

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 16, 2013

M. Bjorklund
Tail-f Systems
May 15, 2013

A YANG Data Model for Interface Management
draft-ietf-netmod-interfaces-cfg-11

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. The data model includes configuration data, state data and counters for the collection of statistics.

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 November 16, 2013.

Copyright Notice

Copyright (c) 2013 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	3
1.1.	Terminology	3
1.2.	Tree Diagrams	3
2.	Objectives	5
3.	Interfaces Data Model	6
3.1.	The interface Lists	6
3.2.	Interface References	7
3.3.	Interface Layering	7
4.	Relationship to the IF-MIB	9
5.	Interfaces YANG Module	11
6.	IANA Considerations	26
7.	Security Considerations	27
8.	Acknowledgments	28
9.	References	29
9.1.	Normative References	29
9.2.	Informative References	29
Appendix A.	Example: Ethernet Interface Module	30
Appendix B.	Example: Ethernet Bonding Interface Module	32
Appendix C.	Example: VLAN Interface Module	33
Appendix D.	Example: NETCONF <get> reply	34
Appendix E.	Examples: Interface Naming Schemes	37
E.1.	Router with Restricted Interface Names	37
E.2.	Router with Arbitrary Interface Names	38
E.3.	Ethernet Switch with Restricted Interface Names	39
E.4.	Generic Host with Restricted Interface Names	39
E.5.	Generic Host with Arbitrary Interface Names	40
Appendix F.	ChangeLog	42
F.1.	Version -11	42
F.2.	Version -08	42
F.3.	Version -07	42
F.4.	Version -06	42
F.5.	Version -05	42
F.6.	Version -04	43
F.7.	Version -03	43
F.8.	Version -02	43
F.9.	Version -01	43
	Author's Address	44

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, configured, and monitored.

The data model includes configuration data, state data and counters for the collection of statistics.

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:

- o client
- o configuration data
- o server
- o state data

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

- o augment

- o data model
- o data node

[1.2.](#) Tree Diagrams

A simplified graphical representation of the data model is used in this document. The meaning of the symbols in these diagrams is as follows:

- o Brackets "[" and "]" enclose list keys.
- o Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- o Symbols after data node names: "?" means an optional node and "*" denotes a "list" and "leaf-list".
- o Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").
- o Ellipsis ("...") stands for contents of subtrees that are not shown.

[2.](#) Objectives

This section describes some of the design objectives for the model presented in [Section 5](#).

- o 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 data model should be simple to facilitate such mappings.
- o The data model should be suitable for new implementations to use as-is, without requiring a mapping to a different native model.
- o References to interfaces should be as simple as possible, preferably by using a single leafref.
- o The mapping to ifIndex [[RFC2863](#)] used by SNMP to identify interfaces must be clear.
- o 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 results

from the aggregation of N other interfaces, or when N interfaces are multiplexed over one other interface.

- o 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.
- o The data model should support both physical interfaces as well as logical interfaces.
- o The data model should include read-only counters in order to gather statistics for octets, packets and errors, sent and received.

[3.](#) Interfaces Data Model

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

```
+--rw interfaces
|  +--rw interface* [name]
|    +--rw name                string
|    +--rw description?       string
|    +--rw type                ianaift:iana-if-type
|    +--rw enabled?           boolean
|    +--rw link-up-down-trap-enable? enumeration
+--ro interfaces-state
  +--ro interface* [name]
    +--ro name                string
```

```

+--ro type                ianaift:iana-if-type
+--ro admin-status        enumeration
+--ro oper-status         enumeration
+--ro last-change?        yang:date-and-time
+--ro if-index            int32
+--ro phys-address?       yang:phys-address
+--ro higher-layer-if*    interface-state-ref
+--ro lower-layer-if*     interface-state-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 Lists

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.

There is one list of configured interfaces ("/interfaces/interface"), and a separate list for the operational state of all interfaces

("/interfaces-state/interface").

It is expected that interface type specific data models augment the interface lists, 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 {
      ...
    }
  }
}
```

For physical interfaces, the "name" is the device-specific name of the interface. It is used to identify the physical hardware interface. The 'config false' list "/interfaces-state/interface" contains all currently existing interfaces on the device.

If the device supports arbitrarily named logical interfaces, the NETCONF server advertises the feature "arbitrary-names". If the device does not advertise this feature, the names of logical interfaces MUST match the device's naming scheme. How a client can learn the naming scheme of such devices 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" and "interface-state-ref" typedefs, which other YANG modules SHOULD use when they need to reference a configured interface or operationally used interface, respectively.

[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 `"/interfaces-state/interface"` list is typically mapped to one `ifEntry`. The `"if-index"` leaf MUST contain 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 MUST 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 data model defined in this document, cannot have a 1-1 mapping between the `"name"` leaf and `ifName`.

The configured `"description"` of an `"interface"` has traditionally been mapped to `ifAlias` in some implementations. This document allows this mapping, but implementers should be aware of the differences in the value space and persistence for these objects. See the YANG module definition of the leaf `"description"` in [Section 5](#) for details.

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.

There are a number of counters in the IF-MIB that exist in two versions; one with 32 bits and one with 64 bits. The YANG module contains the 64 bits counters only. Note that NETCONF and SNMP may differ in the time granularity in which they provide access to the counters. For example, it is common that SNMP implementations cache counter values for some time.

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 / admin-status	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

YANG data nodes and related IF-MIB objects

5. Interfaces YANG Module

This YANG module imports typedefs from [[I-D.ietf-netmod-rfc6021-bis](#)] and [[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@2013-05-15.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) 2013 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

Bjorklund

Expires November 16, 2013

[Page 11]

Internet-Draft

YANG Interface Management

May 2013

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 2013-05-15 {  
  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
  configured interfaces.";
}

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

/* Features */

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

feature pre-provisioning {

```

```

description
  "This feature indicates that the device supports
  pre-provisioning of interface configuration, i.e., it is
  possible to configure an interface whose physical interface
  hardware is not present on the device.";
}

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

/* Data nodes */

container interfaces {
  description

```

```

    "Interface configuration parameters.";

list interface {
    key "name";

    description
        "The list of configured interfaces on the device.

        The operational state of an interface is available in the
        /interfaces-state/interface list. If the configuration of a
        physical interface cannot be used by the system (e.g., the
        physical interface present is not matching the interface
        type), then the configuration is not applied to the physical
        interface shown in the /interfaces-state/interface list. If
        the the configuration of a logical interface cannot be used
        by the system, the configured interface is not instantiated
        in the /interfaces-state/interface list.";

    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 of the interface.

            For physical interfaces, this leaf is the device-specific
            name of the interface. The 'config false' list
            /interfaces-state/interface contains the currently
            existing interfaces on the device."
    }
}

```

If a client tries to create configuration for a physical interface that is not present, the server MAY reject the request, if the implementation does not support pre-provisioning of interfaces, or if the name refers to an interface that can never exist in the system. A NETCONF server MUST reply with an rpc-error with the error-tag 'invalid-value' in this case.

If the device supports pre-provisioning of interface configuration, the feature 'pre-provisioning' is advertised.

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

When a configured logical interface is created by the system, it is instantiated in the /interface-state/interface list. Since the name in that list MAY be mapped to ifName by an implementation, such an implementation MUST 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'.");

}

leaf description {

 type string;

 description

 "A textual description of the interface.

This leaf MAY be mapped to ifAlias by an implementation. Such an implementation MUST 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'.

Since ifAlias is defined to be stored in non-volatile storage, the SNMP implementation MUST map ifAlias to the value of 'description' in the persistently stored datastore.

Specifically, if the device supports ':startup', when ifAlias is read the device MUST return the value of 'description' in the 'startup' datastore, and when it is written, it MUST be written to the 'running' and 'startup'

datastores. Note that it is up to the implementation if it modifies this single leaf in 'startup', or if it performs an implicit copy-config from 'running' to 'startup'.

If the device does not support ':startup', ifAlias MUST be mapped to the 'description' leaf in the 'running' datastore.";

```
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.

If a client tries to set the type of an interface to a value that can never be used by the system, e.g., if the type is not supported or if the type does not match the name of the interface, the server MUST reject the request. A NETCONF server MUST reply with an rpc-error with the error-tag 'invalid-value' in this case.";

```
reference
  "RFC 2863: The Interfaces Group MIB - ifType";
}
```

```
leaf enabled {
  type boolean;
  default "true";
  description
    "This leaf contains the configured, desired state of the
    interface.
```

Systems that implement the IF-MIB use the value of this leaf in the 'running' datastore to set IF-MIB.ifAdminStatus to 'up' or 'down' after an ifEntry has been initialized, as described in [RFC 2863](#).

```
        Changes in this leaf in the 'running' datastore are
        reflected in ifAdminStatus, but if ifAdminStatus is
        changed over SNMP, this leaf is not affected.";
    reference
        "RFC 2863: The Interfaces Group MIB - ifAdminStatus";
}

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

container interfaces-state {
    config false;
    description
        "Data nodes for the operational state of interfaces.";

    list interface {
        key "name";

        description
            "The list of interfaces on the device.

            Physical interfaces detected by the system are always
            present in this list, if they are configured or not.";

        leaf name {
```

```
type string;
```

```
description
  "The name of the interface.

  This leaf MAY be mapped to ifName by an implementation.
  Such an implementation MUST restrict the values
  for this leaf so that it matches the restrictions of
  ifName.";
reference
  "RFC 2863: The Interfaces Group MIB - ifName";
}

leaf type {
  type ianaift:iana-if-type;
  mandatory true;
  description
    "The type of the interface.";
  reference
    "RFC 2863: The Interfaces Group MIB - ifType";
}

leaf admin-status {
  if-feature if-mib;
  type enumeration {
    enum up {
      value 1;
      description
        "Ready to pass packets.";
    }
    enum down {
      value 2;
      description
        "Not ready to pass packets and not in some test mode.";
    }
    enum testing {
      value 3;
      description
        "In some test mode.";
    }
  }
}
mandatory true;
```

```
description
    "The desired state of the interface.

    This leaf has the same semantics as ifAdminStatus.";
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).";
}
}
mandatory true;
description
    "The current operational state of the interface.

    This leaf has the same semantics as ifOperStatus.";
reference
    "RFC 2863: The Interfaces Group MIB - ifOperStatus";
}

```

```

leaf last-change {
    type yang:date-and-time;
    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";
    }
    mandatory true;
    description
        "The ifIndex value for the ifEntry represented by this
        interface.";
    reference
        "RFC 2863: The Interfaces Group MIB - ifIndex";
}

leaf phys-address {
    type yang:phys-address;
}

```

```

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-state-ref;
  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-state-ref;
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";
  description
    "An estimate of the interface's current bandwidth in bits
    per second. For interfaces that 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 have no concept of bandwidth, this
    node is not present.";
  reference

```

```

        "RFC 2863: The Interfaces Group MIB -
            ifSpeed, ifHighSpeed";
    }

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

        leaf discontinuity-time {
            type yang:date-and-time;
            mandatory true;
            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

Bjorklund

Expires November 16, 2013

[Page 24]

Internet-Draft

YANG Interface Management

May 2013

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](#)]. The NETCONF access control model [[RFC6536](#)] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

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.

Bjorklund

Expires November 16, 2013

[Page 27]

Internet-Draft

YANG Interface Management

May 2013

[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-06](#) (work in

progress), April 2013.

- [I-D.ietf-netmod-rfc6021-bis]
Schoenwaelder, J., "Common YANG Data Types",
[draft-ietf-netmod-rfc6021-bis-02](#) (work in progress),
May 2013.
- [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.
- [RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", [RFC 6536](#), March 2012.

[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 also shows how operational state parameters can be conditionally augmented to the operational interface list. The example 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;
  }

  // configuration parameters for ethernet interfaces
  augment "/if:interfaces/if:interface" {
    when "if:type = 'ethernetCsmacd'";

    container ethernet {
      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...
}
```

```
}

// operational state parameters for ethernet interfaces
augment "/if:interfaces-state/if:interface" {
  when "if:type = 'ethernetCsmacd'";

  container ethernet {
    leaf duplex {
      type enumeration {
        enum "half";
        enum "full";
      }
    }
    // 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"
      xmlns:vlan="http://example.com/vlan">

      <interface>
        <name>eth0</name>
        <type>ethernetCsmacd</type>
        <enabled>>false</enabled>
      </interface>

      <interface>
        <name>eth1</name>

```

```

    <type>ethernetCsmacd</type>
    <enabled>true</enabled>
    <vlan:vlan-tagging>true</vlan:vlan-tagging>
</interface>

<interface>
  <name>eth1.10</name>
  <type>l2vlan</type>
  <enabled>true</enabled>
  <vlan:base-interface>eth1</vlan:base-interface>
  <vlan:vlan-id>10</vlan:vlan-id>
</interface>

<interface>
  <name>lo1</name>
  <type>softwareLoopback</type>
  <enabled>true</enabled>
</interface>

</interfaces>

<interfaces-state
  xmlns="urn:ietf:params:xml:ns:yang:ietf-interfaces">

  <interface>
    <name>eth0</name>

```

```

    <type>ethernetCsmacd</type>
    <admin-status>down</admin-status>
    <oper-status>down</oper-status>
    <if-index>2</if-index>
    <phys-address>00:01:02:03:04:05</phys-address>
    <statistics>
      <discontinuity-time>2013-04-01T03:00:00Z</discontinuity-time>
      <!-- counters now shown here -->
    </statistics>
</interface>

<interface>
  <name>eth1</name>
  <type>ethernetCsmacd</type>
  <admin-status>up</admin-status>

```

```

<oper-status>up</oper-status>
<if-index>7</if-index>
<phys-address>00:01:02:03:04:06</phys-address>
<higher-layer-if>eth1.10</higher-layer-if>
<statistics>
  <discontinuity-time>2013-04-01T03:00:00Z</discontinuity-time>
  <!-- counters now shown here -->
</statistics>
</interface>

<interface>
  <name>eth1.10</name>
  <type>l2vlan</type>
  <admin-status>up</admin-status>
  <oper-status>up</oper-status>
  <if-index>9</if-index>
  <lower-layer-if>eth1</lower-layer-if>
  <statistics>
    <discontinuity-time>2013-04-01T03:00:00Z</discontinuity-time>
    <!-- counters now shown here -->
  </statistics>
</interface>

<!-- This interface is not configured -->
<interface>
  <name>eth2</name>
  <type>ethernetCsmacd</type>
  <admin-status>down</admin-status>
  <oper-status>down</oper-status>
  <if-index>8</if-index>
  <phys-address>00:01:02:03:04:07</phys-address>
  <statistics>
    <discontinuity-time>2013-04-01T03:00:00Z</discontinuity-time>

```

```

  <!-- counters now shown here -->
</statistics>
</interface>

<interface>
  <name>lo1</name>
  <type>softwareLoopback</type>
  <admin-status>up</admin-status>

```



```
<oper-status>up</oper-status>
<if-index>1</if-index>
<statistics>
  <discontinuity-time>2013-04-01T03:00:00Z</discontinuity-time>
  <!-- counters now shown here -->
</statistics>
</interface>

</interfaces-state>
</data>
</rpc-reply>
```

This section gives examples of some implementation strategies.

The examples make use of the example data model "ex-vlan" (see [Appendix C](#)) to show how logical interfaces can be configured.

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 device-specific names for these physical interfaces are "fastethernet-N/M" or "gigabitethernet-N/M".

The name of a vlan interface is restricted to the form "<physical-interface-name>.<subinterface-number>".

It is assumed that the operator is aware of this naming scheme. The implementation auto-initializes the value for "type" based on the interface name.

The NETCONF server does not advertise the 'arbitrary-names' feature in the <hello> message.

An operator can configure a physical 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". 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>
</interface>
```

The client can configure a vlan interface by sending an <edit-config> containing:

```
<interface nc:operation="create">
  <name>fastethernet-1/0.10005</name>
  <type>l2-vlan</type>
  <vlan:base-interface>fastethernet-1/0</vlan:base-interface>
  <vlan:vlan-id>5</vlan:vlan-id>
</interface>
```

If the client tries to change the type of the physical interface with an `<edit-config>` containing:

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

then the server will reply with an "invalid-value" error, since the new type 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 device-specific names for these physical interfaces are "fastethernet-N/M" or "gigabitethernet-N/M".

The implementation does not restrict the logical interface names. This allows to more easily apply the interface configuration to a different 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 NETCONF server advertises the 'arbitrary-names' feature in the `<hello>` message.

Physical interfaces are configured as in [Appendix E.1](#).

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

```
<interface nc:operation="create">
  <name>acme-interface</name>
  <type>l2-vlan</type>
  <vlan:base-interface>fastethernet-1/0</vlan:base-interface>
```

```
<vlan:vlan-id>5</vlan:vlan-id>
</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>
  <vlan:base-interface>fastethernet-1/1</vlan:base-interface>
</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 device-specific names for the physical interfaces are numbers that match the physical port number.

An operator can configure a physical 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". Thus, if the client performs a `<get-config>` right after the `<edit-config>` above, it will get:

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

[E.4.](#) Generic Host with Restricted Interface Names

In this example, a generic host has interfaces named by the kernel. The system identifies the physical interface by the name assigned by the operating system to the interface.

The name of a vlan interface is restricted to the form "`<physical-interface-name>:<vlan-number>`".

The NETCONF server does not advertise the 'arbitrary-names' feature in the `<hello>` message.

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

Bjorklund

Expires November 16, 2013

[Page 39]

Internet-Draft

YANG Interface Management

May 2013

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

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

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

The client can configure a vlan interface by sending an `<edit-config>` containing:

```
<interface nc:operation="create">
  <name>eth8:5</name>
  <type>l2-vlan</type>
  <vlan:base-interface>eth8</vlan:base-interface>
  <vlan:vlan-id>5</vlan:vlan-id>
</interface>
```

[E.5.](#) Generic Host with Arbitrary Interface Names

In this example, a generic host has interfaces named by the kernel. The system identifies the physical interface by the name assigned by the operating system to the interface.

The implementation does not restrict the logical interface names. This allows to more easily apply the interface configuration to a different 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 NETCONF server advertises the 'arbitrary-names' feature in the <hello> message.

Physical interfaces are configured as in [Appendix E.4](#).

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

```
<interface nc:operation="create">
  <name>acme-interface</name>
  <type>l2-vlan</type>
  <vlan:base-interface>eth8</vlan:base-interface>
  <vlan:vlan-id>5</vlan:vlan-id>
```

```
</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>
  <vlan:base-interface>eth3</vlan:base-interface>
</interface>
```

[Appendix F](#). ChangeLog

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

[F.1](#). Version -11

- o Separated the operational state from the configuration.
- o Removed 'location', and instead use the name to identify physical interfaces.
- o Added the feature 'pre-provisioning'.
- o Made 'oper-status' and 'if-index' mandatory in the data model.
- o Added 'admin-status'.
- o Clarified why description can be mapped to ifAlias.

- o Clarified that 64-bit counters only are used, where there exist 64-bit and 32-bit counters in IF-MIB.
- o Updated Security Considerations section with a reference to NACM.

[F.2.](#) Version -08

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

[F.3.](#) Version -07

- o Made leaf speed config false.

[F.4.](#) Version -06

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

[F.5.](#) Version -05

- o Added an Informative References section.

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

[F.6.](#) Version -04

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

[F.7.](#) Version -03

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

[F.8.](#) Version -02

- o Editorial fixes

[F.9.](#) Version -01

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

Author's Address

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com