                    The Interfaces Group MIB using SMIv2

Status of this Memo

   This document specifies an Internet standards track protocol for the
   Internet community, and requests discussion and suggestions for
   improvements. Please refer to the current edition of the "Internet
   Official Protocol Standards" (STD 1) for the standardization state
   and status of this protocol. Distribution of this memo is unlimited.

Table of Contents

1. Introduction

   This memo defines a portion of the Management Information Base
   (MIB) for use with network management protocols in the Internet
   community. In particular, it describes managed objects used for
   managing Network Interfaces.

   This memo discusses the 'interfaces' group of MIB-II, especially the
   experience gained from the definition of numerous media- specific MIB
   modules for use in conjunction with the 'interfaces' group for
   managing various sub-layers beneath the internetwork- layer. It
   specifies clarifications to, and extensions of, the architectural
   issues within the previous model used for the 'interfaces' group.

   This memo also includes a MIB module. As well as including new
   MIB definitions to support the architectural extensions, this MIB
   module also re-specifies the 'interfaces' group of MIB-II in a
   manner that is both compliant to the SNMPv2 SMI and semantically-
   identical to the existing SNMPv1-based definitions.

   The key words "MUST" and "MUST NOT" in this document are to be
   interpreted as described in RFC 2119 [10].

2. The SNMP Network Management Framework

   The SNMP Network Management Framework presently consists of three
   major components. They are:

   o   RFC 1902 which defines the SMI, the mechanisms used for
        describing and naming objects for the purpose of management.

   o   STD 17, RFC 1213 defines MIB-II, the core set of managed
        objects for the Internet suite of protocols.

   o   STD 15, RFC 1157 and RFC 1905 which define two versions of
        the protocol used for network access to managed objects.

The Framework permits new objects to be defined for the purpose of
experimentation and evaluation.

2.1. Object Definitions

Managed objects are accessed via a virtual information store,
termed the Management Information Base or MIB. Objects in the MIB
are defined using the subset of Abstract Syntax Notation One
(ASN.1) defined in the SMI. In particular, each object object
type is named by an OBJECT IDENTIFIER, an administratively
assigned name. The object type together with an object instance
serves to uniquely identify a specific instantiation of the
object. For human convenience, we often use a textual string,
termed the descriptor, to refer to the object type.

3. Experience with the Interfaces Group

One of the strengths of internetwork-layer protocols such as IP
[6] is that they are designed to run over any network interface.
In achieving this, IP considers any and all protocols it runs over
as a single "network interface" layer. A similar view is taken by
other internetwork-layer protocols. This concept is represented
in MIB-II by the 'interfaces' group which defines a generic set of
managed objects such that any network interface can be managed in
an interface-independent manner through these managed objects.
The 'interfaces' group provides the means for additional managed
objects specific to particular types of network interface (e.g., a
specific medium such as Ethernet) to be defined as extensions to
the 'interfaces' group for media-specific management. Since the
standardization of MIB-II, many such media-specific MIB modules
have been defined.

Experience in defining these media-specific MIB modules has shown
that the model defined by MIB-II is too simplistic and/or static
for some types of media-specific management. As a result, some of
these media-specific MIB modules assume an evolution or loosening
of the model. This memo documents and standardizes that evolution
of the model and fills in the gaps caused by that evolution. This
memo also incorporates the interfaces group extensions documented
in RFC 1229 [7].

3.1. Clarifications/Revisions

There are several areas for which experience has indicated that
clarification, revision, or extension of the model would be
helpful. The following sections discuss the changes in the
interfaces group adopted by this memo in each of these areas.

In some sections, one or more paragraphs contain discussion of
rejected alternatives to the model adopted in this memo. Readers
not familiar with the MIB-II model and not interested in the
rationale behind the new model may want to skip these paragraphs.

3.1.1. Interface Sub-Layers

Experience in defining media-specific management information has
shown the need to distinguish between the multiple sub-layers
beneath the internetwork-layer. In addition, there is a need to
manage these sub-layers in devices (e.g., MAC-layer bridges) which
are unaware of which, if any, internetwork protocols run over
these sub-layers. As such, a model of having a single conceptual
row in the interfaces table (MIB-II's ifTable) represent a whole
interface underneath the internetwork-layer, and having a single
associated media-specific MIB module (referenced via the ifType
object) is too simplistic. A further problem arises with the
value of the ifType object which has enumerated values for each
type of interface.

Consider, for example, an interface with PPP running over an HDLC
link which uses a RS232-like connector. Each of these sub-layers
has its own media-specific MIB module. If all of this is
represented by a single conceptual row in the ifTable, then an
enumerated value for ifType is needed for that specific
combination which maps to the specific combination of media-
specific MIBs. Furthermore, such a model still lacks a method to
describe the relationship of all the sub-layers of the MIB stack.

An associated problem is that of upward and downward multiplexing
of the sub-layers. An example of upward multiplexing is MLP
(Multi-Link-Procedure) which provides load-sharing over several
serial lines by appearing as a single point-to-point link to the
sub-layer(s) above. An example of downward multiplexing would be
several instances of PPP, each framed within a separate X.25
virtual circuit, all of which run over one fractional T1 channel,
concurrently with other uses of the T1 link. The MIB structure
must allow these sorts of relationships to be described.

Several solutions for representing multiple sub-layers were
rejected. One was to retain the concept of one conceptual row for
all the sub-layers of an interface and have each media-specific
MIB module identify its "superior" and "subordinate" sub-layers
through OBJECT IDENTIFIER "pointers". This scheme would have
several drawbacks: the superior/subordinate pointers would be
contained in the media-specific MIB modules; thus, a manager could
not learn the structure of an interface without inspecting
multiple pointers in different MIB modules; this would be overly

complex and only possible if the manager had knowledge of all the
relevant media-specific MIB modules; MIB modules would all need to
be retrofitted with these new "pointers"; this scheme would not
adequately address the problem of upward and downward
multiplexing; and finally, enumerated values of ifType would be
needed for each combination of sub-layers. Another rejected
solution also retained the concept of one conceptual row for all
the sub-layers of an interface but had a new separate MIB table to
identify the "superior" and "subordinate" sub-layers and to
contain OBJECT IDENTIFIER "pointers" to the media-specific MIB
module for each sub-layer. Effectively, one conceptual row in the
ifTable would represent each combination of sub-layers between the
internetwork-layer and the wire. While this scheme has fewer
drawbacks, it still would not support downward multiplexing, such
as PPP over MLP: observe that MLP makes two (or more) serial
lines appear to the layers above as a single physical interface,
and thus PPP over MLP should appear to the internetwork-layer as a
single interface; in contrast, this scheme would result in two (or
more) conceptual rows in the ifTable, both of which the
internetwork-layer would run over. This scheme would also require
enumerated values of ifType for each combination of sub-layers.

The solution adopted by this memo is to have an individual
conceptual row in the ifTable to represent each sub-layer, and
have a new separate MIB table (the ifStackTable, see section 6
below) to identify the "superior" and "subordinate" sub-layers
through INTEGER "pointers" to the appropriate conceptual rows in
the ifTable. This solution supports both upward and downward
multiplexing, allows the IANAifType to Media-Specific MIB mapping
to identify the media-specific MIB module for that sub-layer, such
that the new table need only be referenced to obtain information
about layering, and it only requires enumerated values of ifType
for each sub-layer, not for combinations of them. However, it
does require that the descriptions of some objects in the ifTable
(specifically, ifType, ifPhysAddress, ifInUcastPkts, and
ifOutUcastPkts) be generalized so as to apply to any sub-layer
(rather than only to a sub-layer immediately beneath the network
layer as previously), plus some (specifically, ifSpeed) which need
to have appropriate values identified for use when a generalized
definition does not apply to a particular sub-layer.

In addition, this adopted solution makes no requirement that a
device, in which a sub-layer is instrumented by a conceptual row
of the ifTable, be aware of whether an internetwork protocol runs
on top of (i.e., at some layer above) that sub-layer. In fact,
the counters of packets received on an interface are defined as
counting the number "delivered to a higher-layer protocol". This
meaning of "higher-layer" includes:

(1) Delivery to a forwarding module which accepts
     packets/frames/octets and forwards them on at the same
     protocol layer. For example, for the purposes of this
     definition, the forwarding module of a MAC-layer bridge is
     considered as a "higher-layer" to the MAC-layer of each port
     on the bridge.

(2) Delivery to a higher sub-layer within a interface stack. For
     example, for the purposes of this definition, if a PPP module
     operated directly over a serial interface, the PPP module
     would be considered the higher sub-layer to the serial
     interface.

(3) Delivery to a higher protocol layer which does not do packet
     forwarding for sub-layers that are "at the top of" the
     interface stack. For example, for the purposes of this
     definition, the local IP module would be considered the
     higher layer to a SLIP serial interface.

Similarly, for output, the counters of packets transmitted out an
interface are defined as counting the number "that higher-level
protocols requested to be transmitted". This meaning of "higher-
layer" includes:

(1) A forwarding module, at the same protocol layer, which
     transmits packets/frames/octets that were received on an
     different interface. For example, for the purposes of this
     definition, the forwarding module of a MAC-layer bridge is
     considered as a "higher-layer" to the MAC-layer of each port
     on the bridge.

(2) The next higher sub-layer within an interface stack. For
     example, for the purposes of this definition, if a PPP module
     operated directly over a serial interface, the PPP module
     would be a "higher layer" to the serial interface.

(3) For sub-layers that are "at the top of" the interface stack,
     a higher element in the network protocol stack. For example,
     for the purposes of this definition, the local IP module
     would be considered the higher layer to an Ethernet
     interface.

3.1.2. Guidance on Defining Sub-layers

   The designer of a media-specific MIB must decide whether to divide
   the interface into sub-layers or not, and if so, how to make the
   divisions. The following guidance is offered to assist the
   media-specific MIB designer in these decisions.

In general, the number of entries in the ifTable should be kept to
the minimum required for network management. In particular, a
group of related interfaces should be treated as a single
interface with one entry in the ifTable providing that:

(1) None of the group of interfaces performs multiplexing for any
     other interface in the agent,
(2) There is a meaningful and useful way for all of the ifTable's
     information (e.g., the counters, and the status variables),
     and all of the ifTable's capabilities (e.g., write access to
     ifAdminStatus), to apply to the group of interfaces as a
     whole.

Under these circumstances, there should be one entry in the
ifTable for such a group of interfaces, and any internal structure
which needs to be represented to network management should be
captured in a MIB module specific to the particular type of
interface.

Note that application of bullet 2 above to the ifTable's ifType
object requires that there is a meaningful media-specific MIB and
a meaningful ifType value which apply to the group of interfaces
as a whole. For example, it is not appropriate to treat an HDLC
sub-layer and an RS-232 sub-layer as a single ifTable entry when
the media-specific MIBs and the ifType values for HDLC and RS-232
are separate (rather than combined).

Subject to the above, it is appropriate to assign an ifIndex value
to any interface that can occur in an interface stack (in the
ifStackTable) where the bottom of the stack is a physical
interface (ifConnectorPresent has the value 'true') and there is a
layer-3 or other application that "points down" to the top of this
stack. An example of an application that points down to the top
of the stack is the Character MIB [9].

Note that the sub-layers of an interface on one device will
sometimes be different from the sub-layers of the interconnected
interface of another device; for example, for a frame-relay DTE
interface connected a frameRelayService interface, the inter-
connected DTE and DCE interfaces have different ifType values and
media-specific MIBs.

These guidelines are just that, guidelines. The designer of a
media-specific MIB is free to lay out the MIB in whatever SMI
conformant manner is desired. However, in doing so, the media-
specific MIB MUST completely specify the sub-layering model used
for the MIB, and provide the assumptions, reasoning, and rationale
used to develop that model.

3.1.3. Virtual Circuits

   Several of the sub-layers for which media-specific MIB modules
   have been defined are connection oriented (e.g., Frame Relay,
   X.25). Experience has shown that each effort to define such a MIB
   module revisits the question of whether separate conceptual rows
   in the ifTable are needed for each virtual circuit. Most, if not
   all, of these efforts to date have decided to have all virtual
   circuits reference a single conceptual row in the ifTable.

   This memo strongly recommends that connection-oriented sub-layers
   do not have a conceptual row in the ifTable for each virtual
   circuit. This avoids the proliferation of conceptual rows,
   especially those which have considerable redundant information.
   (Note, as a comparison, that connection-less sub-layers do not
   have conceptual rows for each remote address.) There may,
   however, be circumstances under which it is appropriate for a
   virtual circuit of a connection-oriented sub-layer to have its own
   conceptual row in the ifTable; an example of this might be PPP
   over an X.25 virtual circuit. The MIB in section 6 of this memo
   supports such circumstances.

   If a media-specific MIB wishes to assign an entry in the ifTable
   to each virtual circuit, the MIB designer must present the
   rationale for this decision in the media-specific MIB's
   specification.

3.1.4. Bit, Character, and Fixed-Length Interfaces

   RS-232 is an example of a character-oriented sub-layer over which
   (e.g., through use of PPP) IP datagrams can be sent. Due to the
   packet-based nature of many of the objects in the ifTable,
   experience has shown that it is not appropriate to have a
   character-oriented sub-layer represented by a whole conceptual row
   in the ifTable.

   Experience has also shown that it is sometimes desirable to have
   some management information for bit-oriented interfaces, which are
   similarly difficult to represent by a whole conceptual row in the
   ifTable. For example, to manage the channels of a DS1 circuit,
   where only some of the channels are carrying packet-based data.

   A further complication is that some subnetwork technologies
   transmit data in fixed length transmission units. One example of
   such a technology is cell relay, and in particular Asynchronous
   Transfer Mode (ATM), which transmits data in fixed-length cells.
   Representing such a interface as a packet-based interface produces

redundant objects if the relationship between the number of
packets and the number of octets in either direction is fixed by
the size of the transmission unit (e.g., the size of a cell).

About half the objects in the ifTable are applicable to every type
of interface: packet-oriented, character-oriented, and bit-
oriented. Of the other half, two are applicable to both
character-oriented and packet-oriented interfaces, and the rest
are applicable only to packet-oriented interfaces. Thus, while it
is desirable for consistency to be able to represent any/all types
of interfaces in the ifTable, it is not possible to implement the
full ifTable for bit- and character-oriented sub-layers.

A rejected solution to this problem would be to split the ifTable
into two (or more) new MIB tables, one of which would contain
objects that are relevant only to packet-oriented interfaces
(e.g., PPP), and another that may be used by all interfaces. This
is highly undesirable since it would require changes in every
agent implementing the ifTable (i.e., just about every existing
SNMP agent).

The solution adopted in this memo builds upon the fact that
compliance statements in SNMPv2 (in contrast to SNMPv1) refer to
object groups, where object groups are explicitly defined by
listing the objects they contain. Thus, in SNMPv2, multiple
compliance statements can be specified, one for all interfaces and
additional ones for specific types of interfaces. The separate
compliance statements can be based on separate object groups,
where the object group for all interfaces can contain only those
objects from the ifTable which are appropriate for every type of
interfaces. Using this solution, every sub-layer can have its own
conceptual row in the ifTable.

Thus, section 6 of this memo contains definitions of the objects
of the existing 'interfaces' group of MIB-II, in a manner which is
both SNMPv2-compliant and semantically-equivalent to the existing
MIB-II definitions. With equivalent semantics, and with the BER
("on the wire") encodings unchanged, these definitions retain the
same OBJECT IDENTIFIER values as assigned by MIB-II. Thus, in
general, no rewrite of existing agents which conform to MIB-II and
the ifExtensions MIB is required.

In addition, this memo defines several object groups for the
purposes of defining which objects apply to which types of
interface:

    (1)   the ifGeneralInformationGroup. This group contains those
        objects applicable to all types of network interfaces,
        including bit-oriented interfaces.

    (2)   the ifPacketGroup. This group contains those objects
        applicable to packet-oriented network interfaces.

    (3)   the ifFixedLengthGroup. This group contains the objects
        applicable not only to character-oriented interfaces, such as
        RS-232, but also to those subnetwork technologies, such as
        cell-relay/ATM, which transmit data in fixed length
        transmission units. As well as the octet counters, there are
        also a few other counters (e.g., the error counters) which
        are useful for this type of interface, but are currently
        defined as being packet-oriented. To accommodate this, the
        definitions of these counters are generalized to apply to
        character-oriented interfaces and fixed-length-transmission
        interfaces.

It should be noted that the octet counters in the ifTable
aggregate octet counts for unicast and non-unicast packets into a
single octet counter per direction (received/transmitted). Thus,
with the above definition of fixed-length-transmission interfaces,
where such interfaces which support non-unicast packets, separate
counts of unicast and multicast/broadcast transmissions can only
be maintained in a media-specific MIB module.

3.1.5. Interface Numbering

MIB-II defines an object, ifNumber, whose value represents:

    "The number of network interfaces (regardless of their
    current state) present on this system."

Each interface is identified by a unique value of the ifIndex
object, and the description of ifIndex constrains its value as
follows:

    "Its value ranges between 1 and the value of ifNumber. The
    value for each interface must remain constant at least from
    one re-initialization of the entity's network management
    system to the next re-initialization."

This constancy requirement on the value of ifIndex for a
particular interface is vital for efficient management. However,
an increasing number of devices allow for the dynamic
addition/removal of network interfaces. One example of this is a
dynamic ability to configure the use of SLIP/PPP over a

character-oriented port. For such dynamic additions/removals, the
combination of the constancy requirement and the restriction that
the value of ifIndex is less than ifNumber is problematic.

Redefining ifNumber to be the largest value of ifIndex was
rejected since it would not help. Such a re-definition would
require ifNumber to be deprecated and the utility of the redefined
object would be questionable. Alternatively, ifNumber could be
deprecated and not replaced. However, the deprecation of ifNumber
would require a change to that portion of ifIndex's definition
which refers to ifNumber. So, since the definition of ifIndex
must be changed anyway in order to solve the problem, changes to
ifNumber do not benefit the solution.

The solution adopted in this memo is just to delete the
requirement that the value of ifIndex must be less than the value
of ifNumber, and to retain ifNumber with its current definition.
This is a minor change in the semantics of ifIndex; however, all
existing agent implementations conform to this new definition, and
in the interests of not requiring changes to existing agent
implementations and to the many existing media-specific MIBs, this
memo assumes that this change does not require ifIndex to be
deprecated. Experience indicates that this assumption does
"break" a few management applications, but this is considered
preferable to breaking all agent implementations.

This solution also results in the possibility of "holes" in the
ifTable, i.e., the ifIndex values of conceptual rows in the
ifTable are not necessarily contiguous, but SNMP's GetNext (and
SNMPv2's GetBulk) operation easily deals with such holes. The
value of ifNumber still represents the number of conceptual rows,
which increases/decreases as new interfaces are dynamically
added/removed.

The requirement for constancy (between re-initializations) of an
interface's ifIndex value is met by requiring that after an
interface is dynamically removed, its ifIndex value is not re-used
by a *different* dynamically added interface until after the
following re-initialization of the network management system.
This avoids the need for assignment (in advance) of ifIndex values
for all possible interfaces that might be added dynamically. The
exact meaning of a "different" interface is hard to define, and
there will be gray areas. Any firm definition in this document
would likely to turn out to be inadequate. Instead, implementors
must choose what it means in their particular situation, subject
to the following rules:

     (1)   a previously-unused value of ifIndex must be assigned to a
           dynamically added interface if an agent has no knowledge of
           whether the interface is the "same" or "different" to a
           previously incarnated interface.

     (2)   a management station, not noticing that an interface has gone
           away and another has come into existence, must not be
           confused when calculating the difference between the counter
           values retrieved on successive polls for a particular ifIndex
           value.

     When the new interface is the same as an old interface, but a
     discontinuity in the value of the interface's counters cannot be
     avoided, the ifTable has (until now) required that a new ifIndex
     value be assigned to the returning interface. That is, either all
     counter values have had to be retained during the absence of an
     interface in order to use the same ifIndex value on that
     interface's return, or else a new ifIndex value has had to be
     assigned to the returning interface. Both alternatives have
     proved to be burdensome to some implementations:

     (1)   maintaining the counter values may not be possible (e.g., if
           they are maintained on removable hardware),

     (2)   using a new ifIndex value presents extra work for management
           applications. While the potential need for such extra work
           is unavoidable on agent re-initializations, it is desirable
           to avoid it between re-initializations.

     To address this, a new object, ifCounterDiscontinuityTime, has
     been defined to record the time of the last discontinuity in an
     interface's counters. By monitoring the value of this new object,
     a management application can now detect counter discontinuities
     without the ifIndex value of the interface being changed. Thus,
     an agent which implements this new object should, when a new
     interface is the same as an old interface, retain that interface's
     ifIndex value and update if necessary the interface's value of
     ifCounterDiscontinuityTime. With this new object, a management
     application must, when calculating differences between counter
     values retrieved on successive polls, discard any calculated
     difference for which the value of ifCounterDiscontinuityTime is
     different for the two polls. (Note that this test must be
     performed in addition to the normal checking of sysUpTime to
     detect an agent re-initialization.) Since such discards are a
     waste of network management processing and bandwidth, an agent
     should not update the value of ifCounterDiscontinuityTime unless
     absolutely necessary.

While defining this new object is a change in the semantics of the
ifTable counter objects, it is impractical to deprecate and
redefine all these counters because of their wide deployment and
importance. Also, a survey of implementations indicates that many
agents and management applications do not correctly implement this
aspect of the current semantics (because of the burdensome issues
mentioned above), such that the practical implications of such a
change is small. Thus, this breach of the SMI's rules is
considered to be acceptable.

Note, however, that the addition of ifCounterDiscontinuityTime
does not change the fact that:

> It is necessary at certain times for the assignment of ifIndex
> values to change on a reinitialization of the agent (such as a
> reboot).

The possibility of ifIndex value re-assignment must be
accommodated by a management application whenever the value of
sysUpTime is reset to zero.

Note also that some agents support multiple "naming scopes", e.g.,
for an SNMPv1 agent, multiple values of the SNMPv1 community
string. For such an agent (e.g., a CNM agent which supports a
different subset of interfaces for different customers), there is
no required relationship between the ifIndex values which identify
interfaces in one naming scope and those which identify interfaces
in another naming scope. It is the agent's choice as to whether
the same or different ifIndex values identify the same or
different interfaces in different naming scopes.

Because of the restriction of the value of ifIndex to be less than
ifNumber, interfaces have been numbered with small integer values.
This has led to the ability by humans to use the ifIndex values as
(somewhat) user-friendly names for network interfaces (e.g.,
"interface number 3"). With the relaxation of the restriction on
the value of ifIndex, there is now the possibility that ifIndex
values could be assigned as very large numbers (e.g., memory
addresses). Such numbers would be much less user-friendly.
Therefore, this memo recommends that ifIndex values still be
assigned as (relatively) small integer values starting at 1, even
though the values in use at any one time are not necessarily
contiguous. (Note that this makes remembering which values have
been assigned easy for agents which dynamically add new
interfaces).

A new problem is introduced by representing each sub-layer as an
ifTable entry. Previously, there usually was a simple, direct,
mapping of interfaces to the physical ports on systems. This
mapping would be based on the ifIndex value. However, by having
an ifTable entry for each interface sub-layer, mapping from
interfaces to physical ports becomes increasingly problematic.

To address this issue, a new object, ifName, is added to the MIB.
This object contains the device's local name (e.g., the name used
at the device's local console) for the interface of which the
relevant entry in the ifTable is a component. For example,
consider a router having an interface composed of PPP running over
an RS-232 port. If the router uses the name "wan1" for the
(combined) interface, then the ifName objects for the
corresponding PPP and RS-232 entries in the ifTable would both
have the value "wan1". On the other hand, if the router uses the
name "wan1.1" for the PPP interface and "wan1.2" for the RS-232
port, then the ifName objects for the corresponding PPP and RS-232
entries in the ifTable would have the values "wan1.1" and
"wan1.2", respectively. As an another example, consider an agent
which responds to SNMP queries concerning an interface on some
other (proxied) device: if such a proxied device associates a
particular identifier with an interface, then it is appropriate to
use this identifier as the value of the interface's ifName, since
the local console in this case is that of the proxied device.

In contrast, the existing ifDescr object is intended to contain a
description of an interface, whereas another new object, ifAlias,
provides a location in which a network management application can
store a non-volatile interface-naming value of its own choice.
The ifAlias object allows a network manager to give one or more
interfaces their own unique names, irrespective of any interface-
stack relationship. Further, the ifAlias name is non-volatile,
and thus an interface must retain its assigned ifAlias value
across reboots, even if an agent chooses a new ifIndex value for
the interface.

3.1.6. Counter Size

As the speed of network media increase, the minimum time in which
a 32 bit counter will wrap decreases. For example, a 10Mbs stream
of back-to-back, full-size packets causes ifInOctets to wrap in
just over 57 minutes; at 100Mbs, the minimum wrap time is 5.7
minutes, and at 1Gbs, the minimum is 34 seconds. Requiring that
interfaces be polled frequently enough not to miss a counter wrap
is increasingly problematic.

   A rejected solution to this problem was to scale the counters; for
   example, ifInOctets could be changed to count received octets in,
   say, 1024 byte blocks. While it would provide acceptable
   functionality at high rates of the counted-events, at low rates it
   suffers. If there is little traffic on an interface, there might
   be a significant interval before enough of the counted-events
   occur to cause the scaled counter to be incremented. Traffic
   would then appear to be very bursty, leading to incorrect
   conclusions of the network's performance.

   Instead, this memo adopts expanded, 64 bit, counters. These
   counters are provided in new "high capacity" groups. The old,
   32-bit, counters have not been deprecated. The 64-bit counters
   are to be used only when the 32-bit counters do not provide enough
   capacity; that is, when the 32 bit counters could wrap too fast.

   For interfaces that operate at 20,000,000 (20 million) bits per
   second or less, 32-bit byte and packet counters MUST be used. For
   interfaces that operate faster than 20,000,000 bits/second, and
   slower than 650,000,000 bits/second, 32-bit packet counters MUST
   be used and 64-bit octet counters MUST be used. For interfaces
   that operate at 650,000,000 bits/second or faster, 64-bit packet
   counters AND 64-bit octet counters MUST be used.

   These speed thresholds were chosen as reasonable compromises based
   on the following:

   (1) The cost of maintaining 64-bit counters is relatively high,
        so minimizing the number of agents which must support them is
        desirable. Common interfaces (such as 10Mbs Ethernet) should
        not require them.

   (2) 64-bit counters are a new feature, introduced in SNMPv2. It
        is reasonable to expect that support for them will be spotty
        for the immediate future. Thus, we wish to limit them to as
        few systems as possible. This, in effect, means that 64-bit
        counters should be limited to higher speed interfaces.
        Ethernet (10,000,000 bps) and Token Ring (16,000,000 bps) are
        fairly wide-spread so it seems reasonable to not require 64-
        bit counters for these interfaces.

   (3) The 32-bit octet counters will wrap in the following times,
        for the following interfaces (when transmitting maximum-sized
        packets back-to-back):

        -    10Mbs Ethernet: 57 minutes,

        -    16Mbs Token Ring: 36 minutes,

              -    a US T3 line (45 megabits): 12 minutes,

              -    FDDI: 5.7 minutes

     (4)   The 32-bit packet counters wrap in about 57 minutes when 64-
           byte packets are transmitted back-to-back on a 650,000,000
           bit/second link.

     As an aside, a 1-terabit/second (1,000 Gbs) link will cause a 64 bit
     octet counter to wrap in just under 5 years. Conversely, an
     81,000,000 terabit/second link is required to cause a 64-bit counter
     to wrap in 30 minutes. We believe that, while technology rapidly
     marches forward, this link speed will not be achieved for at least
     several years, leaving sufficient time to evaluate the introduction
     of 96 bit counters.

     When 64-bit counters are in use, the 32-bit counters MUST still be
     available. They will report the low 32-bits of the associated 64-bit
     count (e.g., ifInOctets will report the least significant 32 bits of
     ifHCInOctets). This enhances inter-operability with existing
     implementations at a very minimal cost to agents.

     The new "high capacity" groups are:

     (1)   the ifHCFixedLengthGroup for character-oriented/fixed-length
           interfaces, and the ifHCPacketGroup for packet-based interfaces;
           both of these groups include 64 bit counters for octets, and

     (2)   the ifVHCPacketGroup for packet-based interfaces; this group
           includes 64 bit counters for octets and packets.

3.1.7. Interface Speed

     Network speeds are increasing. The range of ifSpeed is limited to
     reporting a maximum speed of $(2**31)-1$ bits/second, or approximately
     2.2Gbs. SONET defines an OC-48 interface, which is defined at
     operating at 48 times 51 Mbs, which is a speed in excess of 2.4Gbs.
     Thus, ifSpeed is insufficient for the future, and this memo defines
     an additional object: ifHighSpeed.

     The ifHighSpeed object reports the speed of the interface in
     1,000,000 (1 million) bits/second units. Thus, the true speed of the
     interface will be the value reported by this object, plus or minus
     500,000 bits/second.

Other alternatives considered (but rejected) were:

(1) Making the interface speed a 64-bit gauge. This was rejected
     since the current SMI does not allow such a syntax.

     Furthermore, even if 64-bit gauges were available, their use
     would require additional complexity in agents due to an
     increased requirement for 64-bit operations.

(2) We also considered making "high-32 bit" and "low-32-bit"
     objects which, when combined, would be a 64-bit value. This
     simply seemed overly complex for what we are trying to do.

     Furthermore, a full 64-bits of precision does not seem
     necessary. The value of ifHighSpeed will be the only report of
     interface speed for interfaces that are faster than
     4,294,967,295 bits per second. At this speed, the granularity
     of ifHighSpeed will be 1,000,000 bits per second, thus the error
     will be 1/4294, or about 0.02%. This seems reasonable.

(3) Adding a "scale" object, which would define the units which
     ifSpeed's value is.

     This would require two additional objects; one for the scaling
     object, and one to replace the current ifSpeed. This later
     object is required since the semantics of ifSpeed would be
     significantly altered, and manager stations which do not
     understand the new semantics would be confused.

3.1.8. Multicast/Broadcast Counters

   In MIB-II, the ifTable counters for multicast and broadcast packets
   are combined as counters of non-unicast packets. In contrast, the
   ifExtensions MIB [7] defined one set of counters for multicast, and a
   separate set for broadcast packets. With the separate counters, the
   original combined counters become redundant. To avoid this
   redundancy, the non-unicast counters are deprecated.

   For the output broadcast and multicast counters defined in RFC 1229,
   their definitions varied slightly from the packet counters in the
   ifTable, in that they did not count errors/discarded packets. Thus,
   this memo defines new objects with better aligned definitions.
   Counters with 64 bits of range are also needed, as explained above.

3.1.9. Trap Enable

   In the multi-layer interface model, each sub-layer for which there is
   an entry in the ifTable can generate linkUp/Down Traps. Since
   interface state changes would tend to propagate through the interface
   (from top to bottom, or bottom to top), it is likely that several
   traps would be generated for each linkUp/Down occurrence.

   It is desirable to provide a mechanism for manager stations to
   control the generation of these traps. To this end, the
   ifLinkUpDownTrapEnable object has been added. This object allows
   managers to limit generation of traps to just the sub-layers of
   interest.

   The default setting should limit the number of traps generated to one
   per interface per linkUp/Down event. Furthermore, it seems that the
   state changes of most interest to network managers occur at the
   lowest level of an interface stack. Therefore we specify that by
   default, only the lowest sub-layer of the interface generate traps.

3.1.10. Addition of New ifType values

   Over time, there is the need to add new ifType enumerated values for
   new interface types. If the syntax of ifType were defined in the MIB
   in section 6, then a new version of this MIB would have to be re-
   issued in order to define new values. In the past, re- issuing of a
   MIB has occurred only after several years.

   Therefore, the syntax of ifType is changed to be a textual
   convention, such that the enumerated integer values are now defined
   in the textual convention, IANAifType, defined in a different
   document. This allows additional values to be documented without
   having to re-issue a new version of this document. The Internet
   Assigned Number Authority (IANA) is responsible for the assignment of
   all Internet numbers, including various SNMP-related numbers, and
   specifically, new ifType values.

3.1.11. InterfaceIndex Textual Convention

   A new textual convention, InterfaceIndex, has been defined. This
   textual convention "contains" all of the semantics of the ifIndex
   object. This allows other mib modules to easily import the semantics
   of ifIndex.

3.1.12. New states for IfOperStatus

   Three new states have been added to ifOperStatus: 'dormant',
   'notPresent', and 'lowerLayerDown'.

   The dormant state indicates that the relevant interface is not
   actually in a condition to pass packets (i.e., it is not "up") but is
   in a "pending" state, waiting for some external event. For "on-
   demand" interfaces, this new state identifies the situation where the
   interface is waiting for events to place it in the up state.
   Examples of such events might be:

   (1) having packets to transmit before establishing a connection
        to a remote system;

   (2) having a remote system establish a connection to the
        interface (e.g. dialing up to a slip-server).

   The notPresent state is a refinement on the down state which
   indicates that the relevant interface is down specifically because
   some component (typically, a hardware component) is not present in
   the managed system. Examples of use of the notPresent state are:

   (1) to allow an interface's conceptual row including its counter
        values to be retained across a "hot swap" of a card/module,
        and/or

   (2) to allow an interface's conceptual row to be created, and
        thereby enable interfaces to be pre-configured prior to
        installation of the hardware needed to make the interface
        operational.

   Agents are not required to support interfaces in the notPresent
   state. However, from a conceptual viewpoint, when a row in the
   ifTable is created, it first enters the notPresent state and then
   subsequently transitions into the down state; similarly, when a row
   in the ifTable is deleted, it first enters the notPresent state and
   then subsequently the object instances are deleted. For an agent
   with no support for notPresent, both of these transitions (from the
   notPresent state to the down state, and from the notPresent state to
   the instances being removed) are immediate, i.e., the transition does
   not last long enough to be recorded by ifOperStatus. Even for those
   agents which do support interfaces in the notPresent state, the
   length of time and conditions under which an interface stays in the
   notPresent state is implementation-specific.

The lowerLayerDown state is also a refinement on the down state.
This new state indicates that this interface runs "on top of" one or
more other interfaces (see ifStackTable) and that this interface is
down specifically because one or more of these lower-layer interfaces
are down.

3.1.13. IfAdminStatus and IfOperStatus

The down state of ifOperStatus now has two meanings, depending on the
value of ifAdminStatus.

(1) if ifAdminStatus is not down and ifOperStatus is down then a
     fault condition is presumed to exist on the interface.

(2) if ifAdminStatus is down, then ifOperStatus will normally
     also be down (or notPresent) i.e., there is not (necessarily) a
     fault condition on the interface.

Note that when ifAdminStatus transitions to down, ifOperStatus will
normally also transition to down. In this situation, it is possible
that ifOperStatus's transition will not occur immediately, but rather
after a small time lag to complete certain operations before going
"down"; for example, it might need to finish transmitting a packet.
If a manager station finds that ifAdminStatus is down and
ifOperStatus is not down for a particular interface, the manager
station should wait a short while and check again. If the condition
still exists, only then should it raise an error indication.
Naturally, it should also ensure that ifLastChange has not changed
during this interval.

Whenever an interface table entry is created (usually as a result of
system initialization), the relevant instance of ifAdminStatus is set
to down, and presumably ifOperStatus will be down or notPresent.

An interface may be enabled in two ways: either as a result of
explicit management action (e.g. setting ifAdminStatus to up) or as a
result of the managed system's initialization process. When
ifAdminStatus changes to the up state, the related ifOperStatus
should do one of the following:

(1) Change to the up state if and only if the interface is able
     to send and receive packets.

(2) Change to the lowerLayerDown state if and only if the
     interface is prevented from entering the up state because of the
     state of one or more of the interfaces beneath it in the
     interface stack.

        (3) Change to the dormant state if and only if the interface is
             found to be operable, but the interface is waiting for other,
             external, events to occur before it can transmit or receive
             packets. Presumably when the expected events occur, the
             interface will then change to the up state.

        (4) Remain in the down state if an error or other fault condition
             is detected on the interface.

        (5) Change to the unknown state if, for some reason, the state of
             the interface can not be ascertained.

        (6) Change to the testing state if some test(s) must be performed
             on the interface. Presumably after completion of the test, the
             interface's state will change to up, dormant, or down, as
             appropriate.

        (7) Remain in the notPresent state if interface components are
             missing.

3.1.14. IfOperStatus in an Interface Stack

   When an interface is a part of an interface-stack, but is not the
   lowest interface in the stack, then:

   (1) ifOperStatus has the value 'up' if it is able to pass packets
        due to one or more interfaces below it in the stack being 'up',
        irrespective of whether other interfaces below it are 'down',
        'dormant', 'notPresent', 'lowerLayerDown', 'unknown' or
        'testing'.

   (2) ifOperStatus may have the value 'up' or 'dormant' if one or
        more interfaces below it in the stack are 'dormant', and all
        others below it are either 'down', 'dormant', 'notPresent',
        'lowerLayerDown', 'unknown' or 'testing'.

   (3) ifOperStatus has the value 'lowerLayerDown' while all
        interfaces below it in the stack are either 'down',
        'notPresent', 'lowerLayerDown', or 'testing'.

3.1.15. Traps

   The exact definition of when linkUp and linkDown traps are generated
   has been changed to reflect the changes to ifAdminStatus and
   ifOperStatus.

   Operational experience indicates that management stations are most
   concerned with an interface being in the down state and the fact that
   this state may indicate a failure. Thus, it is most useful to
   instrument transitions into/out of either the up state or the down
   state.

   Instrumenting transitions into or out of the up state was rejected
   since it would have the drawback that a demand interface might have
   many transitions between up and dormant, leading to many linkUp traps
   and no linkDown traps. Furthermore, if a node's only interface is
   the demand interface, then a transition to dormant would entail
   generation of a linkDown trap, necessitating bringing the link to the
   up state (and a linkUp trap)!!

   On the other hand, instrumenting transitions into or out of the down
   state (to/from all other states except notPresent) has the
   advantages:

   (1) A transition into the down state (from a state other than
        notPresent) will occur when an error is detected on an
        interface. Error conditions are presumably of great interest to
        network managers.

   (2) Departing the down state (to a state other than the
        notPresent state) generally indicates that the interface is
        going to either up or dormant, both of which are considered
        "healthy" states.

   Furthermore, it is believed that generating traps on transitions into
   or out of the down state (except to/from the notPresent state) is
   generally consistent with current usage and interpretation of these
   traps by manager stations.

   Transitions to/from the notPresent state are concerned with the
   insertion and removal of hardware, and are outside the scope of these
   traps.

   Therefore, this memo defines that LinkUp and linkDown traps are
   generated on just after ifOperStatus leaves, or just before it
   enters, the down state, respectively; except that LinkUp and linkDown
   traps never generated on transitions to/from the notPresent state.

   Note that this definition allows a node with only one interface to
   transmit a linkDown trap before that interface goes down. (Of
   course, when the interface is going down because of a failure
   condition, the linkDown trap probably cannot be successfully
   transmitted anyway.)

Some interfaces perform a link "training" function when trying to
bring the interface up. In the event that such an interface were
defective, then the training function would fail and the interface
would remain down, and the training function might be repeated at
appropriate intervals. If the interface, while performing this
training function, were considered to the in the testing state, then
linkUp and linkDown traps would be generated for each start and end
of the training function. This is not the intent of the linkUp and
linkDown traps, and therefore, while performing such a training
function, the interface's state should be represented as down.

An exception to the above generation of linkUp/linkDown traps on
changes in ifOperStatus, occurs when an interface is "flapping",
i.e., when it is rapidly oscillating between the up and down states.
If traps were generated for each such oscillation, the network and
the network management system would be flooded with unnecessary
traps. In such a situation, the agent should rate- limit its
generation of traps.

3.1.16. ifSpecific

The original definition of the OBJECT IDENTIFIER value of ifSpecific
was not sufficiently clear. As a result, different implementors used
it differently, and confusion resulted. Some implementations set the
value of ifSpecific to the OBJECT IDENTIFIER that defines the media-
specific MIB, i.e., the "foo" of:

                foo OBJECT IDENTIFIER ::= { transmission xxx }

while others set it to be OBJECT IDENTIFIER of the specific table or
entry in the appropriate media-specific MIB (i.e., fooTable or
fooEntry), while still others set it be the OBJECT IDENTIFIER of the
index object of the table's row, including instance identifier,
(i.e., fooIfIndex.ifIndex). A definition based on the latter would
not be sufficient unless it also allowed for media- specific MIBs
which include several tables, where each table has its own
(different) indexing.

The only definition that can both be made explicit and can cover all
the useful situations is to have ifSpecific be the most general value
for the media-specific MIB module (the first example given above).
This effectively makes it redundant because it contains no more
information than is provided by ifType. Thus, ifSpecific has been
deprecated.

3.1.17. Creation/Deletion of Interfaces

   While some interfaces, for example, most physical interfaces, cannot
   be created via network management, other interfaces such as logical
   interfaces sometimes can be. The ifTable contains only generic
   information about an interface. Almost all 'create-able' interfaces
   have other, media-specific, information through which configuration
   parameters may be supplied prior to creating such an interface.
   Thus, the ifTable does not itself support the creation or deletion of
   an interface (specifically, it has no RowStatus [2] column). Rather,
   if a particular interface type supports the dynamic creation and/or
   deletion of an interface of that type, then that media-specific MIB
   should include an appropriate RowStatus object (see the ATM LAN-
   Emulation Client MIB [8] for an example of a MIB which does this).
   Typically, when such a RowStatus object is created/deleted, then the
   conceptual row in the ifTable appears/disappears as a by-product, and
   an ifIndex value (chosen by the agent) is stored in an appropriate
   object in the media-specific MIB.

3.1.18. All Values Must be Known

   There are a number of situations where an agent does not know the
   value of one or more objects for a particular interface. In all such
   circumstances, an agent MUST NOT instantiate an object with an
   incorrect value; rather, it MUST respond with the appropriate
   error/exception condition (e.g., noSuchInstance for SNMPv2).

   One example is where an agent is unable to count the occurrences
   defined by one (or more) of the ifTable counters. In this
   circumstance, the agent MUST NOT instantiate the particular counter
   with a value of, say, zero. To do so would be to provide mis-
   information to a network management application reading the zero
   value, and thereby assuming that there have been no occurrences of
   the event (e.g., no input errors because ifInErrors is always zero).

   Sometimes the lack of knowledge of an object's value is temporary.
   For example, when the MTU of an interface is a configured value and a
   device dynamically learns the configured value through (after)
   exchanging messages over the interface (e.g., ATM LAN- Emulation
   [8]). In such a case, the value is not known until after the ifTable
   entry has already been created. In such a case, the ifTable entry
   should be created without an instance of the object whose value is
   unknown; later, when the value becomes known, the missing object can
   then be instantiated (e.g., the instance of ifMtu is only
   instantiated once the interface's MTU becomes known).

As a result of this "known values" rule, management applications MUST
be able to cope with the responses to retrieving the object instances
within a conceptual row of the ifTable revealing that some of the
row's columnar objects are missing/not available.

4. Media-Specific MIB Applicability

The exact use and semantics of many objects in this MIB are open to
some interpretation. This is a result of the generic nature of this
MIB. It is not always possible to come up with specific,
unambiguous, text that covers all cases and yet preserves the generic
nature of the MIB.

Therefore, it is incumbent upon a media-specific MIB designer to,
wherever necessary, clarify the use of the objects in this MIB with
respect to the media-specific MIB.

Specific areas of clarification include

Layering Model
     The media-specific MIB designer MUST completely and
     unambiguously specify the layering model used. Each individual
     sub-layer must be identified, as must the ifStackTable's
     portrayal of the relationship(s) between the sub-layers.

Virtual Circuits
     The media-specific MIB designer MUST specify whether virtual
     circuits are assigned entries in the ifTable or not. If they
     are, compelling rationale must be presented.

ifRcvAddressTable
     The media-specific MIB designer MUST specify the applicability
     of the ifRcvAddressTable.

ifType
     For each of the ifType values to which the media-specific MIB
     applies, it must specify the mapping of ifType values to media-
     specific MIB module(s) and instances of MIB objects within those
     modules.

However, wherever this interface MIB is specific in the semantics,
DESCRIPTION, or applicability of objects, the media-specific MIB
designer MUST NOT change said semantics, DESCRIPTION, or
applicability.

5. Overview

    This MIB consists of 4 tables:

    ifTable
         This table is the ifTable from MIB-II.

         ifXTable
         This table contains objects that have been added to the
         Interface MIB as a result of the Interface Evolution effort, or
         replacements for objects of the original (MIB-II) ifTable that
         were deprecated because the semantics of said objects have
         significantly changed. This table also contains objects that
         were previously in the ifExtnsTable.

    ifStackTable
         This table contains objects that define the relationships among
         the sub-layers of an interface.

    ifRcvAddressTable
         This table contains objects that are used to define the media-
         level addresses which this interface will receive. This table
         is a generic table. The designers of media- specific MIBs must
         define exactly how this table applies to their specific MIB.

6. Interfaces Group Definitions

    IF-MIB DEFINITIONS ::= BEGIN

    IMPORTS
         MODULE-IDENTITY, OBJECT-TYPE, Counter32, Gauge32, Counter64,
         Integer32, TimeTicks, mib-2,
         NOTIFICATION-TYPE   FROM SNMPv2-SMI
         TEXTUAL-CONVENTION, DisplayString,
         PhysAddress, TruthValue, RowStatus,
         TimeStamp, AutonomousType, TestAndIncr   FROM SNMPv2-TC
         MODULE-COMPLIANCE, OBJECT-GROUP   FROM SNMPv2-CONF
         snmpTraps   FROM SNMPv2-MIB
         IANAifType   FROM IANAifType-MIB;

```
     ifMIB MODULE-IDENTITY
         LAST-UPDATED "9611031355Z"
         ORGANIZATION "IETF Interfaces MIB Working Group"
         CONTACT-INFO
                 "    Keith McCloghrie
                     Cisco Systems, Inc.
                     170 West Tasman Drive
                     San Jose, CA  95134-1706
                     US

                     408-526-5260
                     kzm@cisco.com"
         DESCRIPTION
                 "The MIB module to describe generic objects for
                 network interface sub-layers.  This MIB is an updated
                 version of MIB-II's ifTable, and incorporates the
                 extensions defined in RFC 1229."
         REVISION       "9602282155Z"
         DESCRIPTION
                 "Revisions made by the Interfaces MIB WG."
         REVISION       "9311082155Z"
         DESCRIPTION
                 "Initial revision, published as part of RFC 1573."
         ::= { mib-2 31 }


     ifMIBObjects OBJECT IDENTIFIER ::= { ifMIB 1 }

     interfaces   OBJECT IDENTIFIER ::= { mib-2 2 }


     OwnerString ::= TEXTUAL-CONVENTION
         DISPLAY-HINT "255a"
         STATUS        current
         DESCRIPTION
                 "This data type is used to model an administratively
                 assigned name of the owner of a resource.  This
                 information is taken from the NVT ASCII character set.
                 It is suggested that this name contain one or more of
                 the following: ASCII form of the manager station's
                 transport address, management station name (e.g.,
                 domain name), network management personnel's name,
                 location, or phone number.  In some cases the agent
                 itself will be the owner of an entry.  In these cases,
                 this string shall be set to a string starting with
                 'agent'."
         SYNTAX       OCTET STRING (SIZE(0..255))
```

```
     -- InterfaceIndex contains the semantics of ifIndex and
     -- should be used for any objects defined on other mib
     -- modules that need these semantics.

     InterfaceIndex ::= TEXTUAL-CONVENTION
         DISPLAY-HINT "d"
         STATUS        current
         DESCRIPTION
                 "A unique value, greater than zero, for each interface
                 or interface sub-layer in the managed system.  It is
                 recommended that values are assigned contiguously
                 starting from 1.  The value for each interface sub-
                 layer must remain constant at least from one re-
                 initialization of the entity's network management
                 system to the next re-initialization."
         SYNTAX        Integer32 (1..2147483647)


     InterfaceIndexOrZero ::= TEXTUAL-CONVENTION
         DISPLAY-HINT "d"
         STATUS        current
         DESCRIPTION
                 "This textual convention is an extension of the
                 InterfaceIndex convention.  The latter defines a
                 greater than zero value used to identify an interface
                 or interface sub-layer in the managed system.  This
                 extension permits the additional value of zero.  the
                 value zero is object-specific and must therefore be
                 defined as part of the description of any object which
                 uses this syntax.  Examples of the usage of zero might
                 include situations where interface was unknown, or
                 when none or all interfaces need to be referenced."
         SYNTAX        Integer32 (0..2147483647)


     ifNumber  OBJECT-TYPE
         SYNTAX        Integer32
         MAX-ACCESS    read-only
         STATUS        current
         DESCRIPTION
                 "The number of network interfaces (regardless of their
                 current state) present on this system."
         ::= { interfaces 1 }
```

```
   ifTableLastChange  OBJECT-TYPE
       SYNTAX        TimeTicks
       MAX-ACCESS  read-only
       STATUS        current
       DESCRIPTION
               "The value of sysUpTime at the time of the last
               creation or deletion of an entry in the ifTable.  If
               the number of entries has been unchanged since the
               last re-initialization of the local network management
               subsystem, then this object contains a zero value."
       ::= { ifMIBObjects 5 }


   -- the Interfaces table

   -- The Interfaces table contains information on the entity's
   -- interfaces.  Each sub-layer below the internetwork-layer
   -- of a network interface is considered to be an interface.

   ifTable OBJECT-TYPE
       SYNTAX        SEQUENCE OF IfEntry
       MAX-ACCESS  not-accessible
       STATUS        current
       DESCRIPTION
               "A list of interface entries.  The number of entries
               is given by the value of ifNumber."
       ::= { interfaces 2 }

   ifEntry OBJECT-TYPE
       SYNTAX        IfEntry
       MAX-ACCESS  not-accessible
       STATUS        current
       DESCRIPTION
               "An entry containing management information applicable
               to a particular interface."
       INDEX    { ifIndex }



       ::= { ifTable 1 }

   IfEntry ::=
       SEQUENCE {
           ifIndex                 InterfaceIndex,
           ifDescr                 DisplayString,
           ifType                  IANAifType,
           ifMtu                   Integer32,
           ifSpeed                 Gauge32,
```

```
        ifPhysAddress            PhysAddress,
        ifAdminStatus            INTEGER,
        ifOperStatus             INTEGER,
        ifLastChange             TimeTicks,
        ifInOctets               Counter32,
        ifInUcastPkts            Counter32,
        ifInNUcastPkts           Counter32,   -- deprecated
        ifInDiscards             Counter32,
        ifInErrors               Counter32,
        ifInUnknownProtos        Counter32,
        ifOutOctets              Counter32,
        ifOutUcastPkts           Counter32,
        ifOutNUcastPkts          Counter32,   -- deprecated
        ifOutDiscards            Counter32,
        ifOutErrors              Counter32,
        ifOutQLen                Gauge32,     -- deprecated
        ifSpecific               OBJECT IDENTIFIER -- deprecated
    }


ifIndex OBJECT-TYPE
    SYNTAX      InterfaceIndex
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION
            "A unique value, greater than zero, for each
            interface.  It is recommended that values are assigned
            contiguously starting from 1.  The value for each
            interface sub-layer must remain constant at least from
            one re-initialization of the entity's network
            management system to the next re-initialization."
    ::= { ifEntry 1 }

ifDescr OBJECT-TYPE
    SYNTAX      DisplayString (SIZE (0..255))
    MAX-ACCESS  read-only


    STATUS      current
    DESCRIPTION
            "A textual string containing information about the
            interface.  This string should include the name of the
            manufacturer, the product name and the version of the
            interface hardware/software."
    ::= { ifEntry 2 }
```

```
    ifType OBJECT-TYPE
        SYNTAX        IANAifType
        MAX-ACCESS  read-only
        STATUS        current
        DESCRIPTION
                "The type of interface.  Additional values for ifType
                are assigned by the Internet Assigned Numbers
                Authority (IANA), through updating the syntax of the
                IANAifType textual convention."
        ::= { ifEntry 3 }

    ifMtu OBJECT-TYPE
        SYNTAX        Integer32
        MAX-ACCESS  read-only
        STATUS        current
        DESCRIPTION
                "The size of the largest packet which can be
                sent/received on the interface, specified in octets.
                For interfaces that are used for transmitting network
                datagrams, this is the size of the largest network
                datagram that can be sent on the interface."
        ::= { ifEntry 4 }

    ifSpeed OBJECT-TYPE
        SYNTAX        Gauge32
        MAX-ACCESS  read-only
        STATUS        current
        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 object should contain the nominal
                bandwidth.  If the bandwidth of the interface is
                greater than the maximum value reportable by this
                object then this object should report its maximum
                value (4,294,967,295) and ifHighSpeed must be used to
                report the interace's speed.  For a sub-layer which
                has no concept of bandwidth, this object should be
                zero."
        ::= { ifEntry 5 }

    ifPhysAddress OBJECT-TYPE
        SYNTAX        PhysAddress
        MAX-ACCESS  read-only
        STATUS        current
```

         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 MIB must define the bit and byte
                 ordering and the format of the value of this object.
                 For interfaces which do not have such an address
                 (e.g., a serial line), this object should contain an
                 octet string of zero length."
         ::= { ifEntry 6 }

    ifAdminStatus OBJECT-TYPE
        SYNTAX INTEGER {
                 up(1),   -- ready to pass packets
                 down(2),
                 testing(3)   -- in some test mode
             }
        MAX-ACCESS read-write
        STATUS   current
        DESCRIPTION
                 "The desired state of the interface. The testing(3)
                 state indicates that no operational packets can be
                 passed. When a managed system initializes, all
                 interfaces start with ifAdminStatus in the down(2)
                 state. As a result of either explicit management
                 action or per configuration information retained by
                 the managed system, ifAdminStatus is then changed to
                 either the up(1) or testing(3) states (or remains in
                 the down(2) state)."
         ::= { ifEntry 7 }

    ifOperStatus OBJECT-TYPE
        SYNTAX INTEGER {
                 up(1),   -- ready to pass packets
                 down(2),
                 testing(3),   -- in some test mode
                 unknown(4),   -- status can not be determined
                                 -- for some reason.
                 dormant(5),
                 notPresent(6),   -- some component is missing
                 lowerLayerDown(7) -- down due to state of
                                   -- lower-layer interface(s)
             }

```
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
                "The current operational state of the interface.  The
                testing(3) state indicates that no operational packets
                can be passed.  If ifAdminStatus is down(2) then
                ifOperStatus should be down(2).  If ifAdminStatus is
                changed to up(1) then ifOperStatus should change to
                up(1) if the interface is ready to transmit and
                receive network traffic; it should change to
                dormant(5) if the interface is waiting for external
                actions (such as a serial line waiting for an incoming
                connection); it should remain in the down(2) state if
                and only if there is a fault that prevents it from
                going to the up(1) state; it should remain in the
                notPresent(6) state if the interface has missing
                (typically, hardware) components."
        ::= { ifEntry 8 }

    ifLastChange OBJECT-TYPE
        SYNTAX      TimeTicks
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
                "The value of sysUpTime at 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
                object contains a zero value."
        ::= { ifEntry 9 }

    ifInOctets OBJECT-TYPE
        SYNTAX      Counter32
        MAX-ACCESS  read-only
        STATUS      current
        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
                ifCounterDiscontinuityTime."
        ::= { ifEntry 10 }
```

    ifInUcastPkts OBJECT-TYPE
        SYNTAX   Counter32
        MAX-ACCESS read-only
        STATUS   current
        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
                ifCounterDiscontinuityTime."
        ::= { ifEntry 11 }

    ifInNUcastPkts OBJECT-TYPE
        SYNTAX Counter32
        MAX-ACCESS read-only
        STATUS   deprecated
        DESCRIPTION
                "The number of packets, delivered by this sub-layer to
                a higher (sub-)layer, which were 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
                ifCounterDiscontinuityTime.

                This object is deprecated in favour of
                ifInMulticastPkts and ifInBroadcastPkts."
        ::= { ifEntry 12 }

    ifInDiscards OBJECT-TYPE
        SYNTAX   Counter32
        MAX-ACCESS read-only
        STATUS   current
        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
                ifCounterDiscontinuityTime."

```
     ::= { ifEntry 13 }

ifInErrors OBJECT-TYPE
    SYNTAX        Counter32
    MAX-ACCESS  read-only
    STATUS        current
    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
            ifCounterDiscontinuityTime."
    ::= { ifEntry 14 }

ifInUnknownProtos OBJECT-TYPE
    SYNTAX        Counter32
    MAX-ACCESS  read-only
    STATUS        current
    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 will always be 0.

            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
            ifCounterDiscontinuityTime."
    ::= { ifEntry 15 }
```

    ifOutOctets OBJECT-TYPE
        SYNTAX   Counter32
        MAX-ACCESS   read-only
        STATUS   current
        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
                ifCounterDiscontinuityTime."
        ::= { ifEntry 16 }

    ifOutUcastPkts OBJECT-TYPE
        SYNTAX   Counter32
        MAX-ACCESS   read-only
        STATUS   current
        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
                ifCounterDiscontinuityTime."
        ::= { ifEntry 17 }

    ifOutNUcastPkts OBJECT-TYPE
        SYNTAX   Counter32
        MAX-ACCESS   read-only
        STATUS   deprecated
        DESCRIPTION
                "The total number of packets that higher-level
                protocols requested be transmitted, and which were
                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
                ifCounterDiscontinuityTime.

                   This object is deprecated in favour of
                   ifOutMulticastPkts and ifOutBroadcastPkts."
           ::= { ifEntry 18 }

       ifOutDiscards OBJECT-TYPE
           SYNTAX   Counter32
           MAX-ACCESS read-only
           STATUS   current
           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
                   ifCounterDiscontinuityTime."
           ::= { ifEntry 19 }

       ifOutErrors OBJECT-TYPE
           SYNTAX   Counter32
           MAX-ACCESS read-only
           STATUS   current
           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
                   ifCounterDiscontinuityTime."
           ::= { ifEntry 20 }

       ifOutQLen OBJECT-TYPE
           SYNTAX   Gauge32
           MAX-ACCESS read-only
           STATUS   deprecated
           DESCRIPTION
                   "The length of the output packet queue (in packets)."
           ::= { ifEntry 21 }

```
    ifSpecific OBJECT-TYPE
        SYNTAX       OBJECT IDENTIFIER
        MAX-ACCESS   read-only
        STATUS       deprecated
        DESCRIPTION
                "A reference to MIB definitions specific to the
                particular media being used to realize the interface.
                It is recommended that this value point to an instance
                of a MIB object in the media-specific MIB, i.e., that
                this object have the semantics associated with the
                InstancePointer textual convention defined in RFC
                1903.  In fact, it is recommended that the media-
                specific MIB specify what value ifSpecific should/can
                take for values of ifType.  If no MIB definitions
                specific to the particular media are available, the
                value should be set to the OBJECT IDENTIFIER { 0 0 }."
        ::= { ifEntry 22 }



    --
    --    Extension to the interface table
    --
    -- This table replaces the ifExtnsTable table.
    --

    ifXTable        OBJECT-TYPE
        SYNTAX       SEQUENCE OF IfXEntry
        MAX-ACCESS   not-accessible
        STATUS       current
        DESCRIPTION
                "A list of interface entries.  The number of entries
                is given by the value of ifNumber.  This table
                contains additional objects for the interface table."
        ::= { ifMIBObjects 1 }

    ifXEntry        OBJECT-TYPE
        SYNTAX       IfXEntry
        MAX-ACCESS   not-accessible
        STATUS       current
        DESCRIPTION
                "An entry containing additional management information
                applicable to a particular interface."
        AUGMENTS     { ifEntry }
        ::= { ifXTable 1 }
```

```
    IfXEntry ::=
        SEQUENCE {
            ifName                  DisplayString,
            ifInMulticastPkts       Counter32,
            ifInBroadcastPkts       Counter32,
            ifOutMulticastPkts      Counter32,
            ifOutBroadcastPkts      Counter32,
            ifHCInOctets            Counter64,
            ifHCInUcastPkts         Counter64,
            ifHCInMulticastPkts     Counter64,
            ifHCInBroadcastPkts     Counter64,
            ifHCOutOctets           Counter64,
            ifHCOutUcastPkts        Counter64,
            ifHCOutMulticastPkts    Counter64,
            ifHCOutBroadcastPkts    Counter64,
            ifLinkUpDownTrapEnable  INTEGER,
            ifHighSpeed             Gauge32,
            ifPromiscuousMode       TruthValue,
            ifConnectorPresent      TruthValue,
            ifAlias                 DisplayString,
            ifCounterDiscontinuityTime TimeStamp
        }


    ifName OBJECT-TYPE
        SYNTAX      DisplayString
        MAX-ACCESS  read-only
        STATUS      current
        DESCRIPTION
                "The textual name of the interface.  The value of this
                object should be the name of the interface as assigned
                by the local device and should be suitable for use in
                commands entered at the device's 'console'.  This
                might be a text name, such as 'le0' or a simple port
                number, such as '1', depending on the interface naming
                syntax of the device.  If several entries in the
                ifTable together represent a single interface as named
                by the device, then each will have the same value of
                ifName.  Note that for an agent which responds to SNMP
                queries concerning an interface on some other
                (proxied) device, then the value of ifName for such an
                interface is the proxied device's local name for it.

                If there is no local name, or this object is otherwise
                not applicable, then this object contains a zero-
                length string."
        ::= { ifXEntry 1 }
```

```
    ifInMulticastPkts OBJECT-TYPE
        SYNTAX       Counter32
        MAX-ACCESS  read-only
        STATUS       current
        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
                ifCounterDiscontinuityTime."
        ::= { ifXEntry 2 }

    ifInBroadcastPkts OBJECT-TYPE
        SYNTAX       Counter32
        MAX-ACCESS  read-only
        STATUS       current
        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
                ifCounterDiscontinuityTime."
        ::= { ifXEntry 3 }

    ifOutMulticastPkts OBJECT-TYPE
        SYNTAX       Counter32
        MAX-ACCESS  read-only
        STATUS       current
        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
                ifCounterDiscontinuityTime."
```

```
        ::= { ifXEntry 4 }

   ifOutBroadcastPkts OBJECT-TYPE
       SYNTAX       Counter32
       MAX-ACCESS   read-only
       STATUS       current
       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
               ifCounterDiscontinuityTime."
       ::= { ifXEntry 5 }

   --
   -- High Capacity Counter objects.  These objects are all
   -- 64 bit versions of the "basic" ifTable counters.  These
   -- objects all have the same basic semantics as their 32-bit
   -- counterparts, however, their syntax has been extended
   -- to 64 bits.
   --

   ifHCInOctets OBJECT-TYPE
       SYNTAX       Counter64
       MAX-ACCESS   read-only
       STATUS       current

       DESCRIPTION
               "The total number of octets received on the interface,
               including framing characters.  This object is a 64-bit
               version of ifInOctets.

               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
               ifCounterDiscontinuityTime."
       ::= { ifXEntry 6 }
```

```
    ifHCInUcastPkts OBJECT-TYPE
        SYNTAX       Counter64
        MAX-ACCESS   read-only
        STATUS       current
        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.
                This object is a 64-bit version of ifInUcastPkts.

                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
                ifCounterDiscontinuityTime."
        ::= { ifXEntry 7 }

    ifHCInMulticastPkts OBJECT-TYPE
        SYNTAX       Counter64
        MAX-ACCESS   read-only
        STATUS       current
        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.  This object is a 64-bit version of
                ifInMulticastPkts.

                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
                ifCounterDiscontinuityTime."
        ::= { ifXEntry 8 }


    ifHCInBroadcastPkts OBJECT-TYPE
        SYNTAX       Counter64
        MAX-ACCESS   read-only
        STATUS       current
        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.  This object is a
                64-bit version of ifInBroadcastPkts.

                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
```

```
                          ifCounterDiscontinuityTime."
                 ::= { ifXEntry 9 }

         ifHCOutOctets OBJECT-TYPE
             SYNTAX        Counter64
             MAX-ACCESS  read-only
             STATUS        current
             DESCRIPTION
                     "The total number of octets transmitted out of the
                     interface, including framing characters.  This object
                     is a 64-bit version of ifOutOctets.

                     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
                     ifCounterDiscontinuityTime."
                 ::= { ifXEntry 10 }

         ifHCOutUcastPkts OBJECT-TYPE
             SYNTAX        Counter64
             MAX-ACCESS  read-only
             STATUS        current
             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.  This object is a 64-bit version of
                     ifOutUcastPkts.

                     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
                     ifCounterDiscontinuityTime."
                 ::= { ifXEntry 11 }

         ifHCOutMulticastPkts OBJECT-TYPE
             SYNTAX        Counter64
             MAX-ACCESS  read-only
             STATUS        current
             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.  This object is a 64-bit version
                     of ifOutMulticastPkts.
```

                    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
                    ifCounterDiscontinuityTime."
            ::= { ifXEntry 12 }

    ifHCOutBroadcastPkts OBJECT-TYPE
        SYNTAX   Counter64
        MAX-ACCESS   read-only
        STATUS   current
        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. This
                    object is a 64-bit version of ifOutBroadcastPkts.

                    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
                    ifCounterDiscontinuityTime."
            ::= { ifXEntry 13 }

    ifLinkUpDownTrapEnable   OBJECT-TYPE
        SYNTAX   INTEGER { enabled(1), disabled(2) }
        MAX-ACCESS   read-write
        STATUS   current
        DESCRIPTION

                    "Indicates whether linkUp/linkDown traps should be
                    generated for this interface.

                    By default, this object should have the value
                    enabled(1) for interfaces which do not operate on
                    'top' of any other interface (as defined in the
                    ifStackTable), and disabled(2) otherwise."
            ::= { ifXEntry 14 }

    ifHighSpeed OBJECT-TYPE
        SYNTAX   Gauge32
        MAX-ACCESS   read-only
        STATUS   current
        DESCRIPTION
                    "An estimate of the interface's current bandwidth in
                    units of 1,000,000 bits per second. If this object
                    reports a value of 'n' then the speed of the interface
                    is somewhere in the range of 'n-500,000' to
                    'n+499,999'. For interfaces which do not vary in

                   bandwidth or for those where no accurate estimation
                   can be made, this object should contain the nominal
                   bandwidth. For a sub-layer which has no concept of
                   bandwidth, this object should be zero."
           ::= { ifXEntry 15 }

       ifPromiscuousMode OBJECT-TYPE
           SYNTAX   TruthValue
           MAX-ACCESS read-write
           STATUS   current
           DESCRIPTION
                   "This object has a value of false(2) if this interface
                   only accepts packets/frames that are addressed to this
                   station. This object has a value of true(1) when the
                   station accepts all packets/frames transmitted on the
                   media. The value true(1) is only legal on certain
                   types of media. If legal, setting this object to a
                   value of true(1) may require the interface to be reset
                   before becoming effective.

                   The value of ifPromiscuousMode does not affect the
                   reception of broadcast and multicast packets/frames by
                   the interface."
           ::= { ifXEntry 16 }

       ifConnectorPresent   OBJECT-TYPE
           SYNTAX   TruthValue
           MAX-ACCESS read-only
           STATUS   current
           DESCRIPTION
                   "This object has the value 'true(1)' if the interface
                   sublayer has a physical connector and the value
                   'false(2)' otherwise."
           ::= { ifXEntry 17 }

       ifAlias   OBJECT-TYPE
           SYNTAX   DisplayString (SIZE(0..64))
           MAX-ACCESS read-write
           STATUS   current
           DESCRIPTION
                   "This object is an 'alias' name for the interface as
                   specified by a network manager, and provides a non-
                   volatile 'handle' for the interface.

                   On the first instantiation of an interface, the value
                   of ifAlias associated with that interface is the
                   zero-length string. As and when a value is written
                   into an instance of ifAlias through a network

                    management set operation, then the agent must retain
                    the supplied value in the ifAlias instance associated
                    with the same interface for as long as that interface
                    remains instantiated, including across all re-
                    initializations/reboots of the network management
                    system, including those which result in a change of
                    the interface's ifIndex value.

                    An example of the value which a network manager might
                    store in this object for a WAN interface is the
                    (Telco's) circuit number/identifier of the interface.

                    Some agents may support write-access only for
                    interfaces having particular values of ifType. An
                    agent which supports write access to this object is
                    required to keep the value in non-volatile storage,
                    but it may limit the length of new values depending on
                    how much storage is already occupied by the current
                    values for other interfaces."
            ::= { ifXEntry 18 }

    ifCounterDiscontinuityTime OBJECT-TYPE
        SYNTAX   TimeStamp
        MAX-ACCESS read-only
        STATUS   current
        DESCRIPTION
                    "The value of sysUpTime on the most recent occasion at
                    which any one or more of this interface's counters
                    suffered a discontinuity. The relevant counters are
                    the specific instances associated with this interface
                    of any Counter32 or Counter64 object contained in the
                    ifTable or ifXTable. If no such discontinuities have
                    occurred since the last re-initialization of the local
                    management subsystem, then this object contains a zero
                    value."
            ::= { ifXEntry 19 }

    --   The Interface Stack Group
    --
    -- Implementation of this group is mandatory for all systems
    --

    ifStackTable OBJECT-TYPE
        SYNTAX   SEQUENCE OF IfStackEntry
        MAX-ACCESS   not-accessible
        STATUS   current
        DESCRIPTION

            "The table containing information on the relationships
            between the multiple sub-layers of network interfaces.
            In particular, it contains information on which sub-
            layers run 'on top of' which other sub-layers, where
            each sub-layer corresponds to a conceptual row in the
            ifTable. For example, when the sub-layer with ifIndex
            value x runs over the sub-layer with ifIndex value y,
            then this table contains:

              ifStackStatus.x.y=active

            For each ifIndex value, I, which identifies an active
            interface, there are always at least two instantiated
            rows in this table associated with I. For one of
            these rows, I is the value of ifStackHigherLayer; for
            the other, I is the value of ifStackLowerLayer. (If I
            is not involved in multiplexing, then these are the
            only two rows associated with I.)

            For example, two rows exist even for an interface
            which has no others stacked on top or below it:

              ifStackStatus.0.x=active
              ifStackStatus.x.0=active "
        ::= { ifMIBObjects 2 }

    ifStackEntry OBJECT-TYPE
        SYNTAX   IfStackEntry
        MAX-ACCESS   not-accessible
        STATUS   current
        DESCRIPTION
              "Information on a particular relationship between two
              sub-layers, specifying that one sub-layer runs on
              'top' of the other sub-layer. Each sub-layer
              corresponds to a conceptual row in the ifTable."
        INDEX { ifStackHigherLayer, ifStackLowerLayer }
        ::= { ifStackTable 1 }

    IfStackEntry ::=
        SEQUENCE {
            ifStackHigherLayer Integer32,
            ifStackLowerLayer   Integer32,
            ifStackStatus   RowStatus
        }

    ifStackHigherLayer OBJECT-TYPE
        SYNTAX   Integer32
        MAX-ACCESS   not-accessible
        STATUS   current
        DESCRIPTION
                "The value of ifIndex corresponding to the higher
                sub-layer of the relationship, i.e., the sub-layer
                which runs on 'top' of the sub-layer identified by the
                corresponding instance of ifStackLowerLayer. If there
                is no higher sub-layer (below the internetwork layer),
                then this object has the value 0."
        ::= { ifStackEntry 1 }

    ifStackLowerLayer OBJECT-TYPE
        SYNTAX   Integer32
        MAX-ACCESS   not-accessible
        STATUS   current
        DESCRIPTION
                "The value of ifIndex corresponding to the lower sub-
                layer of the relationship, i.e., the sub-layer which
                runs 'below' the sub-layer identified by the
                corresponding instance of ifStackHigherLayer. If
                there is no lower sub-layer, then this object has the
                value 0."
        ::= { ifStackEntry 2 }

    ifStackStatus   OBJECT-TYPE
        SYNTAX   RowStatus
        MAX-ACCESS   read-create
        STATUS   current
        DESCRIPTION
                "The status of the relationship between two sub-
                layers.

                Changing the value of this object from 'active' to
                'notInService' or 'destroy' will likely have
                consequences up and down the interface stack. Thus,
                write access to this object is likely to be
                inappropriate for some types of interfaces, and many
                implementations will choose not to support write-
                access for any type of interface."
        ::= { ifStackEntry 3 }

    ifStackLastChange OBJECT-TYPE
        SYNTAX   TimeTicks
        MAX-ACCESS   read-only

        STATUS   current
        DESCRIPTION
                "The value of sysUpTime at the time of the last change
                of the (whole) interface stack. A change of the
                interface stack is defined to be any creation,
                deletion, or change in value of any instance of
                ifStackStatus. If the interface stack has been
                unchanged since the last re-initialization of the
                local network management subsystem, then this object
                contains a zero value."
        ::= { ifMIBObjects 6 }

    --   Generic Receive Address Table
    --
    -- This group of objects is mandatory for all types of
    -- interfaces which can receive packets/frames addressed to
    -- more than one address.
    --
    -- This table replaces the ifExtnsRcvAddr table. The main
    -- difference is that this table makes use of the RowStatus
    -- textual convention, while ifExtnsRcvAddr did not.

    ifRcvAddressTable OBJECT-TYPE
        SYNTAX   SEQUENCE OF IfRcvAddressEntry
        MAX-ACCESS not-accessible
        STATUS   current
        DESCRIPTION
                "This table contains an entry for each address
                (broadcast, multicast, or uni-cast) for which the
                system will receive packets/frames on a particular
                interface, except as follows:

                - for an interface operating in promiscuous mode,
                entries are only required for those addresses for
                which the system would receive frames were it not
                operating in promiscuous mode.

                - for 802.5 functional addresses, only one entry is
                required, for the address which has the functional
                address bit ANDed with the bit mask of all functional
                addresses for which the interface will accept frames.

                A system is normally able to use any unicast address
                which corresponds to an entry in this table as a
                source address."
        ::= { ifMIBObjects 4 }

```
    ifRcvAddressEntry  OBJECT-TYPE
        SYNTAX        IfRcvAddressEntry
        MAX-ACCESS  not-accessible
        STATUS        current
        DESCRIPTION
                "A list of objects identifying an address for which
                the system will accept packets/frames on the
                particular interface identified by the index value
                ifIndex."
        INDEX  { ifIndex, ifRcvAddressAddress }
        ::= { ifRcvAddressTable 1 }

    IfRcvAddressEntry ::=
        SEQUENCE {
            ifRcvAddressAddress    PhysAddress,
            ifRcvAddressStatus     RowStatus,
            ifRcvAddressType       INTEGER
        }

    ifRcvAddressAddress OBJECT-TYPE
        SYNTAX        PhysAddress
        MAX-ACCESS  not-accessible
        STATUS        current
        DESCRIPTION
                "An address for which the system will accept
                packets/frames on this entry's interface."

        ::= { ifRcvAddressEntry 1 }

    ifRcvAddressStatus OBJECT-TYPE
        SYNTAX        RowStatus
        MAX-ACCESS  read-create
        STATUS        current
        DESCRIPTION
                "This object is used to create and delete rows in the
                ifRcvAddressTable."

        ::= { ifRcvAddressEntry 2 }

    ifRcvAddressType OBJECT-TYPE
        SYNTAX         INTEGER {
                        other(1),
                        volatile(2),
                        nonVolatile(3)
                    }

        MAX-ACCESS  read-create
        STATUS        current
```

          DESCRIPTION
                 "This object has the value nonVolatile(3) for those
                 entries in the table which are valid and will not be
                 deleted by the next restart of the managed system.
                 Entries having the value volatile(2) are valid and
                 exist, but have not been saved, so that will not exist
                 after the next restart of the managed system. Entries
                 having the value other(1) are valid and exist but are
                 not classified as to whether they will continue to
                 exist after the next restart."

          DEFVAL   { volatile }
          ::= { ifRcvAddressEntry 3 }

     -- definition of interface-related traps.

     linkDown NOTIFICATION-TYPE
             OBJECTS { ifIndex, ifAdminStatus, ifOperStatus }
             STATUS current
             DESCRIPTION
                 "A linkDown trap signifies that the SNMPv2 entity,
                 acting in an agent role, has detected that the
                 ifOperStatus object for one of its communication links
                 is about to enter the down state from some other state
                 (but not from the notPresent state). This other state
                 is indicated by the included value of ifOperStatus."
          ::= { snmpTraps 3 }

     linkUp NOTIFICATION-TYPE
             OBJECTS { ifIndex, ifAdminStatus, ifOperStatus }
             STATUS current
             DESCRIPTION
                 "A linkDown trap signifies that the SNMPv2 entity,
                 acting in an agent role, has detected that the
                 ifOperStatus object for one of its communication links
                 left the down state and transitioned into some other
                 state (but not into the notPresent state). This other
                 state is indicated by the included value of
                 ifOperStatus."
          ::= { snmpTraps 4 }

     -- conformance information

     ifConformance OBJECT IDENTIFIER ::= { ifMIB 2 }

     ifGroups   OBJECT IDENTIFIER ::= { ifConformance 1 }
     ifCompliances OBJECT IDENTIFIER ::= { ifConformance 2 }

        -- compliance statements

     ifCompliance2 MODULE-COMPLIANCE
         STATUS current
         DESCRIPTION
                 "The compliance statement for SNMPv2 entities which
                 have network interfaces."

         MODULE -- this module
             MANDATORY-GROUPS { ifGeneralInformationGroup, ifStackGroup2,
                                ifCounterDiscontinuityGroup }

             GROUP   ifFixedLengthGroup
             DESCRIPTION
                 "This group is mandatory for all network interfaces
                 which are character-oriented or transmit data in
                 fixed-length transmission units."

             GROUP   ifHCFixedLengthGroup
             DESCRIPTION
                 "This group is mandatory only for those network
                 interfaces which are character-oriented or transmit
                 data in fixed-length transmission units, and for which
                 the value of the corresponding instance of ifSpeed is
                 greater than 20,000,000 bits/second."

             GROUP   ifPacketGroup
             DESCRIPTION
                 "This group is mandatory for all network interfaces
                 which are packet-oriented."

             GROUP   ifHCPacketGroup
             DESCRIPTION
                 "This group is mandatory only for those network
                 interfaces which are packet-oriented and for which the
                 value of the corresponding instance of ifSpeed is
                 greater than 650,000,000 bits/second."

             GROUP   ifRcvAddressGroup
             DESCRIPTION
                 "The applicability of this group MUST be defined by
                 the media-specific MIBs. Media-specific MIBs must
                 define the exact meaning, use, and semantics of the
                 addresses in this group."

```
            OBJECT       ifLinkUpDownTrapEnable
            MIN-ACCESS   read-only
            DESCRIPTION
                "Write access is not required."

            OBJECT       ifPromiscuousMode
            MIN-ACCESS   read-only
            DESCRIPTION
                "Write access is not required."

            OBJECT       ifStackStatus
            SYNTAX       INTEGER { active(1) } -- subset of RowStatus
            MIN-ACCESS   read-only
            DESCRIPTION
                "Write access is not required, and only one of the six
                enumerated values for the RowStatus textual convention
                need be supported, specifically: active(1)."

            OBJECT       ifAdminStatus
            SYNTAX       INTEGER { up(1), down(2) }
            MIN-ACCESS   read-only
            DESCRIPTION
                "Write access is not required, nor is support for the
                value testing(3)."

            OBJECT       ifAlias
            MIN-ACCESS   read-only
            DESCRIPTION
                "Write access is not required."

        ::= { ifCompliances 2 }

    -- units of conformance

    ifGeneralInformationGroup    OBJECT-GROUP
        OBJECTS { ifIndex, ifDescr, ifType, ifSpeed, ifPhysAddress,
                  ifAdminStatus, ifOperStatus, ifLastChange,
                  ifLinkUpDownTrapEnable, ifConnectorPresent,
                  ifHighSpeed, ifName, ifNumber, ifAlias,
                  ifTableLastChange }
        STATUS  current
        DESCRIPTION
                "A collection of objects providing information
                applicable to all network interfaces."
        ::= { ifGroups 10 }

    -- the following five groups are mutually exclusive; at most
    -- one of these groups is implemented for any interface
```

```
    ifFixedLengthGroup    OBJECT-GROUP
        OBJECTS { ifInOctets, ifOutOctets, ifInUnknownProtos,
                  ifInErrors, ifOutErrors }
        STATUS   current
        DESCRIPTION
                "A collection of objects providing information
                specific to non-high speed (non-high speed interfaces
                transmit and receive at speeds less than or equal to
                20,000,000 bits/second) character-oriented or fixed-
                length-transmission network interfaces."
        ::= { ifGroups 2 }

    ifHCFixedLengthGroup    OBJECT-GROUP
        OBJECTS { ifHCInOctets, ifHCOutOctets,
                  ifInOctets, ifOutOctets, ifInUnknownProtos,
                  ifInErrors, ifOutErrors }
        STATUS   current
        DESCRIPTION
                "A collection of objects providing information
                specific to high speed (greater than 20,000,000
                bits/second) character-oriented or fixed-length-
                transmission network interfaces."
        ::= { ifGroups 3 }

    ifPacketGroup    OBJECT-GROUP
        OBJECTS { ifInOctets, ifOutOctets, ifInUnknownProtos,
                  ifInErrors, ifOutErrors,
                  ifMtu, ifInUcastPkts, ifInMulticastPkts,
                  ifInBroadcastPkts, ifInDiscards,
                  ifOutUcastPkts, ifOutMulticastPkts,
                  ifOutBroadcastPkts, ifOutDiscards,
                  ifPromiscuousMode }
        STATUS   current
        DESCRIPTION
                "A collection of objects providing information
                specific to non-high speed (non-high speed interfaces
                transmit and receive at speeds less than or equal to
                20,000,000 bits/second) packet-oriented network
                interfaces."
        ::= { ifGroups 4 }

    ifHCPacketGroup    OBJECT-GROUP
        OBJECTS { ifHCInOctets, ifHCOutOctets,
                  ifInOctets, ifOutOctets, ifInUnknownProtos,
                  ifInErrors, ifOutErrors,
                  ifMtu, ifInUcastPkts, ifInMulticastPkts,
                  ifInBroadcastPkts, ifInDiscards,
                  ifOutUcastPkts, ifOutMulticastPkts,
```

                     ifOutBroadcastPkts, ifOutDiscards,
                     ifPromiscuousMode }
            STATUS current
            DESCRIPTION
                    "A collection of objects providing information
                    specific to high speed (greater than 20,000,000
                    bits/second but less than or equal to 650,000,000
                    bits/second) packet-oriented network interfaces."
            ::= { ifGroups 5 }

    ifVHCPacketGroup   OBJECT-GROUP
            OBJECTS { ifHCInUcastPkts, ifHCInMulticastPkts,
                     ifHCInBroadcastPkts, ifHCOutUcastPkts,
                     ifHCOutMulticastPkts, ifHCOutBroadcastPkts,
                     ifHCInOctets, ifHCOutOctets,
                     ifInOctets, ifOutOctets, ifInUnknownProtos,
                     ifInErrors, ifOutErrors,
                     ifMtu, ifInUcastPkts, ifInMulticastPkts,
                     ifInBroadcastPkts, ifInDiscards,
                     ifOutUcastPkts, ifOutMulticastPkts,
                     ifOutBroadcastPkts, ifOutDiscards,
                     ifPromiscuousMode }
            STATUS current
            DESCRIPTION
                    "A collection of objects providing information
                    specific to higher speed (greater than 650,000,000
                    bits/second) packet-oriented network interfaces."
            ::= { ifGroups 6 }

    ifRcvAddressGroup   OBJECT-GROUP
            OBJECTS { ifRcvAddressStatus, ifRcvAddressType }
            STATUS current
            DESCRIPTION
                    "A collection of objects providing information on the
                    multiple addresses which an interface receives."
            ::= { ifGroups 7 }

    ifStackGroup2   OBJECT-GROUP
            OBJECTS { ifStackStatus, ifStackLastChange }
            STATUS current
            DESCRIPTION
                    "A collection of objects providing information on the
                    layering of MIB-II interfaces."
            ::= { ifGroups 11 }

    ifCounterDiscontinuityGroup OBJECT-GROUP
            OBJECTS { ifCounterDiscontinuityTime }
            STATUS current

         DESCRIPTION
                 "A collection of objects providing information
                 specific to interface counter discontinuities."
         ::= { ifGroups 13 }

    -- Deprecated Definitions - Objects

    --
    --   The Interface Test Table
    --
    -- This group of objects is optional. However, a media-specific
    -- MIB may make implementation of this group mandatory.
    --
    -- This table replaces the ifExtnsTestTable
    --

    ifTestTable   OBJECT-TYPE
        SYNTAX   SEQUENCE OF IfTestEntry
        MAX-ACCESS not-accessible
        STATUS   deprecated
        DESCRIPTION
                 "This table contains one entry per interface. It
                 defines objects which allow a network manager to
                 instruct an agent to test an interface for various
                 faults. Tests for an interface are defined in the
                 media-specific MIB for that interface. After invoking
                 a test, the object ifTestResult can be read to
                 determine the outcome. If an agent can not perform
                 the test, ifTestResult is set to so indicate. The
                 object ifTestCode can be used to provide further
                 test-specific or interface-specific (or even
                 enterprise-specific) information concerning the
                 outcome of the test. Only one test can be in progress
                 on each interface at any one time. If one test is in
                 progress when another test is invoked, the second test
                 is rejected. Some agents may reject a test when a
                 prior test is active on another interface.

                 Before starting a test, a manager-station must first
                 obtain 'ownership' of the entry in the ifTestTable for
                 the interface to be tested. This is accomplished with
                 the ifTestId and ifTestStatus objects as follows:

             try_again:
                 get (ifTestId, ifTestStatus)
                 while (ifTestStatus != notInUse)
                     /*

```
               * Loop while a test is running or some other
               * manager is configuring a test.
               */
              short delay
              get (ifTestId, ifTestStatus)
          }

          /*
           * Is not being used right now -- let's compete
           * to see who gets it.
           */
          lock_value = ifTestId

          if ( set(ifTestId = lock_value, ifTestStatus = inUse,
                   ifTestOwner = 'my-IP-address') == FAILURE)
              /*
               * Another manager got the ifTestEntry -- go
               * try again
               */
              goto try_again;

          /*
           * I have the lock
           */
          set up any test parameters.

          /*
           * This starts the test
           */
          set(ifTestType = test_to_run);

          wait for test completion by polling ifTestResult

          when test completes, agent sets ifTestResult
              agent also sets ifTestStatus = 'notInUse'

          retrieve any additional test results, and ifTestId

          if (ifTestId == lock_value+1) results are valid
```

          A manager station first retrieves the value of the
          appropriate ifTestId and ifTestStatus objects,
          periodically repeating the retrieval if necessary,
          until the value of ifTestStatus is 'notInUse'. The
          manager station then tries to set the same ifTestId
          object to the value it just retrieved, the same
          ifTestStatus object to 'inUse', and the corresponding
          ifTestOwner object to a value indicating itself. If

the set operation succeeds then the manager has
obtained ownership of the ifTestEntry, and the value of
the ifTestId object is incremented by the agent (per
the semantics of TestAndIncr). Failure of the set
operation indicates that some other manager has
obtained ownership of the ifTestEntry.

Once ownership is obtained, any test parameters can be
setup, and then the test is initiated by setting
ifTestType. On completion of the test, the agent sets
ifTestStatus to 'notInUse'. Once this occurs, the
manager can retrieve the results. In the (rare) event
that the invocation of tests by two network managers
were to overlap, then there would be a possibility that
the first test's results might be overwritten by the
second test's results prior to the first results being
read. This unlikely circumstance can be detected by a
network manager retrieving ifTestId at the same time as
retrieving the test results, and ensuring that the
results are for the desired request.

If ifTestType is not set within an abnormally long
period of time after ownership is obtained, the agent
should time-out the manager, and reset the value of the
ifTestStatus object back to 'notInUse'. It is
suggested that this time-out period be 5 minutes.

In general, a management station must not retransmit a
request to invoke a test for which it does not receive
a response; instead, it properly inspects an agent's
MIB to determine if the invocation was successful.
Only if the invocation was unsuccessful, is the
invocation request retransmitted.

Some tests may require the interface to be taken off-
line in order to execute them, or may even require the
agent to reboot after completion of the test. In these
circumstances, communication with the management
station invoking the test may be lost until after
completion of the test. An agent is not required to
support such tests. However, if such tests are
supported, then the agent should make every effort to
transmit a response to the request which invoked the
test prior to losing communication. When the agent is
restored to normal service, the results of the test are
properly made available in the appropriate objects.
Note that this requires that the ifIndex value assigned
to an interface must be unchanged even if the test

```
                causes a reboot.  An agent must reject any test for
                which it cannot, perhaps due to resource constraints,
                make available at least the minimum amount of
                information after that test completes."
         ::= { ifMIBObjects 3 }

   ifTestEntry OBJECT-TYPE
       SYNTAX        IfTestEntry
       MAX-ACCESS    not-accessible
       STATUS        deprecated
       DESCRIPTION
               "An entry containing objects for invoking tests on an
               interface."
       AUGMENTS  { ifEntry }
       ::= { ifTestTable 1 }

   IfTestEntry ::=
       SEQUENCE {
           ifTestId          TestAndIncr,
           ifTestStatus      INTEGER,
           ifTestType        AutonomousType,
           ifTestResult      INTEGER,
           ifTestCode        OBJECT IDENTIFIER,
           ifTestOwner       OwnerString
       }

   ifTestId          OBJECT-TYPE
       SYNTAX        TestAndIncr
       MAX-ACCESS    read-write
       STATUS        deprecated
       DESCRIPTION
               "This object identifies the current invocation of the
               interface's test."
       ::= { ifTestEntry 1 }

   ifTestStatus      OBJECT-TYPE
       SYNTAX        INTEGER { notInUse(1), inUse(2) }
       MAX-ACCESS    read-write
       STATUS        deprecated
       DESCRIPTION
               "This object indicates whether or not some manager
               currently has the necessary 'ownership' required to
               invoke a test on this interface.  A write to this
               object is only successful when it changes its value
               from 'notInUse(1)' to 'inUse(2)'.  After completion of
               a test, the agent resets the value back to
               'notInUse(1)'."
       ::= { ifTestEntry 2 }
```

```
   ifTestType        OBJECT-TYPE
       SYNTAX         AutonomousType
       MAX-ACCESS     read-write
       STATUS         deprecated
       DESCRIPTION
               "A control variable used to start and stop operator-
               initiated interface tests.  Most OBJECT IDENTIFIER
               values assigned to tests are defined elsewhere, in
               association with specific types of interface.
               However, this document assigns a value for a full-
               duplex loopback test, and defines the special meanings
               of the subject identifier:

                   noTest  OBJECT IDENTIFIER ::= { 0 0 }

               When the value noTest is written to this object, no
               action is taken unless a test is in progress, in which
               case the test is aborted.  Writing any other value to
               this object is only valid when no test is currently in
               progress, in which case the indicated test is
               initiated.

               When read, this object always returns the most recent
               value that ifTestType was set to.  If it has not been
               set since the last initialization of the network
               management subsystem on the agent, a value of noTest
               is returned."
       ::= { ifTestEntry 3 }

   ifTestResult  OBJECT-TYPE
       SYNTAX         INTEGER {
                       none(1),            -- no test yet requested
                       success(2),
                       inProgress(3),
                       notSupported(4),
                       unAbleToRun(5),   -- due to state of system
                       aborted(6),
                       failed(7)
                   }
       MAX-ACCESS     read-only
       STATUS         deprecated
       DESCRIPTION
               "This object contains the result of the most recently
               requested test, or the value none(1) if no tests have
               been requested since the last reset.  Note that this
               facility provides no provision for saving the results
               of one test when starting another, as could be
               required if used by multiple managers concurrently."
```

```
        ::= { ifTestEntry 4 }

    ifTestCode   OBJECT-TYPE
        SYNTAX       OBJECT IDENTIFIER
        MAX-ACCESS   read-only
        STATUS       deprecated
        DESCRIPTION
                "This object contains a code which contains more
                specific information on the test result, for example
                an error-code after a failed test.  Error codes and
                other values this object may take are specific to the
                type of interface and/or test.  The value may have the
                semantics of either the AutonomousType or
                InstancePointer textual conventions as defined in RFC
                1903.  The identifier:

                    testCodeUnknown  OBJECT IDENTIFIER ::= { 0 0 }

                is defined for use if no additional result code is
                available."
        ::= { ifTestEntry 5 }

    ifTestOwner      OBJECT-TYPE
        SYNTAX       OwnerString
        MAX-ACCESS   read-write
        STATUS       deprecated
        DESCRIPTION
                "The entity which currently has the 'ownership'
                required to invoke a test on this interface."
        ::= { ifTestEntry 6 }

    -- Deprecated Definitions - Groups


    ifGeneralGroup    OBJECT-GROUP
        OBJECTS { ifDescr, ifType, ifSpeed, ifPhysAddress,
                  ifAdminStatus, ifOperStatus, ifLastChange,
                  ifLinkUpDownTrapEnable, ifConnectorPresent,
                  ifHighSpeed, ifName }
        STATUS  deprecated
        DESCRIPTION
                "A collection of objects deprecated in favour of
                ifGeneralInformationGroup."
        ::= { ifGroups 1 }


    ifTestGroup     OBJECT-GROUP
        OBJECTS { ifTestId, ifTestStatus, ifTestType,
```

```
                     ifTestResult, ifTestCode, ifTestOwner }
        STATUS  deprecated
        DESCRIPTION
                "A collection of objects providing the ability to
                invoke tests on an interface."
        ::= { ifGroups 8 }


    ifStackGroup    OBJECT-GROUP
        OBJECTS { ifStackStatus }
        STATUS  deprecated
        DESCRIPTION
                "The previous collection of objects providing
                information on the layering of MIB-II interfaces."
        ::= { ifGroups 9 }


    ifOldObjectsGroup    OBJECT-GROUP
        OBJECTS { ifInNUcastPkts, ifOutNUcastPkts,
                  ifOutQLen, ifSpecific }
        STATUS  deprecated
        DESCRIPTION
                "The collection of objects deprecated from the
                original MIB-II interfaces group."
        ::= { ifGroups 12 }


    -- Deprecated Definitions - Compliance

    ifCompliance MODULE-COMPLIANCE
        STATUS  deprecated
        DESCRIPTION
                "The previous compliance statement for SNMPv2 entities
                which have network interfaces."

        MODULE  -- this module
            MANDATORY-GROUPS { ifGeneralGroup, ifStackGroup }

            GROUP       ifFixedLengthGroup
            DESCRIPTION
                "This group is mandatory for all network interfaces
                which are character-oriented or transmit data in
                fixed-length transmission units."

            GROUP       ifHCFixedLengthGroup
            DESCRIPTION
                "This group is mandatory only for those network
                interfaces which are character-oriented or transmit
```

            data in fixed-length transmission units, and for which
            the value of the corresponding instance of ifSpeed is
            greater than 20,000,000 bits/second."

        GROUP   ifPacketGroup
        DESCRIPTION
            "This group is mandatory for all network interfaces
            which are packet-oriented."

        GROUP   ifHCPacketGroup
        DESCRIPTION
            "This group is mandatory only for those network
            interfaces which are packet-oriented and for which the
            value of the corresponding instance of ifSpeed is
            greater than 650,000,000 bits/second."

        GROUP   ifTestGroup
        DESCRIPTION
            "This group is optional. Media-specific MIBs which
            require interface tests are strongly encouraged to use
            this group for invoking tests and reporting results.
            A medium specific MIB which has mandatory tests may
            make implementation of this group mandatory."

        GROUP   ifRcvAddressGroup
        DESCRIPTION
            "The applicability of this group MUST be defined by
            the media-specific MIBs. Media-specific MIBs must
            define the exact meaning, use, and semantics of the
            addresses in this group."

        OBJECT   ifLinkUpDownTrapEnable
        MIN-ACCESS read-only
        DESCRIPTION
            "Write access is not required."

        OBJECT   ifPromiscuousMode
        MIN-ACCESS read-only
        DESCRIPTION
            "Write access is not required."

        OBJECT   ifStackStatus
        SYNTAX   INTEGER { active(1) } -- subset of RowStatus
        MIN-ACCESS read-only
        DESCRIPTION
            "Write access is not required, and only one of the six
            enumerated values for the RowStatus textual convention
            need be supported, specifically: active(1)."

```
        OBJECT          ifAdminStatus
        SYNTAX          INTEGER { up(1), down(2) }
        MIN-ACCESS      read-only
        DESCRIPTION
            "Write access is not required, nor is support for the
            value testing(3)."
    ::= { ifCompliances 1 }
```

    END

7. Acknowledgements

    This memo has been produced by the IETF's Interfaces MIB working-
    group.

    The original proposal evolved from conversations and discussions with
    many people, including at least the following: Fred Baker, Ted
    Brunner, Chuck Davin, Jeremy Greene, Marshall Rose, Kaj Tesink, and
    Dean Throop.

8. References

    [1] Case, J., McCloghrie, K., Rose, M., and
         S. Waldbusser, "Structure of Management Information for
         version 2 of the Simple Network Management Protocol
         (SNMPv2)", RFC 1902, January 1996.

    [2] Case, J., McCloghrie, K., Rose, M., and
         S. Waldbusser, "Textual Conventions for version 2 of the
         Simple Network Management Protocol (SNMPv2)", RFC 1903,
         January 1996.

    [3] Case, J., McCloghrie, K., Rose, M., and
         S. Waldbusser, "Protocol Operations for version 2 of the
         Simple Network Management Protocol (SNMPv2)", RFC 1905,
         January 1996.

    [4] McCloghrie, K., and M. Rose, "Management Information Base for
         Network Management of TCP/IP-based internets - MIB-II", STD
         17, RFC 1213, March 1991.

    [5] Case, J., Fedor, M., Schoffstall, M., and J. Davin, "Simple
         Network Management Protocol", STD 15, RFC 1157, May 1990.

    [6] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

    [7] McCloghrie, K., "Extensions to the Generic-Interface MIB", RFC
         1229, May 1991.

   [8]   ATM Forum Technical Committee, "LAN Emulation Client
         Management: Version 1.0 Specification", af-lane-0044.000, ATM
         Forum, September 1995.

   [9]   Stewart, B., "Definitions of Managed Objects for Character
         Stream Devices using SMIv2", RFC 1658, July 1994.

   [10] Bradner, S., "Key words for use in RFCs to Indicate
         Requirements Levels", RFC 2119, March 1997.

9. Security Considerations

   This MIB contains both readable objects whose values provide the
   number and status of a device's network interfaces, and write-able
   objects which allow an administrator to control the interfaces and to
   perform tests on the interfaces. Unauthorized access to the readable
   objects is relatively innocuous. Unauthorized access to the write-
   able objects could cause a denial of service, or in combination with
   other (e.g., physical) security breaches, could cause unauthorized
   connectivity to a device.

10. Authors' Addresses

   Keith McCloghrie
   Cisco Systems, Inc.
   170 West Tasman Drive
   San Jose, CA 95134-1706

   Phone: 408-526-5260
   EMail: kzm@cisco.com

   Frank Kastenholz
   FTP Software
   2 High Street
   North Andover, Mass. USA 01845

   Phone: 508-685-4000
   EMail: kasten@ftp.com

11. Full Copyright Statement