
        "Data scaling factor of 10-3.";
    }
    enum units {
      value 9;
      description
        "Data scaling factor of 100.";
    }
  }
}
```



```
}
enum kilo {
  value 10;
  description
    "Data scaling factor of 10^3.";
}
enum mega {
  value 11;
  description
    "Data scaling factor of 10^6.";
}
enum giga {
  value 12;
  description
    "Data scaling factor of 10^9.";
}
enum tera {
  value 13;
  description
    "Data scaling factor of 10^12.";
}
enum peta {
  value 14;
  description
    "Data scaling factor of 10^15.";
}
enum exa {
  value 15;
  description
    "Data scaling factor of 10^18.";
}
enum zetta {
  value 16;
  description
    "Data scaling factor of 10^21.";
}
enum yotta {
  value 17;
  description
    "Data scaling factor of 10^24.";
}
}
description
  "A node using this data type represents a data scaling factor,
  represented with an International System of Units (SI) prefix.
  The actual data units are determined by examining a node of
  this type together with the associated sensor-value-type."
```


A node of this type SHOULD be defined together with nodes of type sensor-value-type and sensor-value-precision. Together, associated nodes of these three types are used to identify the semantics of a node of type sensor-value.";

reference "[RFC 3433](#): EntitySensorDataScale";

}

typedef sensor-value-precision {
 type int8 {
 range "-8 .. 9";
 }
 description
 "A node using this data type represents a sensor value
 precision range.

A node of this type SHOULD be defined together with nodes of type sensor-value-type and sensor-value-scale. Together, associated nodes of these three types are used to identify the semantics of a node of type sensor-value.

If a node of this type contains a value in the range 1 to 9, it represents the number of decimal places in the fractional part of an associated sensor-value fixed- point number.

If a node of this type contains a value in the range -8 to -1, it represents the number of accurate digits in the associated sensor-value fixed-point number.

The value zero indicates the associated sensor-value node is not a fixed-point number.

Server implementers must choose a value for the associated sensor-value-precision node so that the precision and accuracy of the associated sensor-value node is correctly indicated.

For example, a component representing a temperature sensor that can measure 0 degrees to 100 degrees C in 0.1 degree increments, +/- 0.05 degrees, would have a sensor-value-precision value of '1', a sensor-value-scale value of 'units', and a sensor-value ranging from '0' to '1000'. The sensor-value would be interpreted as 'degrees C * 10'.";

reference "[RFC 3433](#): EntitySensorPrecision";

}

typedef sensor-value {
 type int32 {
 range "-10000000000 .. 10000000000";


```
}
description
  "A node using this data type represents a sensor value.

  A node of this type SHOULD be defined together with nodes of
  type sensor-value-type, sensor-value-scale, and
  sensor-value-precision. Together, associated nodes of those
  three types are used to identify the semantics of a node of
  this data type.

  The semantics of a node using this data type are determined by
  the value of the associated sensor-value-type node.

  If the associated sensor-value-type node is equal to 'voltsAC',
  'voltsDC', 'amperes', 'watts', 'hertz', 'celsius', or 'cmm',
  then a node of this type MUST contain a fixed point number
  ranging from -999,999,999 to +999,999,999. The value
  -10000000000 indicates an underflow error. The value +10000000000
  indicates an overflow error. The sensor-value-precision
  indicates how many fractional digits are represented in the
  associated sensor-value node.

  If the associated sensor-value-type node is equal to
  'percentRH', then a node of this type MUST contain a number
  ranging from 0 to 100.

  If the associated sensor-value-type node is equal to 'rpm',
  then a node of this type MUST contain a number ranging from
  -999,999,999 to +999,999,999.

  If the associated sensor-value-type node is equal to
  'truth-value', then a node of this type MUST contain either the
  value 1 (true) or the value 2 (false)'.

  If the associated sensor-value-type node is equal to 'other' or
  'unknown', then a node of this type MUST contain a number
  ranging from -10000000000 to 10000000000."
reference "RFC 3433: EntitySensorValue";
}

typedef sensor-status {
  type enumeration {
    enum ok {
      value 1;
      description
        "Indicates that the server can obtain the sensor value.";
    }
    enum unavailable {
```



```
        value 2;
        description
            "Indicates that the server presently cannot obtain the
             sensor value.";
    }
    enum nonoperational {
        value 3;
        description
            "Indicates that the server believes the sensor is broken.
             The sensor could have a hard failure (disconnected wire),
             or a soft failure such as out-of-range, jittery, or wildly
             fluctuating readings.";
    }
}
description
    "A node using this data type represents the operational status
     of a physical sensor.";
reference "RFC 3433: EntitySensorStatus";
}

/*
 * Data nodes
 */

container hardware {
    description
        "Data nodes representing components.

        If the server supports configuration of hardware components,
        then this data model is instantiated in the configuration
        datastores supported by the server. The leaf-list 'datastore'
        for the module 'ietf-hardware' in the YANG library provides
        this information.";

    leaf last-change {
        type yang:date-and-time;
        config false;
        description
            "The time the '/hardware/component' list changed in the
             operational state.";
    }

    list component {
        key name;
        description
            "List of components.

            When the server detects a new hardware component, it
```


initializes a list entry in the operational state.

If the server does not support configuration of hardware components, list entries in the operational state are initialized with values for all nodes as detected by the implementation.

Otherwise, the following procedure is followed:

1. If there is an entry in the /hardware/component list in the intended configuration with values for the nodes 'class', 'parent', 'parent-rel-pos' that are equal to the detected values, then the list entry in operational state is initialized with the configured values, including the 'name'.
2. Otherwise (i.e., there is no matching configuration entry), the list entry in the operational state is initialized with values for all nodes as detected by the implementation.

If the /hardware/component list in the intended configuration is modified, then the system MUST behave as if it re-initializes itself, and follow the procedure in (1).";
reference "[RFC 6933](#): entPhysicalEntry";

```
leaf name {  
  type string;  
  description  
    "The name assigned to this component.  
  
    This name is not required to be the same as  
    entPhysicalName.";  
}  
  
leaf class {  
  type identityref {  
    base ianahw:hardware-class;  
  }  
  mandatory true;  
  description  
    "An indication of the general hardware type of the  
    component.";  
  reference "RFC 6933: entPhysicalClass";  
}  
  
leaf physical-index {  
  if-feature entity-mib;
```



```
    type int32 {
      range "1..2147483647";
    }
    config false;
    description
      "The entPhysicalIndex for the entPhysicalEntry represented
      by this list entry.";
    reference "RFC 6933: entPhysicalIndex";
  }

  leaf description {
    type string;
    config false;
    description
      "A textual description of component. This node should
      contain a string that identifies the manufacturer's name
      for the component and should be set to a distinct value
      for each version or model of the component.";
    reference "RFC 6933: entPhysicalDescr";
  }

  leaf parent {
    type leafref {
      path "../../component/name";
      require-instance false;
    }
    description
      "The name of the component that physically contains this
      component.

      If this leaf is not instantiated, it indicates that this
      component is not contained in any other component.

      In the event that a physical component is contained by
      more than one physical component (e.g., double-wide
      modules), this node contains the name of one of these
      components. An implementation MUST use the same name
      every time this node is instantiated.";
    reference "RFC 6933: entPhysicalContainedIn";
  }

  leaf parent-rel-pos {
    type int32 {
      range "0 .. 2147483647";
    }
    description
      "An indication of the relative position of this child
      component among all its sibling components. Sibling
```


components are defined as components that:

- o Share the same value of the 'parent' node; and
- o Share a common base identity for the 'class' node.

Note that the last rule gives implementations flexibility in how components are numbered. For example, some implementations might have a single number series for all components derived from 'ianahw:port', while some others might have different number series for different components with identities derived from 'ianahw:port' (for example, one for RJ45 and one for SFP).";

```
reference "RFC 6933: entPhysicalParentRelPos";
}

leaf-list contains-child {
  type leafref {
    path "../../component/name";
  }
  config false;
  description
    "The name of the contained component.";
  reference "RFC 6933: entPhysicalChildIndex";
}

leaf hardware-rev {
  type string;
  config false;
  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: entPhysicalHardwareRev";
}

leaf firmware-rev {
  type string;
  config false;
  description
    "The vendor-specific firmware revision string for the
    component.";
  reference "RFC 6933: entPhysicalFirmwareRev";
}

leaf software-rev {
```



```
    type string;
    config false;
    description
        "The vendor-specific software revision string for the
        component.";
    reference "RFC 6933: entPhysicalSoftwareRev";
}

leaf serial-num {
    type string;
    config false;
    description
        "The vendor-specific serial number string for the
        component. The preferred value is the serial number
        string actually printed on the component itself (if
        present).";
    reference "RFC 6933: entPhysicalSerialNum";
}

leaf mfg-name {
    type string;
    config false;
    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,
        firmware-rev, software-rev, and the serial-num nodes are
        only meaningful amongst component 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: entPhysicalMfgName";
}

leaf model-name {
    type string;
    config false;
    description
        "The vendor-specific model name identifier string
        associated with this physical component. The preferred
        value is the customer-visible part number, which may be
        printed on the component itself.

        If the model name string associated with the physical
```



```
        component is unknown to the server, then this node is not
        instantiated.";
    reference "RFC 6933: entPhysicalModelName";
}

leaf alias {
    type string;
    description
        "An 'alias' name for the component, as specified by a
        network manager, and provides a non-volatile 'handle' for
        the component.

        If no configured value exists, the server MAY set the
        value of this node to a locally unique value in the
        operational state.

        A server implementation MAY map this leaf to the
        entPhysicalAlias MIB object. Such an implementation needs
        to use some mechanism to handle the differences in size
        and characters allowed between this leaf and
        entPhysicalAlias. The definition of such a mechanism is
        outside the scope of this document.";
    reference "RFC 6933: entPhysicalAlias";
}

leaf asset-id {
    type string;
    description
        "This node is a user-assigned asset tracking identifier for
        the component.

        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: entPhysicalAssetID";
}

leaf is-fru {
    type boolean;
    config false;
    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: entPhysicalIsFRU";
}

leaf mfg-date {
    type yang:date-and-time;
    config false;
    description
        "The date of manufacturing of the managed component.";
    reference "RFC 6933: entPhysicalMfgDate";
}

leaf-list uri {
    type inet:uri;
    description
        "This node contains identification information about the
        component.";
    reference "RFC 6933: entPhysicalUris";
}

leaf uuid {
    type yang:uuid;
    config false;
    description
        "A Universally Unique Identifier of the component.";
    reference "RFC 6933: entPhysicalUUID";
}

container state {
    if-feature hardware-state;
    description
        "State-related nodes";
    reference "RFC 4268: Entity State MIB";

    leaf state-last-changed {
        type yang:date-and-time;
        config false;
        description
            "The date and time when the value of any of the
            admin-state, oper-state, usage-state, alarm-state, or
            standby-state changed for this component.

            If there has been no change since the last
            re-initialization of the local system, this node
            contains the date and time of local system
            initialization.  If there has been no change since the
```



```
        component was added to the local system, this node
        contains the date and time of the insertion.";
    reference "RFC 4268: entStateLastChanged";
}

leaf admin-state {
    type admin-state;
    description
        "The administrative state for this component.

        This node refers to a component's administrative
        permission to service both other components within its
        containment hierarchy as well other users of its
        services defined by means outside the scope of this
        module.

        Some components exhibit only a subset of the remaining
        administrative state values. Some components cannot be
        locked, and hence this node exhibits only the 'unlocked'
        state. Other components cannot be shutdown gracefully,
        and hence this node does not exhibit the 'shutting-down'
        state.";
    reference "RFC 4268: entStateAdmin";
}

leaf oper-state {
    type oper-state;
    config false;
    description
        "The operational state for this component.

        Note that this node does not follow the administrative
        state. An administrative state of down does not predict
        an operational state of disabled.

        Note that some implementations may not be able to
        accurately report oper-state while the admin-state node
        has a value other than 'unlocked'. In these cases, this
        node MUST have a value of 'unknown'.";
    reference "RFC 4268: entStateOper";
}

leaf usage-state {
    type usage-state;
    config false;
    description
        "The usage state for this component.
```


This node refers to a component's ability to service more components in a containment hierarchy.

Some components will exhibit only a subset of the usage state values. Components that are unable to ever service any components within a containment hierarchy will always have a usage state of 'busy'. Some components will only ever be able to support one component within its containment hierarchy and will therefore only exhibit values of 'idle' and 'busy'.";

reference "[RFC 4268](#), entStateUsage";

}

leaf alarm-state {

type alarm-state;

config false;

description

"The alarm state for this component. It does not include the alarms raised on child components within its containment hierarchy.";

reference "[RFC 4268](#): entStateAlarm";

}

leaf standby-state {

type standby-state;

config false;

description

"The standby state for this component.

Some components will exhibit only a subset of the remaining standby state values. If this component cannot operate in a standby role, the value of this node will always be 'providing-service'.";

reference "[RFC 4268](#): entStateStandby";

}

}

container sensor-data {

when 'derived-from-or-self(..../class,
"ianahw:sensor")' {

description

"Sensor data nodes present for any component of type 'sensor'";

}

if-feature hardware-sensor;

config false;

description


```
"Sensor-related nodes.";
reference "RFC 3433: Entity Sensor MIB";

leaf value {
  type sensor-value;
  description
    "The most recent measurement obtained by the server
    for this sensor.

    A client that periodically fetches this node should also
    fetch the nodes 'value-type', 'value-scale', and
    'value-precision', since they may change when the value
    is changed.";
  reference "RFC 3433: entPhySensorValue";
}

leaf value-type {
  type sensor-value-type;
  description
    "The type of data units associated with the
    sensor value";
  reference "RFC 3433: entPhySensorType";
}

leaf value-scale {
  type sensor-value-scale;
  description
    "The (power of 10) scaling factor associated
    with the sensor value";
  reference "RFC 3433: entPhySensorScale";
}

leaf value-precision {
  type sensor-value-precision;
  description
    "The number of decimal places of precision
    associated with the sensor value";
  reference "RFC 3433: entPhySensorPrecision";
}

leaf oper-status {
  type sensor-status;
  description
    "The operational status of the sensor.";
  reference "RFC 3433: entPhySensorOperStatus";
}

leaf units-display {
```



```
    type string;
    description
      "A textual description of the data units that should be
       used in the display of the sensor value.";
    reference "RFC 3433: entPhySensorUnitsDisplay";
  }

  leaf value-timestamp {
    type yang:date-and-time;
    description
      "The time the status and/or value of this sensor was last
       obtained by the server.";
    reference "RFC 3433: entPhySensorValueTimeStamp";
  }

  leaf value-update-rate {
    type uint32;
    units "milliseconds";
    description
      "An indication of the frequency that the server updates
       the associated 'value' node, representing in
       milliseconds. The value zero indicates:

        - the sensor value is updated on demand (e.g.,
          when polled by the server for a get-request),
        - the sensor value is updated when the sensor
          value changes (event-driven),
        - the server does not know the update rate.";
    reference "RFC 3433: entPhySensorValueUpdateRate";
  }
}
}
}

/*
 * Notifications
 */

notification hardware-state-change {
  description
    "A hardware-state-change notification is generated when the
     value of /hardware/last-change changes in the operational
     state.";
  reference "RFC 6933, entConfigChange";
}

notification hardware-state-oper-enabled {
  if-feature hardware-state;
```



```
description
  "A hardware-state-oper-enabled notification signifies that a
  component has transitioned into the 'enabled' state.";

leaf name {
  type leafref {
    path "/hardware/component/name";
  }
  description
    "The name of the component that has transitioned into the
    'enabled' state.";
}
leaf admin-state {
  type leafref {
    path "/hardware/component/state/admin-state";
  }
  description
    "The administrative state for the component.";
}
leaf alarm-state {
  type leafref {
    path "/hardware/component/state/alarm-state";
  }
  description
    "The alarm state for the component.";
}
reference "RFC 4268, entStateOperEnabled";
}

notification hardware-state-oper-disabled {
  if-feature hardware-state;
  description
    "A hardware-state-oper-disabled notification signifies that a
    component has transitioned into the 'disabled' state.";

  leaf name {
    type leafref {
      path "/hardware/component/name";
    }
    description
      "The name of the component that has transitioned into the
      'disabled' state.";
  }
  leaf admin-state {
    type leafref {
      path "/hardware/component/state/admin-state";
    }
  }
  description
```



```
        "The administrative state for the component.";
    }
    leaf alarm-state {
        type leafref {
            path "/hardware/component/state/alarm-state";
        }
        description
            "The alarm state for the component.";
    }
    reference "RFC 4268, entStateOperDisabled";
}

}

<CODE ENDS>

<CODE BEGINS> file "iana-hardware@2018-01-15.yang"

module iana-hardware {
    yang-version 1.1;
    namespace "urn:ietf:params:xml:ns:yang:iana-hardware";
    prefix ianahw;

    organization "IANA";
    contact
        "
            Internet Assigned Numbers Authority

            Postal: ICANN
                12025 Waterfront Drive, Suite 300
                Los Angeles, CA 90094-2536
                United States

            Tel: +1 310 301 5800
            E-Mail: iana@iana.org";
    // RFC Ed.: replace XXXX with actual RFC number and remove this
    // note.
    description
        "IANA defined identities for hardware class.

        The latest revision of this YANG module can be obtained from
        the IANA web site.

        Requests for new values should be made to IANA via
        email (iana@iana.org).

        Copyright (c) 2018 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 Simplified BSD License set forth in [Section 4.c](#) of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

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

reference

// RFC Ed.: replace PPPP with actual path and remove this note.
"https://www.iana.org/assignments/PPPP";

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

```
revision 2018-01-15 {  
  description  
    "Initial revision.";  
  reference  
    "RFC XXXX: A YANG Data Model for Hardware Management";  
}
```

```
/*  
 * Identities  
 */
```

```
identity hardware-class {  
  description  
    "This identity is the base for all hardware class  
    identifiers.";  
}
```

```
identity unknown {  
  base ianahw:hardware-class;  
  description  
    "This identity is applicable if the hardware class is unknown  
    to the server.";  
}
```

```
identity chassis {  
  base ianahw:hardware-class;  
  description  
    "This identity is applicable if the hardware class is an  
    overall container for networking equipment. Any class of  
    physical component, except a stack, may be contained within a  
    chassis; a chassis may only be contained within a stack.";  
}
```



```
identity backplane {
  base ianahw:hardware-class;
  description
    "This identity is applicable if the hardware class is some sort
    of device for aggregating and forwarding networking traffic,
    such as a shared backplane in a modular ethernet switch. Note
    that an implementation may model a backplane as a single
    physical component, which is actually implemented as multiple
    discrete physical components (within a chassis or stack).";
}

identity container {
  base ianahw:hardware-class;
  description
    "This identity is applicable if the hardware class is capable
    of containing one or more removable physical entities,
    possibly of different types. For example, each (empty or
    full) slot in a chassis will be modeled as a container. Note
    that all removable physical components should be modeled
    within a container component, such as field-replaceable
    modules, fans, or power supplies. Note that all known
    containers should be modeled by the agent, including empty
    containers.";
}

identity power-supply {
  base ianahw:hardware-class;
  description
    "This identity is applicable if the hardware class is a
    power-supplying component.";
}

identity fan {
  base ianahw:hardware-class;
  description
    "This identity is applicable if the hardware class is a fan or
    other heat-reduction component.";
}

identity sensor {
  base ianahw:hardware-class;
  description
    "This identity is applicable if the hardware class is some sort
    of sensor, such as a temperature sensor within a router
    chassis.";
}

identity module {
```



```
base ianahw:hardware-class;
description
  "This identity is applicable if the hardware class is some sort
  of self-contained sub-system.  If a module component is
  removable, then it should be modeled within a container
  component; otherwise, it should be modeled directly within
  another physical component (e.g., a chassis or another
  module).";
}

identity port {
  base ianahw:hardware-class;
  description
    "This identity is applicable if the hardware class is some sort
    of networking port, capable of receiving and/or transmitting
    networking traffic.";
}

identity stack {
  base ianahw:hardware-class;
  description
    "This identity is applicable if the hardware class is some sort
    of super-container (possibly virtual) intended to group
    together multiple chassis entities.  A stack may be realized
    by a virtual cable, a real interconnect cable attached to
    multiple chassis, or multiple interconnect cables.  A stack
    should not be modeled within any other physical components,
    but a stack may be contained within another stack.  Only
    chassis components should be contained within a stack.";
}

identity cpu {
  base ianahw:hardware-class;
  description
    "This identity is applicable if the hardware class is some sort
    of central processing unit.";
}

identity energy-object {
  base ianahw:hardware-class;
  description
    "This identity is applicable if the hardware class is some sort
    of energy object, i.e., a piece of equipment that is part of
    or attached to a communications network that is monitored,
    controlled, or aids in the management of another device for
    Energy Management.";
}
```



```
identity battery {
  base ianahw:hardware-class;
  description
    "This identity is applicable if the hardware class is some sort
    of battery.";
}

identity storage-drive {
  base ianahw:hardware-class;
  description
    "This identity is applicable if the hardware class is some sort
    of component with data storage capability as main
    functionality, e.g., disk drive (HDD), solid state device
    (SSD), hybrid (SSHD), object storage (OSD) or other.";
}
}

<CODE ENDS>
```

8. IANA Considerations

This document defines the initial version of the IANA-maintained "iana-hardware" YANG module.

The "iana-hardware" YANG module is intended to reflect the "IANA-ENTITY-MIB" MIB module so that if a new enumeration is added to the "IANAPhysicalClass" TEXTUAL-CONVENTION, the same class is added as an identity derived from "ianahw:hardware-class".

When the "iana-hardware" YANG module is updated, a new "revision" statement must be added in front of the existing revision statements.

8.1. URI Registrations

This document registers three URIs in the IETF XML registry [[RFC3688](#)]. Following the format in [RFC 3688](#), the following registrations are requested to be made.

URI: urn:ietf:params:xml:ns:yang:iana-hardware
Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-hardware
Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.

URI: urn:ietf:params:xml:ns:yang:ietf-hardware-state
Registrant Contact: The IESG.
XML: N/A, the requested URI is an XML namespace.

8.2. YANG Module Registrations

This document registers three YANG modules in the YANG Module Names registry [[RFC6020](#)].

name: iana-hardware
namespace: urn:ietf:params:xml:ns:yang:iana-hardware
prefix: ianahw
reference: RFC XXXX

name: ietf-hardware
namespace: urn:ietf:params:xml:ns:yang:ietf-hardware
prefix: hw
reference: RFC XXXX

name: ietf-hardware-state
namespace: urn:ietf:params:xml:ns:yang:ietf-hardware-state
prefix: hw-state
reference: RFC XXXX

9. Security Considerations

The YANG modules specified in this document define a schema for data that is designed to be accessed via network management protocols such as NETCONF [[RFC6241](#)] or RESTCONF [[RFC8040](#)]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) [[RFC6242](#)]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [[RFC5246](#)].

The NETCONF access control model [[RFC6536](#)] 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.

There are a number of data nodes defined in the YANG module "ietf-hardware" that are writable/creatable/deletable (i.e., config true, which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

/hardware/component/admin-state: Setting this node to 'locked' or 'shutting-down' can cause disruption of services ranging from those running on a port to those on an entire device, depending on the type of component.

Some of the readable data nodes in these YANG modules 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. These are the subtrees and data nodes and their sensitivity/vulnerability:

/hardware/component: The leafs in this list expose information about the physical components in a device, which may be used to identify the vendor, model, version, and specific device-identification information of each system component.

/hardware/component/sensor-data/value: This node may expose the values of particular physical sensors in a device.

/hardware/component/state: Access to this node allows one to figure out what the active and standby resources in a device are.

10. Acknowledgments

The authors wish to thank the following individuals, who all provided helpful comments on various draft versions of this document: Bart Bogaert, Timothy Carey, William Lupton, Juergen Schoenwaelder.

11. References

11.1. Normative References

[I-D.ietf-netmod-revised-datastores]
Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K.,
and R. Wilton, "Network Management Datastore
Architecture", [draft-ietf-netmod-revised-datastores-10](#)
(work in progress), January 2018.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3433] Bierman, A., Romascanu, D., and K. Norseth, "Entity Sensor Management Information Base", [RFC 3433](#), DOI 10.17487/RFC3433, December 2002, <<https://www.rfc-editor.org/info/rfc3433>>.
- [RFC3688] Mealling, M., "The IETF XML Registry", [BCP 81](#), [RFC 3688](#), DOI 10.17487/RFC3688, January 2004, <<https://www.rfc-editor.org/info/rfc3688>>.
- [RFC4268] Chisholm, S. and D. Perkins, "Entity State MIB", [RFC 4268](#), DOI 10.17487/RFC4268, November 2005, <<https://www.rfc-editor.org/info/rfc4268>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", [RFC 5246](#), DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [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/info/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/info/rfc6241>>.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", [RFC 6242](#), DOI 10.17487/RFC6242, June 2011, <<https://www.rfc-editor.org/info/rfc6242>>.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", [RFC 6536](#), DOI 10.17487/RFC6536, March 2012, <<https://www.rfc-editor.org/info/rfc6536>>.
- [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/info/rfc6933>>.

- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", [RFC 7950](#), DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", [RFC 8040](#), DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in [RFC 2119](#) Key Words", [BCP 14](#), [RFC 8174](#), DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

11.2. Informative References

- [I-D.ietf-netmod-yang-tree-diagrams]
Bjorklund, M. and L. Berger, "YANG Tree Diagrams", [draft-ietf-netmod-yang-tree-diagrams-04](#) (work in progress), December 2017.

Appendix A. Hardware State Data Model

This non-normative appendix contains a data model designed as a temporary solution for implementations that do not yet support the Network Management Datastore Architecture (NMDA) defined in [\[I-D.ietf-netmod-revised-datastores\]](#). It has the following structure:

```
module: ietf-hardware-state
  x--ro hardware
    x--ro last-change?    yang:date-and-time
    x--ro component* [name]
      x--ro name          string
      x--ro class          identityref
      x--ro physical-index? int32 {entity-mib}?
      x--ro description?  string
      x--ro parent?       -> ../../component/name
      x--ro parent-rel-pos? int32
      x--ro contains-child* -> ../../component/name
      x--ro hardware-rev?  string
      x--ro firmware-rev?  string
      x--ro software-rev?  string
      x--ro serial-num?    string
      x--ro mfg-name?      string
      x--ro model-name?    string
      x--ro alias?         string
      x--ro asset-id?      string
      x--ro is-fru?         boolean
      x--ro mfg-date?      yang:date-and-time
```



```

x--ro uri*                inet:uri
x--ro uuid?                yang:uuid
x--ro state {hardware-state}?
| x--ro state-last-changed? yang:date-and-time
| x--ro admin-state?        hw:admin-state
| x--ro oper-state?         hw:oper-state
| x--ro usage-state?        hw:usage-state
| x--ro alarm-state?        hw:alarm-state
| x--ro standby-state?      hw:standby-state
x--ro sensor-data {hardware-sensor}?
  x--ro value?              hw:sensor-value
  x--ro value-type?         hw:sensor-value-type
  x--ro value-scale?        hw:sensor-value-scale
  x--ro value-precision?    hw:sensor-value-precision
  x--ro oper-status?        hw:sensor-status
  x--ro units-display?      string
  x--ro value-timestamp?    yang:date-and-time
  x--ro value-update-rate?  uint32

```

notifications:

```

x---n hardware-state-change
x---n hardware-state-oper-enabled {hardware-state}?
| x--ro name?              -> /hardware/component/name
| x--ro admin-state?       -> /hardware/component/state/admin-state
| x--ro alarm-state?       -> /hardware/component/state/alarm-state
x---n hardware-state-oper-disabled {hardware-state}?
  x--ro name?              -> /hardware/component/name
  x--ro admin-state?       -> /hardware/component/state/admin-state
  x--ro alarm-state?       -> /hardware/component/state/alarm-state

```

[A.1.](#) Hardware State YANG Module

<CODE BEGINS> file "ietf-hardware-state@2017-12-18.yang"

```

module ietf-hardware-state {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-hardware-state";
  prefix hw-state;

  import ietf-inet-types {
    prefix inet;
  }
  import ietf-yang-types {
    prefix yang;
  }
  import iana-hardware {
    prefix ianahw;
  }
}

```



```
import ietf-hardware {
  prefix hw;
}

organization
  "IETF NETMOD (Network Modeling) Working Group";

contact
  "WG Web:  <http://tools.ietf.org/wg/netmod/>
  WG List:  <mailto:netmod@ietf.org>

  Editor:   Andy Bierman
            <mailto:andy@yumaworks.com>

  Editor:   Martin Bjorklund
            <mailto:mbj@tail-f.com>

  Editor:   Jie Dong
            <mailto:jie.dong@huawei.com>

  Editor:   Dan Romascanu
            <mailto:dromasca@gmail.com>";

// RFC Ed.: replace XXXX and YYYY with actual RFC numbers and
// remove this note.

description
  "This module contains a collection of YANG definitions for
  monitoring hardware.

  This data model is designed as a temporary solution for
  implementations that do not yet support the Network Management
  Datastore Architecture (NMDA) defined in RFC YYYY.  Such an
  implementation cannot implement the module 'ietf-hardware'
  properly, since without NMDA support, it is not possible to
  distinguish between instances of nodes in the running
  configuration and operational state.

  The data model in this module is the same as the data model in
  'ietf-hardware', except all nodes are marked as 'config false'.

  If a server that implements this module but doesn't support NMDA
  also supports configuration of hardware components, it SHOULD
  also implement the module 'ietf-hardware' in the configuration
  datastores.  The corresponding state data is found in the
  '/hw-state:hardware' subtree.

  Copyright (c) 2017 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 Simplified BSD License set forth in [Section 4.c](#) of the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>).

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

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

```
revision 2017-12-18 {
  description
    "Initial revision.";
  reference
    "RFC XXXX: A YANG Data Model for Hardware Management";
}
```

```
/*
 * Features
 */
```

```
feature entity-mib {
  status deprecated;
  description
    "This feature indicates that the device implements
    the ENTITY-MIB.";
  reference "RFC 6933: Entity MIB (Version 4)";
}
```

```
feature hardware-state {
  status deprecated;
  description
    "Indicates the ENTITY-STATE-MIB objects are supported";
  reference "RFC 4268: Entity State MIB";
}
```

```
feature hardware-sensor {
  status deprecated;
  description
    "Indicates the ENTITY-SENSOR-MIB objects are supported";
  reference "RFC 3433: Entity Sensor MIB";
}
```

```
/*
```



```
* Data nodes
*/
```

```
container hardware {
  config false;
  status deprecated;
  description
    "Data nodes representing components.";

  leaf last-change {
    type yang:date-and-time;
    status deprecated;
    description
      "The time the '/hardware/component' list changed in the
        operational state.";
  }

  list component {
    key name;
    status deprecated;
    description
      "List of components.
```

When the server detects a new hardware component, it initializes a list entry in the operational state.

If the server does not support configuration of hardware components, list entries in the operational state are initialized with values for all nodes as detected by the implementation.

Otherwise, the following procedure is followed:

1. If there is an entry in the /hardware/component list in the intended configuration with values for the nodes 'class', 'parent', 'parent-rel-pos' that are equal to the detected values, then:
 - 1a. If the configured entry has a value for 'mfg-name' that is equal to the detected value, or if the 'mfg-name' value cannot be detected, then the list entry in the operational state is initialized with the configured values for all configured nodes, including the 'name'.

Otherwise, the list entry in the operational state is initialized with values for all nodes as detected by the implementation. The implementation may raise an

alarm that informs about the 'mfg-name' mismatch condition. How this is done is outside the scope of this document.

- 1b. Otherwise (i.e., there is no matching configuration entry), the list entry in the operational state is initialized with values for all nodes as detected by the implementation.

If the /hardware/component list in the intended configuration is modified, then the system MUST behave as if it re-initializes itself, and follow the procedure in (1).";
reference "[RFC 6933](#): entPhysicalEntry";

```
leaf name {  
  type string;  
  status deprecated;  
  description  
    "The name assigned to this component.
```

```
    This name is not required to be the same as  
    entPhysicalName.";
```

```
}
```

```
leaf class {  
  type identityref {  
    base ianahw:hardware-class;  
  }  
  mandatory true;  
  status deprecated;  
  description  
    "An indication of the general hardware type of the  
    component.";  
  reference "RFC 6933: entPhysicalClass";  
}
```

```
leaf physical-index {  
  if-feature entity-mib;  
  type int32 {  
    range "1..2147483647";  
  }  
  status deprecated;  
  description  
    "The entPhysicalIndex for the entPhysicalEntry represented  
    by this list entry.";  
  reference "RFC 6933: entPhysicalIndex";  
}
```



```
leaf description {  
  type string;  
  status deprecated;  
  description  
    "A textual description of component. This node should  
    contain a string that identifies the manufacturer's name  
    for the component and should be set to a distinct value  
    for each version or model of the component.";  
  reference "RFC 6933: entPhysicalDescr";  
}
```

```
leaf parent {  
  type leafref {  
    path "../../component/name";  
    require-instance false;  
  }  
  status deprecated;  
  description  
    "The name of the component that physically contains this  
    component.  
  
    If this leaf is not instantiated, it indicates that this  
    component is not contained in any other component.  
  
    In the event that a physical component is contained by  
    more than one physical component (e.g., double-wide  
    modules), this node contains the name of one of these  
    components. An implementation MUST use the same name  
    every time this node is instantiated.";  
  reference "RFC 6933: entPhysicalContainedIn";  
}
```

```
leaf parent-rel-pos {  
  type int32 {  
    range "0 .. 2147483647";  
  }  
  status deprecated;  
  description  
    "An indication of the relative position of this child  
    component among all its sibling components. Sibling  
    components are defined as components that:  
  
    o Share the same value of the 'parent' node; and  
  
    o Share a common base identity for the 'class' node.  
  
    Note that the last rule gives implementations flexibility  
    in how components are numbered. For example, some
```


implementations might have a single number series for all components derived from 'ianahw:port', while some others might have different number series for different components with identities derived from 'ianahw:port' (for example, one for RJ45 and one for SFP).";

```
reference "RFC 6933: entPhysicalParentRelPos";
}
```

```
leaf-list contains-child {
  type leafref {
    path "../../component/name";
  }
  status deprecated;
  description
    "The name of the contained component.";
  reference "RFC 6933: entPhysicalChildIndex";
}
```

```
leaf hardware-rev {
  type string;
  status deprecated;
  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: entPhysicalHardwareRev";
}
```

```
leaf firmware-rev {
  type string;
  status deprecated;
  description
    "The vendor-specific firmware revision string for the
    component.";
  reference "RFC 6933: entPhysicalFirmwareRev";
}
```

```
leaf software-rev {
  type string;
  status deprecated;
  description
    "The vendor-specific software revision string for the
    component.";
  reference "RFC 6933: entPhysicalSoftwareRev";
}
```



```
leaf serial-num {
  type string;
  status deprecated;
  description
    "The vendor-specific serial number string for the
    component. The preferred value is the serial number
    string actually printed on the component itself (if
    present).";
  reference "RFC 6933: entPhysicalSerialNum";
}

leaf mfg-name {
  type string;
  status deprecated;
  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,
    firmware-rev, software-rev, and the serial-num nodes are
    only meaningful amongst component 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: entPhysicalMfgName";
}

leaf model-name {
  type string;
  status deprecated;
  description
    "The vendor-specific model name identifier string
    associated with this physical component. The preferred
    value is the customer-visible part number, which may be
    printed on the component itself.

    If the model name string associated with the physical
    component is unknown to the server, then this node is not
    instantiated.";
  reference "RFC 6933: entPhysicalModelName";
}

leaf alias {
  type string;
  status deprecated;
```



```
description
  "An 'alias' name for the component, as specified by a
  network manager, and provides a non-volatile 'handle' for
  the component.

  If no configured value exists, the server MAY set the
  value of this node to a locally unique value in the
  operational state.

  A server implementation MAY map this leaf to the
  entPhysicalAlias MIB object. Such an implementation needs
  to use some mechanism to handle the differences in size
  and characters allowed between this leaf and
  entPhysicalAlias. The definition of such a mechanism is
  outside the scope of this document.";
reference "RFC 6933: entPhysicalAlias";
}

leaf asset-id {
  type string;
  status deprecated;
  description
    "This node is a user-assigned asset tracking identifier for
    the component.

    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: entPhysicalAssetID";
}

leaf is-fru {
  type boolean;
  status deprecated;
  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: entPhysicalIsFRU";
}
```



```
leaf mfg-date {
  type yang:date-and-time;
  status deprecated;
  description
    "The date of manufacturing of the managed component.";
  reference "RFC 6933: entPhysicalMfgDate";
}

leaf-list uri {
  type inet:uri;
  status deprecated;
  description
    "This node contains identification information about the
    component.";
  reference "RFC 6933: entPhysicalUris";
}

leaf uuid {
  type yang:uuid;
  status deprecated;
  description
    "A Universally Unique Identifier of the component.";
  reference "RFC 6933: entPhysicalUUID";
}

container state {
  if-feature hardware-state;
  status deprecated;
  description
    "State-related nodes";
  reference "RFC 4268: Entity State MIB";

  leaf state-last-changed {
    type yang:date-and-time;
    status deprecated;
    description
      "The date and time when the value of any of the
      admin-state, oper-state, usage-state, alarm-state, or
      standby-state changed for this component.

      If there has been no change since the last
      re-initialization of the local system, this node
      contains the date and time of local system
      initialization.  If there has been no change since the
      component was added to the local system, this node
      contains the date and time of the insertion.";
    reference "RFC 4268: entStateLastChanged";
  }
}
```



```
leaf admin-state {
  type hw:admin-state;
  status deprecated;
  description
    "The administrative state for this component.

    This node refers to a component's administrative
    permission to service both other components within its
    containment hierarchy as well other users of its
    services defined by means outside the scope of this
    module.

    Some components exhibit only a subset of the remaining
    administrative state values. Some components cannot be
    locked, and hence this node exhibits only the 'unlocked'
    state. Other components cannot be shutdown gracefully,
    and hence this node does not exhibit the 'shutting-down'
    state.";
  reference "RFC 4268: entStateAdmin";
}

leaf oper-state {
  type hw:oper-state;
  status deprecated;
  description
    "The operational state for this component.

    Note that this node does not follow the administrative
    state. An administrative state of down does not predict
    an operational state of disabled.

    Note that some implementations may not be able to
    accurately report oper-state while the admin-state node
    has a value other than 'unlocked'. In these cases, this
    node MUST have a value of 'unknown'.";
  reference "RFC 4268: entStateOper";
}

leaf usage-state {
  type hw:usage-state;
  status deprecated;
  description
    "The usage state for this component.

    This node refers to a component's ability to service
    more components in a containment hierarchy.

    Some components will exhibit only a subset of the usage
```



```
state values. Components that are unable to ever
service any components within a containment hierarchy
will always have a usage state of 'busy'. Some
components will only ever be able to support one
component within its containment hierarchy and will
therefore only exhibit values of 'idle' and 'busy'.
reference "RFC 4268, entStateUsage";
}

leaf alarm-state {
  type hw:alarm-state;
  status deprecated;
  description
    "The alarm state for this component. It does not
    include the alarms raised on child components within its
    containment hierarchy.";
  reference "RFC 4268: entStateAlarm";
}

leaf standby-state {
  type hw:standby-state;
  status deprecated;
  description
    "The standby state for this component.

    Some components will exhibit only a subset of the
    remaining standby state values. If this component
    cannot operate in a standby role, the value of this node
    will always be 'providing-service'.";
  reference "RFC 4268: entStateStandby";
}

container sensor-data {
  when 'derived-from-or-self(..../class,
                                "ianahw:sensor")' {
    description
      "Sensor data nodes present for any component of type
      'sensor'";
  }
  if-feature hardware-sensor;
  status deprecated;

  description
    "Sensor-related nodes.";
  reference "RFC 3433: Entity Sensor MIB";

  leaf value {
```



```
    type hw:sensor-value;
    status deprecated;
    description
        "The most recent measurement obtained by the server
        for this sensor.

        A client that periodically fetches this node should also
        fetch the nodes 'value-type', 'value-scale', and
        'value-precision', since they may change when the value
        is changed.";
    reference "RFC 3433: entPhySensorValue";
}

leaf value-type {
    type hw:sensor-value-type;
    status deprecated;
    description
        "The type of data units associated with the
        sensor value";
    reference "RFC 3433: entPhySensorType";
}

leaf value-scale {
    type hw:sensor-value-scale;
    status deprecated;
    description
        "The (power of 10) scaling factor associated
        with the sensor value";
    reference "RFC 3433: entPhySensorScale";
}

leaf value-precision {
    type hw:sensor-value-precision;
    status deprecated;
    description
        "The number of decimal places of precision
        associated with the sensor value";
    reference "RFC 3433: entPhySensorPrecision";
}

leaf oper-status {
    type hw:sensor-status;
    status deprecated;
    description
        "The operational status of the sensor.";
    reference "RFC 3433: entPhySensorOperStatus";
}
```



```
leaf units-display {
  type string;
  status deprecated;
  description
    "A textual description of the data units that should be
     used in the display of the sensor value.";
  reference "RFC 3433: entPhySensorUnitsDisplay";
}

leaf value-timestamp {
  type yang:date-and-time;
  status deprecated;
  description
    "The time the status and/or value of this sensor was last
     obtained by the server.";
  reference "RFC 3433: entPhySensorValueTimeStamp";
}

leaf value-update-rate {
  type uint32;
  units "milliseconds";
  status deprecated;
  description
    "An indication of the frequency that the server updates
     the associated 'value' node, representing in
     milliseconds. The value zero indicates:

     - the sensor value is updated on demand (e.g.,
       when polled by the server for a get-request),
     - the sensor value is updated when the sensor
       value changes (event-driven),
     - the server does not know the update rate.";
  reference "RFC 3433: entPhySensorValueUpdateRate";
}
}
}

/*
 * Notifications
 */

notification hardware-state-change {
  status deprecated;
  description
    "A hardware-state-change notification is generated when the
     value of /hardware/last-change changes in the operational
     state.";
```



```
    reference "RFC 6933, entConfigChange";
}

notification hardware-state-oper-enabled {
    if-feature hardware-state;
    status deprecated;
    description
        "A hardware-state-oper-enabled notification signifies that a
        component has transitioned into the 'enabled' state.";

    leaf name {
        type leafref {
            path "/hardware/component/name";
        }
        status deprecated;
        description
            "The name of the component that has transitioned into the
            'enabled' state.";
    }
    leaf admin-state {
        type leafref {
            path "/hardware/component/state/admin-state";
        }
        status deprecated;
        description
            "The administrative state for the component.";
    }
    leaf alarm-state {
        type leafref {
            path "/hardware/component/state/alarm-state";
        }
        status deprecated;
        description
            "The alarm state for the component.";
    }
    reference "RFC 4268, entStateOperEnabled";
}

notification hardware-state-oper-disabled {
    if-feature hardware-state;
    status deprecated;
    description
        "A hardware-state-oper-disabled notification signifies that a
        component has transitioned into the 'disabled' state.";

    leaf name {
        type leafref {
            path "/hardware/component/name";
```



```
    }
    status deprecated;
    description
      "The name of the component that has transitioned into the
        'disabled' state.";
  }
  leaf admin-state {
    type leafref {
      path "/hardware/component/state/admin-state";
    }
    status deprecated;
    description
      "The administrative state for the component.";
  }
  leaf alarm-state {
    type leafref {
      path "/hardware/component/state/alarm-state";
    }
    status deprecated;
    description
      "The alarm state for the component.";
  }
  reference "RFC 4268, entStateOperDisabled";
}

}
```

<CODE ENDS>

Authors' Addresses

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Jie Dong
Huawei Technologies

Email: jie.dong@huawei.com

Dan Romascanu

Email: dromasca@gmail.com