Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 10, 2013

M. Bjorklund
Tail-f Systems

February 6, 2013

A YANG Data Model for Interface Management
draft-ietf-netmod-interfaces-cfg-09

Abstract

This document defines a YANG data model for the management of network interfaces. It is expected that interface type specific data models augment the generic interfaces data model defined in this document.

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 August 10, 2013.

Table of Contents

1. Introduction

This document defines a YANG [RFC6020] data model for the management of network interfaces. It is expected that interface type specific data models augment the generic interfaces data model defined in this document.

Network interfaces are central to the management of many Internet protocols. Thus, it is important to establish a common data model for how interfaces are identified and configured.

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].

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

- client
- server

The following terms are defined in [RFC6020] and are not redefined here:

- augment
- data model
- data node

2. Objectives

This section describes some of the design objectives for the model presented in Section 5.

- It is recognized that existing implementations will have to map the interface data model defined in this memo to their proprietary native data model. The new data model should be simple to facilitate such mappings.

- The data model should be suitable for new implementations to use as-is, without requiring a mapping to a different native model.

- References to interfaces should be as simple as possible, preferably by using a single leafref.

- The mapping to ifIndex [RFC2863] used by SNMP to identify interfaces must be clear.

- The model must support interface layering, both simple layering where one interface is layered on top of exactly one other interface, and more complex scenarios where one interface is aggregated over N other interfaces, or when N interfaces are multiplexed over one other interface.

- The data model should support the pre-provisioning of interface configuration, i.e., it should be possible to configure an interface whose physical interface hardware is not present on the device. It is recommended that devices that support dynamic addition and removal of physical interfaces also support pre-provisioning.

3. Interfaces Data Model

The data model in the module "ietf-interfaces" has the following structure, where square brackets are used to enclose a list's keys, "?" means that the leaf is optional, and "*" denotes a leaf-list:

```
   +--rw interfaces
      +--rw interface [name]
         +--rw name                          string
         +--rw description?                  string
         +--rw type                          ianaift:iana-if-type
         +--rw location?                     string
         +--rw enabled?                      boolean
         +--ro oper-status?                  enumeration
         +--ro last-change?                  yang:date-and-time
         +--ro if-index?                     int32
         +--rw link-up-down-trap-enable?     enumeration
         +--ro phys-address?                 yang:phys-address
         +--ro higher-layer-if*              interface-ref
         +--ro lower-layer-if*               interface-ref
         +--ro speed?                        yang:gauge64
         +--ro statistics
            +--ro discontinuity-time?   yang:date-and-time
            +--ro in-octets?            yang:counter64
            +--ro in-unicast-pkts?      yang:counter64
            +--ro in-broadcast-pkts?    yang:counter64
            +--ro in-multicast-pkts?    yang:counter64
            +--ro in-discards?          yang:counter32
            +--ro in-errors?            yang:counter32
            +--ro in-unknown-protos?    yang:counter32
            +--ro out-octets?           yang:counter64
            +--ro out-unicast-pkts?     yang:counter64
            +--ro out-broadcast-pkts?   yang:counter64
            +--ro out-multicast-pkts?   yang:counter64
            +--ro out-discards?         yang:counter32
            +--ro out-errors?           yang:counter32
```

3.1. The interface List

The data model for interfaces presented in this document uses a flat list of interfaces. Each interface in the list is identified by its name. Furthermore, each interface has a mandatory "type" leaf, and a "location" leaf. The combination of "type" and "location" is unique within the interface list.

It is expected that interface type specific data models augment the interface list, and use the "type" leaf to make the augmentation conditional.

As an example of such an interface type specific augmentation, consider this YANG snippet. For a more complete example, see Appendix A.

```
import interfaces {
    prefix "if";
}

augment "/if:interfaces/if:interface" {
    when "if:type = 'ethernetCsmacd'";

    container ethernet {
        leaf duplex {
            ...
        }
    }
}
```

The "location" leaf is a string. It is optional in the data model, but if the type represents a physical interface, it is mandatory. The format of this string is device- and type-dependent. The device uses the location string to identify the physical or logical entity that the configuration applies to. For example, if a device has a single array of 8 ethernet ports, the location can be one of the strings "1" to "8". As another example, if a device has N cards of M ports, the location can be on the form "n/m", such as "1/0".

How a client can learn which types and locations are present on a certain device is outside the scope of this document.

3.2. Interface References

An interface is identified by its name, which is unique within the server. This property is captured in the "interface-ref" typedef, which other YANG modules SHOULD use when they need to reference an existing interface.

3.3. Interface Layering

There is no generic mechanism for how an interface is configured to be layered on top of some other interface. It is expected that interface type specific models define their own data nodes for interface layering, by using "interface-ref" types to reference lower layers.

Below is an example of a model with such nodes. For a more complete example, see Appendix B.

```
    import interfaces {
        prefix "if";
    }

    augment "/if:interfaces/if:interface" {
        when "if:type = 'ieee8023adLag'";

        leaf-list slave-if {
            type if:interface-ref;
            must "/if:interfaces/if:interface[if:name = current()]"
               + "/if:type = 'ethernetCsmacd'" {
                description
                    "The type of a slave interface must be ethernet";
            }
        }
        // other bonding config params, failover times etc.
    }
```

Two state data leaf-lists, "higher-layer-if" and "lower-layer-if", represent a read-only view of the interface layering hierarchy.

4. Relationship to the IF-MIB

If the device implements IF-MIB [RFC2863], each entry in the "interface" list is typically mapped to one ifEntry. The "if-index" leaf contains the value of the corresponding ifEntry's ifIndex.

In most cases, the "name" of an "interface" entry is mapped to ifName. ifName is defined as an DisplayString [RFC2579] which uses a 7-bit ASCII character set. An implementation MAY restrict the allowed values for "name" to match the restrictions of ifName.

The IF-MIB allows two different ifEntries to have the same ifName. Devices that support this feature, and also support the configuration of these interfaces using the "interface" list, cannot have a 1-1 mapping between the "name" leaf and ifName.

The IF-MIB also defines the writable object ifPromiscuousMode. Since this object typically is not a configuration object, it is not mapped to the "ietf-interfaces" module.

The following table lists the YANG data nodes with corresponding objects in the IF-MIB.

YANG data node	IF-MIB object
interface	ifEntry
name	ifName
description	ifAlias
type	ifType
enabled	ifAdminStatus
oper-status	ifOperStatus
last-change	ifLastChange
if-index	ifIndex
link-up-down-trap-enable	ifLinkUpDownTrapEnable
phys-address	ifPhysAddress
higher-layer-if / lower-layer-if	ifStackTable
speed	ifSpeed
in-octets	ifHCInOctets
in-unicast-pkts	ifHCInUcastPkts
in-broadcast-pkts	ifHCInBroadcastPkts
in-multicast-pkts	ifHCInMulticastPkts
in-discards	ifInDiscards
in-errors	ifInErrors
in-unknown-protos	ifInUnknownProtos
out-octets	ifHCOutOctets
out-unicast-pkts	ifHCOutUcastPkts
out-broadcast-pkts	ifHCOutBroadcastPkts
out-multicast-pkts	ifHCOutMulticastPkts
out-discards	ifOutDiscards
out-errors	ifOutErrors

Mapping of YANG data nodes to IF-MIB objects

5. Interfaces YANG Module

This YANG module imports a typedef from
[I-D.ietf-netmod-iana-if-type].

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

<CODE BEGINS> file "ietf-interfaces@2013-02-06.yang"

```
module ietf-interfaces {

  namespace "urn:ietf:params:xml:ns:yang:ietf-interfaces";
  prefix if;

  import ietf-yang-types {
    prefix yang;
  }
  import iana-if-type {
    prefix ianaift;
  }

  organization
    "IETF NETMOD (NETCONF Data Modeling Language) Working Group";

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

     WG Chair: David Kessens
               <mailto:david.kessens@nsn.com>

     WG Chair: Juergen Schoenwaelder
               <mailto:j.schoenwaelder@jacobs-university.de>

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

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

     Copyright (c) 2012 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.: replace XXXX with actual RFC number and remove this
   // note.

   // RFC Ed.: update the date below with the date of RFC publication
   // and remove this note.
   revision 2013-02-06 {
     description
       "Initial revision.";
     reference
       "RFC XXXX: A YANG Data Model for Interface Management";
   }

   /* Typedefs */

   typedef interface-ref {
     type leafref {
       path "/if:interfaces/if:interface/if:name";
     }
     description
       "This type is used by data models that need to reference
        interfaces.";
   }

   /* Features */

   feature arbitrary-names {
     description
       "This feature indicates that the server allows interfaces to
        be named arbitrarily.";
   }

   feature if-mib {
     description
       "This feature indicates that the server implements IF-MIB.";
     reference
       "RFC 2863: The Interfaces Group MIB";
   }

   /* Data nodes */

   container interfaces {
```

```
      description
        "Interface parameters.";

      list interface {
        key "name";
        unique "type location";

        description
          "The list of interfaces on the device.";

        leaf name {
          type string;
          description
            "The name of the interface.

             A device MAY restrict the allowed values for this leaf,
             possibly depending on the type and location.

             If the device allows arbitrarily named interfaces, the
             feature 'arbitrary-names' is advertised.

             This leaf MAY be mapped to ifName by an implementation.
             Such an implementation MAY restrict the allowed values for
             this leaf so that it matches the restrictions of ifName.
             If a NETCONF server that implements this restriction is
             sent a value that doesn't match the restriction, it MUST
             reply with an rpc-error with the error-tag
             'invalid-value'.";
          reference
            "RFC 2863: The Interfaces Group MIB - ifName";
        }

        leaf description {
          type string;
          description
            "A textual description of the interface.

             This leaf MAY be mapped to ifAlias by an implementation.
             Such an implementation MAY restrict the allowed values for
             this leaf so that it matches the restrictions of ifAlias.
             If a NETCONF server that implements this restriction is
             sent a value that doesn't match the restriction, it MUST
             reply with an rpc-error with the error-tag
             'invalid-value'.";
          reference
            "RFC 2863: The Interfaces Group MIB - ifAlias";
        }
```

```
        leaf type {
          type ianaift:iana-if-type;
          mandatory true;
          description
            "The type of the interface.

             When an interface entry is created, a server MAY
             initialize the type leaf with a valid value, e.g., if it
             is possible to derive the type from the name of the
             interface.";
          reference
            "RFC 2863: The Interfaces Group MIB - ifType";
        }

        leaf location {
          type string;
          description
            "The device-specific location of the interface of a
             particular type.  The format of the location string
             depends on the interface type and the device.  If a
             NETCONF server is sent a value that doesn't match this
             format, it MUST reply with an rpc-error with the error-tag
             'invalid-value'.

             If the interface's type represents a physical interface,
             this leaf MUST be set.

             When an interface entry is created, a server MAY
             initialize the location leaf with a valid value, e.g., if
             it is possible to derive the location from the name of
             the interface.";
        }

        leaf enabled {
          type boolean;
          default "true";
          description
            "The desired state of the interface.

             This leaf contains the configured, desired state of the
             interface.  Systems that implement the IF-MIB use the
             value of this leaf to set IF-MIB.ifAdminStatus to 'up' or
             'down' after an ifEntry has been initialized, as described
             in RFC 2863.";
          reference
            "RFC 2863: The Interfaces Group MIB - ifAdminStatus";
        }
```

```
          leaf oper-status {
            type enumeration {
              enum up {
                value 1;
                description
                  "Ready to pass packets.";
              }
              enum down {
                value 2;
                description
                  "The interface does not pass any packets.";
              }
              enum testing {
                value 3;
                description
                  "In some test mode.  No operational packets can
                   be passed.";
              }
              enum unknown {
                value 4;
                description
                  "Status cannot be determined for some reason.";
              }
              enum dormant {
                value 5;
                description
                  "Waiting for some external event.";
              }
              enum not-present {
                value 6;
                description
                  "Some component (typically hardware) is missing.";
              }
              enum lower-layer-down {
                value 7;
                description
                  "Down due to state of lower-layer interface(s).";
              }
            }
            config false;
            description
              "The current operational state of the interface.

               If 'enabled' is 'false' then 'oper-status'
               should be 'down'.  If 'enabled' is changed to 'true'
               then 'oper-status' should change to 'up' if the interface
               is ready to transmit and receive network traffic; it
               should change to 'dormant' if the interface is waiting for
```

```
          external actions (such as a serial line waiting for an
          incoming connection); it should remain in the 'down' state
          if and only if there is a fault that prevents it from
          going to the 'up' state; it should remain in the
          'not-present' state if the interface has missing
          (typically, hardware) components.";
      reference
        "RFC 2863: The Interfaces Group MIB - ifOperStatus";
    }

    leaf last-change {
      type yang:date-and-time;
      config false;
      description
        "The time the interface entered its current operational
         state.  If the current state was entered prior to the
         last re-initialization of the local network management
         subsystem, then this node is not present.";
      reference
        "RFC 2863: The Interfaces Group MIB - ifLastChange";
    }

    leaf if-index {
      if-feature if-mib;
      type int32 {
        range "1..2147483647";
      }
      config false;
      description
        "The ifIndex value for the ifEntry represented by this
         interface.

         Media-specific modules must specify how the type is
         mapped to entries in the ifTable.";
      reference
        "RFC 2863: The Interfaces Group MIB - ifIndex";
    }

    leaf link-up-down-trap-enable {
      if-feature if-mib;
      type enumeration {
        enum enabled {
          value 1;
        }
        enum disabled {
          value 2;
        }
      }
```

```
        description
          "Indicates whether linkUp/linkDown SNMP notifications
           should be generated for this interface.

           If this node is not configured, the value 'enabled' is
           operationally used by the server for interfaces which do
           not operate on top of any other interface (i.e., there are
           no 'lower-layer-if' entries), and 'disabled' otherwise.";
        reference
          "RFC 2863: The Interfaces Group MIB -
                     ifLinkUpDownTrapEnable";
      }

      leaf phys-address {
        type yang:phys-address;
        config false;
        description
          "The interface's address at its protocol sub-layer.  For
          example, for an 802.x interface, this object normally
          contains a MAC address.  The interface's media-specific
          modules must define the bit and byte ordering and the
          format of the value of this object.  For interfaces that do
          not have such an address (e.g., a serial line), this node
          is not present.";
        reference
          "RFC 2863: The Interfaces Group MIB - ifPhysAddress";
      }

      leaf-list higher-layer-if {
        type interface-ref;
        config false;
        description
          "A list of references to interfaces layered on top of this
           interface.";
        reference
          "RFC 2863: The Interfaces Group MIB - ifStackTable";
      }

      leaf-list lower-layer-if {
        type interface-ref;
        config false;
        description
          "A list of references to interfaces layered underneath this
           interface.";
        reference
          "RFC 2863: The Interfaces Group MIB - ifStackTable";
      }
```

```
        leaf speed {
          type yang:gauge64;
          units "bits / second";
          config false;
          description
              "An estimate of the interface's current bandwidth in bits
               per second.  For interfaces which do not vary in
               bandwidth or for those where no accurate estimation can
               be made, this node should contain the nominal bandwidth.
               For interfaces that has no concept of bandwidth, this
               node is not present.";
          reference
            "RFC 2863: The Interfaces Group MIB -
                       ifSpeed, ifHighSpeed";
        }

        container statistics {
          config false;
          description
            "A collection of interface-related statistics objects.";

          leaf discontinuity-time {
            type yang:date-and-time;
            description
              "The time on the most recent occasion at which any one or
               more of this interface's counters suffered a
               discontinuity.  If no such discontinuities have occurred
               since the last re-initialization of the local management
               subsystem, then this node contains the time the local
               management subsystem re-initialized itself.";
          }

          leaf in-octets {
            type yang:counter64;
            description
              "The total number of octets received on the interface,
               including framing characters.

               Discontinuities in the value of this counter can occur
               at re-initialization of the management system, and at
               other times as indicated by the value of
               'discontinuity-time'.";
            reference
              "RFC 2863: The Interfaces Group MIB - ifHCInOctets";
          }
          leaf in-unicast-pkts {
            type yang:counter64;
            description
```

```
          "The number of packets, delivered by this sub-layer to a
           higher (sub-)layer, which were not addressed to a
           multicast or broadcast address at this sub-layer.

           Discontinuities in the value of this counter can occur
           at re-initialization of the management system, and at
           other times as indicated by the value of
           'discontinuity-time'.";
        reference
          "RFC 2863: The Interfaces Group MIB - ifHCInUcastPkts";
      }
      leaf in-broadcast-pkts {
        type yang:counter64;
        description
          "The number of packets, delivered by this sub-layer to a
           higher (sub-)layer, which were addressed to a broadcast
           address at this sub-layer.

           Discontinuities in the value of this counter can occur
           at re-initialization of the management system, and at
           other times as indicated by the value of
           'discontinuity-time'.";
        reference
          "RFC 2863: The Interfaces Group MIB -
                     ifHCInBroadcastPkts";
      }
      leaf in-multicast-pkts {
        type yang:counter64;
        description
          "The number of packets, delivered by this sub-layer to a
           higher (sub-)layer, which were addressed to a multicast
           address at this sub-layer.  For a MAC layer protocol,
           this includes both Group and Functional addresses.

           Discontinuities in the value of this counter can occur
           at re-initialization of the management system, and at
           other times as indicated by the value of
           'discontinuity-time'.";
        reference
          "RFC 2863: The Interfaces Group MIB -
                     ifHCInMulticastPkts";
      }
      leaf in-discards {
        type yang:counter32;
        description
          "The number of inbound packets which were chosen to be
           discarded even though no errors had been detected to
           prevent their being deliverable to a higher-layer
```

```
          protocol.  One possible reason for discarding such a
          packet could be to free up buffer space.

          Discontinuities in the value of this counter can occur
          at re-initialization of the management system, and at
          other times as indicated by the value of
          'discontinuity-time'.";
        reference
          "RFC 2863: The Interfaces Group MIB - ifInDiscards";
      }
      leaf in-errors {
        type yang:counter32;
        description
          "For packet-oriented interfaces, the number of inbound
           packets that contained errors preventing them from being
           deliverable to a higher-layer protocol.  For character-
           oriented or fixed-length interfaces, the number of
           inbound transmission units that contained errors
           preventing them from being deliverable to a higher-layer
           protocol.

           Discontinuities in the value of this counter can occur
           at re-initialization of the management system, and at
           other times as indicated by the value of
           'discontinuity-time'.";
        reference
          "RFC 2863: The Interfaces Group MIB - ifInErrors";
      }
      leaf in-unknown-protos {
        type yang:counter32;
        description
          "For packet-oriented interfaces, the number of packets
           received via the interface which were discarded because
           of an unknown or unsupported protocol.  For
           character-oriented or fixed-length interfaces that
           support protocol multiplexing the number of transmission
           units received via the interface which were discarded
           because of an unknown or unsupported protocol.  For any
           interface that does not support protocol multiplexing,
           this counter is not present.

           Discontinuities in the value of this counter can occur
           at re-initialization of the management system, and at
           other times as indicated by the value of
           'discontinuity-time'.";
        reference
          "RFC 2863: The Interfaces Group MIB - ifInUnknownProtos";
      }
```

```
          leaf out-octets {
            type yang:counter64;
            description
              "The total number of octets transmitted out of the
               interface, including framing characters.

               Discontinuities in the value of this counter can occur
               at re-initialization of the management system, and at
               other times as indicated by the value of
               'discontinuity-time'.";
            reference
              "RFC 2863: The Interfaces Group MIB - ifHCOutOctets";
          }
          leaf out-unicast-pkts {
            type yang:counter64;
            description
              "The total number of packets that higher-level protocols
               requested be transmitted, and which were not addressed
               to a multicast or broadcast address at this sub-layer,
               including those that were discarded or not sent.

               Discontinuities in the value of this counter can occur
               at re-initialization of the management system, and at
               other times as indicated by the value of
               'discontinuity-time'.";
            reference
              "RFC 2863: The Interfaces Group MIB - ifHCOutUcastPkts";
          }
          leaf out-broadcast-pkts {
            type yang:counter64;
            description
              "The total number of packets that higher-level protocols
               requested be transmitted, and which were addressed to a
               broadcast address at this sub-layer, including those
               that were discarded or not sent.

               Discontinuities in the value of this counter can occur
               at re-initialization of the management system, and at
               other times as indicated by the value of
               'discontinuity-time'.";
            reference
              "RFC 2863: The Interfaces Group MIB -
                         ifHCOutBroadcastPkts";
          }
          leaf out-multicast-pkts {
            type yang:counter64;
            description
              "The total number of packets that higher-level protocols
```

```
            requested be transmitted, and which were addressed to a
            multicast address at this sub-layer, including those
            that were discarded or not sent.  For a MAC layer
            protocol, this includes both Group and Functional
            addresses.

            Discontinuities in the value of this counter can occur
            at re-initialization of the management system, and at
            other times as indicated by the value of
            'discontinuity-time'.";
          reference
            "RFC 2863: The Interfaces Group MIB -
                       ifHCOutMulticastPkts";
        }
        leaf out-discards {
          type yang:counter32;
          description
            "The number of outbound packets which were chosen to be
             discarded even though no errors had been detected to
             prevent their being transmitted.  One possible reason
             for discarding such a packet could be to free up buffer
             space.

             Discontinuities in the value of this counter can occur
             at re-initialization of the management system, and at
             other times as indicated by the value of
             'discontinuity-time'.";
          reference
            "RFC 2863: The Interfaces Group MIB - ifOutDiscards";
        }
        leaf out-errors {
          type yang:counter32;
          description
            "For packet-oriented interfaces, the number of outbound
             packets that could not be transmitted because of errors.
             For character-oriented or fixed-length interfaces, the
             number of outbound transmission units that could not be
             transmitted because of errors.

             Discontinuities in the value of this counter can occur
             at re-initialization of the management system, and at
             other times as indicated by the value of
             'discontinuity-time'.";
          reference
            "RFC 2863: The Interfaces Group MIB - ifOutErrors";
        }
      }
    }
```

```
    }
  }
```

<CODE ENDS>

6. IANA Considerations

This document registers a URI in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made.

URI: urn:ietf:params:xml:ns:yang:ietf-interfaces

Registrant Contact: The IESG.

XML: N/A, the requested URI is an XML namespace.

This document registers a YANG module in the YANG Module Names registry [RFC6020].

```
name:         ietf-interfaces
namespace:    urn:ietf:params:xml:ns:yang:ietf-interfaces
prefix:       if
reference:    RFC XXXX
```

7. Security Considerations

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory-to-implement secure transport is SSH [RFC6242].

There are a number of data nodes defined in the YANG module which 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:

/interfaces/interface: This list specifies the configured interfaces on a device. Unauthorized access to this list could cause the device to ignore packets it should receive and process.

/interfaces/interface/enabled: This leaf controls if an interface is enabled or not. Unauthorized access to this leaf could cause the device to ignore packets it should receive and process.

8. Acknowledgments

The author wishes to thank Alexander Clemm, Per Hedeland, Ladislav Lhotka, and Juergen Schoenwaelder for their helpful comments.

9. References

9.1. Normative References

[I-D.ietf-netmod-iana-if-type]
Bjorklund, M., "IANA Interface Type and Address Family YANG Modules", draft-ietf-netmod-iana-if-type-02 (work in progress), April 2012.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2863] McCloghrie, K. and F. Kastenholz, "The Interfaces Group MIB", RFC 2863, June 2000.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

9.2. Informative References

[RFC2579] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Textual Conventions for SMIv2", STD 58, RFC 2579, April 1999.

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

Appendix A. Example: Ethernet Interface Module

This section gives a simple example of how an Ethernet interface module could be defined. It demonstrates how media-specific configuration parameters can be conditionally augmented to the generic interface list. It is not intended as a complete module for ethernet configuration.

```
module ex-ethernet {
  namespace "http://example.com/ethernet";
  prefix "eth";

  import ietf-interfaces {
    prefix if;
  }

  augment "/if:interfaces/if:interface" {
    when "if:type = 'ethernetCsmacd'";

    container ethernet {
      must "../if:location" {
        description
          "An ethernet interface must specify the physical location
           of the ethernet hardware.";
      }
      choice transmission-params {
        case auto {
          leaf auto-negotiate {
            type empty;
          }
        }
        case manual {
          leaf duplex {
            type enumeration {
              enum "half";
              enum "full";
            }
          }
          leaf speed {
            type enumeration {
              enum "10Mb";
              enum "100Mb";
              enum "1Gb";
              enum "10Gb";
            }
          }
        }
      }
      // other ethernet specific params...
    }
  }
}
```

Appendix B. Example: Ethernet Bonding Interface Module

This section gives an example of how interface layering can be defined. An ethernet bonding interface is defined, which bonds several ethernet interfaces into one logical interface.

```
module ex-ethernet-bonding {
  namespace "http://example.com/ethernet-bonding";
  prefix "bond";

  import ietf-interfaces {
    prefix if;
  }

  augment "/if:interfaces/if:interface" {
    when "if:type = 'ieee8023adLag'";

    leaf-list slave-if {
      type if:interface-ref;
      must "/if:interfaces/if:interface[if:name = current()]"
         + "/if:type = 'ethernetCsmacd'" {
        description
          "The type of a slave interface must be ethernet.";
      }
    }
    leaf bonding-mode {
      type enumeration {
        enum round-robin;
        enum active-backup;
        enum broadcast;
      }
    }
    // other bonding config params, failover times etc.
  }
}
```

Appendix C. Example: VLAN Interface Module

This section gives an example of how a vlan interface module can be defined.

```
module ex-vlan {
  namespace "http://example.com/vlan";
  prefix "vlan";

  import ietf-interfaces {
    prefix if;
  }

  augment "/if:interfaces/if:interface" {
    when "if:type = 'ethernetCsmacd' or
          if:type = 'ieee8023adLag'";
    leaf vlan-tagging {
      type boolean;
      default false;
    }
  }

  augment "/if:interfaces/if:interface" {
    when "if:type = 'l2vlan'";

    leaf base-interface {
      type if:interface-ref;
      must "/if:interfaces/if:interface[if:name = current()]"
         + "/vlan:vlan-tagging = true" {
        description
          "The base interface must have vlan tagging enabled.";
      }
    }
    leaf vlan-id {
      type uint16 {
        range "1..4094";
      }
      must "../base-interface" {
        description
          "If a vlan-id is defined, a base-interface must
           be specified.";
      }
    }
  }
}
```

Appendix D. Example: NETCONF <get> reply

This section gives an example of a reply to the NETCONF <get> request for a device that implements the example data models above.

```
<rpc-reply
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
    message-id="101">
  <data>
    <interfaces
        xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">
      <interface>
        <name>eth0</name>
        <type>ethernetCsmacd</type>
        <location>0</location>
        <enabled>true</enabled>
        <if-index>2</if-index>
      </interface>
      <interface>
        <name>eth1</name>
        <type>ethernetCsmacd</type>
        <location>1</location>
        <enabled>true</enabled>
        <if-index>7</if-index>
        <vlan-tagging
            xmlns="http://example.com/vlan">true</vlan-tagging>
      </interface>
    </interfaces>
  </data>
</rpc-reply>
```

Appendix E. Examples: Interface Naming Schemes

This section gives examples of some implementation strategies.

E.1. Router with Restricted Interface Names

In this example, a router has support for 4 line cards, each with 8 ports. The slots for the cards are physically numbered from 0 to 3, and the ports on each card from 0 to 7. Each card has fast- or gigabit-ethernet ports.

The implementation restricts the names of the interfaces to one of "fastethernet-N/M" or "gigabitethernet-N/M". The "location" of an interface is a string on the form "N/M". The implementation auto-initializes the values for "type" and "location" based on the interface name.

An operator can configure a new interface by sending an <edit-config> containing:

```
<interface nc:operation="create">
  <name>fastethernet-1/0</name>
</interface>
```

When the server processes this request, it will set the leaf "type" to "ethernetCsmacd" and "location" to "1/0". Thus, if the client performs a <get-config> right after the <edit-config> above, it will get:

```
<interface>
  <name>fastethernet-1/0</name>
  <type>ethernetCsmacd</type>
  <location>1/0</location>
</interface>
```

If the client tries to change the location of this interface with an <edit-config> containing:

```
<interface nc:operation="merge">
  <name>fastethernet-1/0</name>
  <location>1/1</location>
</interface>
```

then the server will reply with an "invalid-value" error, since the new location does not match the name.

E.2. Router with Arbitrary Interface Names

In this example, a router has support for 4 line cards, each with 8 ports. The slots for the cards are physically numbered from 0 to 3, and the ports on each card from 0 to 7. Each card has fast- or gigabit-ethernet ports.

The implementation does not restrict the interface names. This allows to more easily apply the interface configuration to a different physical interface. However, the additional level of indirection also makes it a bit more complex to map interface names found in other protocols to configuration entries. The "location" of an interface is a string on the form "N/M".

An operator can configure a new interface by sending an <edit-config> containing:

```
<interface nc:operation="create">
  <name>acme-interface</name>
  <type>ethernetCsmacd</type>
  <location>1/0</location>
</interface>
```

If necessary, the operator can move the configuration named "acme-interface" over to a different physical interface with an <edit-config> containing:

```
<interface nc:operation="merge">
  <name>acme-interface</name>
  <location>2/4</location>
</interface>
```

E.3. Ethernet Switch with Restricted Interface Names

In this example, an ethernet switch has a number of ports, each port identified by a simple port number.

The implementation restricts the interface names to numbers that match the physical port number.

An operator can configure a new interface by sending an <edit-config> containing:

```
<interface nc:operation="create">
  <name>6</name>
</interface>
```

When the server processes this request, it will set the leaf "type"

to "ethernetCsmacd" and "location" to "6". Thus, if the client performs a <get-config> right after the <edit-config> above, it will get:

```
  <interface>
    <name>6</name>
    <type>ethernetCsmacd</type>
    <location>6</location>
  </interface>
```

If the client tries to change the location of this interface with an <edit-config> containing:

```
  <interface nc:operation="merge">
    <name>6</name>
    <location>5</location>
  </interface>
```

then the server will reply with an "invalid-value" error, since the new location does not match the name.

E.4. Generic Host with Restricted Interface Names

In this example, a generic host has interfaces named by the kernel and without easily usable location information. The system identifies the physical interface by the name assigned by the operating system to the interface.

The implementation restricts the interface name to the operating system level name of the physical interface.

An operator can configure a new interface by sending an <edit-config> containing:

```
  <interface nc:operation="create">
    <name>eth8</name>
  </interface>
```

When the server processes this request, it will set the leaf "type" to "ethernetCsmacd" and "location" to "eth8". Thus, if the client performs a <get-config> right after the <edit-config> above, it will get:

```
  <interface>
    <name>eth8</name>
    <type>ethernetCsmacd</type>
    <location>eth8</location>
  </interface>
```

If the client tries to change the location of this interface with an <edit-config> containing:

```
<interface nc:operation="merge">
  <name>eth8</name>
  <location>eth7</location>
</interface>
```

then the server will reply with an "invalid-value" error, since the new location does not match the name.

E.5. Generic Host with Arbitrary Interface Names

In this example, a generic host has interfaces named by the kernel and without easily usable location information. The system identifies the physical interface by the name assigned by the operating system to the interface.

The implementation does not restrict the interface name to the operating system level name of the physical interface. This allows to more easily apply the interface configuration to a different physical interface. However, the additional level of indirection also makes it a bit more complex to map interface names found in other protocols to configuration entries.

An operator can configure a new interface by sending an <edit-config> containing:

```
<interface nc:operation="create">
  <name>acme-interface</name>
  <type>ethernetCsmacd</type>
  <location>eth4</location>
</interface>
```

If necessary, the operator can move the configuration named "acme-interface" over to a different physical interface with an <edit-config> containing:

```
<interface nc:operation="merge">
  <name>acme-interface</name>
  <location>eth3</location>
</interface>
```

Appendix F. ChangeLog

RFC Editor: remove this section upon publication as an RFC.

F.1. Version -08

- Removed the mtu leaf.
- Added examples of different interface naming schemes.

F.2. Version -07

- Made leaf speed config false.

F.3. Version -06

- Added oper-status leaf.
- Added leaf-lists higher-layer-if and lower-layer-if, that show the interface layering.
- Added container statistics with counters.

F.4. Version -05

- Added an Informative References section.
- Updated the Security Considerations section.
- Clarified the behavior of an NETCONF server when invalid values are received.

F.5. Version -04

- Clarified why ifPromiscuousMode is not part of this data model.
- Added a table that shows the mapping between this YANG data model and IF-MIB.

F.6. Version -03

- Added the section Relationship to the IF-MIB.
- Changed if-index to be a leaf instead of leaf-list.
- Explained the notation used in the data model tree picture.

F.7. Version -02

- Editorial fixes

F.8. Version -01

- Changed leaf "if-admin-status" to leaf "enabled".
- Added Security Considerations

Author's Address

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com