Network Working Group
Request for Comments: 1573
Obsoletes: 1229
Category: Standards Track

K. McCloghrie
Hughes LAN Systems
F. Kastenholz
FTP Software

January 1994

Evolution of the Interfaces Group of MIB-II

Status of this Memo

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 proposes clarifications to, and extensions of, the architectural issues within the current 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 which is both compliant to the SNMPv2 SMI and semantically-identical to the existing SNMPv1-based definitions.

2. The SNMPv2 Network Management Framework

The SNMPv2 Network Management Framework consists of four major components. They are:

- RFC 1442 which defines the SMI, the mechanisms used for describing and naming objects for the purpose of management.
- STD 17, RFC 1213 defines MIB-II, the core set of managed objects for the Internet suite of protocols.
- RFC 1445 which defines the administrative and other architectural aspects of the framework.
- RFC 1448 which defines 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 have assumed an evolution or loosening of the model. This memo is a proposal to document and standardize the evolution of the model and to fill in the gaps caused by that evolution.

A previous effort to extend the interfaces group resulted in the publication of RFC 1229 [7]. As part of defining the evolution of the interfaces group, this memo applies that evolution to, and thereby incorporates, the RFC 1229 extensions.

3.1. Areas of Clarification/Revision

There are several areas for which experience indicates that clarification, revision, or extension of the model would be helpful. The next sections discuss these.

3.1.1. 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.

3.1.2. 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, there is still a lack of 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 current MIB structure does not allow for these sorts of relationships to be described.

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.

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).

3.1.5. Counter Size

As the speed of network media increase, the minimum time in which a 32 bit counter will wrap decreases. For example, on an Ethernet, a stream of back-to-back, full-size packets will cause ifInOctets to wrap in just over 57 minutes. For a T3 line, the minimum wrap-time

is just over 12 minutes. For FDDI, it will wrap in 5.7 minutes. For a 1-gigabit medium, the counter might wrap in as little as 34 seconds. Requiring that interfaces be polled frequently enough not to miss a counter wrap will be increasingly problematic.

3.1.6. 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 2.4gbits. Thus, ifSpeed will be of diminishing utility over the next several years.

3.1.7. Multicast/Broadcast Counters

The counters in the ifTable for packets addressed to a multicast or the broadcast address, are combined as counters of non-unicast packets. In contrast, the ifExtensions MIB [7] defines one set of counters for multicast, and a separate set for broadcast packets. With the separate counters, the original combined counters become redundant.

3.1.8. Addition of New ifType values

Over time new ifType enumerated values have been needed for new interface types. With the syntax of ifType being defined in a MIB, this requires the new MIB to be re-issued in order to define the new values. In the past, re-issuing of the MIB has occurred only after several years.

3.1.9. ifSpecific

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

```
foo OBJECT IDENTIFIER ::= { transmission xxx }
```

while others have it be the OBJECT IDENTIFIER of the table or entry in the appropriate media-specific MIB (e.g. fooTable or fooEntry), while still others have it be the OBJECT IDENTIFIER of the index object of the table’s row, including instance identifier (e.g., 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.

3.2. Clarifications/Revisions

The following clarifications and/or revisions are proposed.

3.2.1. Interface Numbering

One solution to the interface numbering problem would be to redefine ifNumber to be the largest value of ifIndex, but the utility of such an object is questionable, and such a re-definition would require ifNumber to be deprecated. Thus, an improvement would be to deprecate ifNumber and not replace it. 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 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. It could be argued that this is a change in the semantics of ifIndex; however, all existing implementations conform to this new definition, and in the interests of not requiring changes in existing implementations and in the many existing media-specific MIBs, it is proposed that this change does not require ifIndex to be deprecated.

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 vital constancy requirement 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 a priori assignment of ifIndex values for all possible interfaces which might be added dynamically.

The exact meaning of a "different" interface is hard to define, and there will be gray areas. One important criterion is that a management station, not noticing that an interface has gone away and another come into existence, should not be confused when it calculates the difference between the counter values retrieved on successive polls for a particular ifIndex value. However, any firm definition in this document would likely to turn out to be inadequate. Instead, the following guidelines are offered to allow

implementors to choose what "different" means in their particular situation.

A previously-unused value of ifIndex should be assigned to a dynamically added interface if:

(1) the assignment of a previously-used ifIndex value to the interface could result in a discontinuity in the values of ifTable counters for that value of ifIndex; or,

(2) an agent has no knowledge of whether the interface is the "same" or "different" from a previous interface incarnation.

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.)

This proposed change introduces a new problem of its own. 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 removing the previous restrictions on the values allowed for ifIndex, along with the interface sub-layer concept (see the following section), 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 name for the interface of which the relevant entry in the ifTable is a component. For example, if a router has an interface named wan1, which is composed of PPP running over an RS-232 port, the ifName objects for the corresponding PPP and RS-232 entries in the ifTable will contain the string "wan1".

3.2.2. Interface Sub-Layers

One possible but not recommended solution to the problem of representing multiple sub-layers would be 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". The drawbacks of this scheme are: 1) the superior/subordinate pointers are contained in the media-specific MIB modules, and thus, a manager could not learn the structure of an interface, without inspecting multiple pointers in different MIB modules; this is overly complex and only possible if the manager has knowledge of all the relevant media-specific MIB modules; 2) current MIB modules would all need to be retrofitted with these new "pointers"; 3) this scheme does not adequately address the problem of upward and downward multiplexing; and 4) enumerated values of ifType are needed for each combination of sub-layers.

Another possible but not recommended scheme would be to retain the concept of one conceptual row for all the sub-layers of an interface and have a new separate MIB table to identify the "superior" and "subordinate" sub-layers which contain OBJECT IDENTIFIER "pointers" to media-specific MIB module(s) 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 does not support downward multiplexing, such as PPP over MLP; since MLP makes two (or more) serial lines appear to the layers above as a single physical interface, PPP over MLP should appear to the internetwork-layer as a single interface. However, this scheme would result in two (or more) conceptual rows in the ifTable and the internetwork-layer would run over both of them. This scheme also requires enumerated values of ifType for each combination of sub-layers.

The solution adopted in 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 5 of this memo) 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. It also allows the IANAIfType to Media-Specific MIB mapping to identify the media-specific MIB module for each sub- layer. The new table (ifStackTable) need be referenced only to obtain information about layering. Enumerated values for ifType are required for each sub-layer only, not for combinations of them.

However, this solution 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 at present). It also requires that some objects (specifically, ifSpeed) 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.

3.2.3. Guidance on Defining Sub-layers

The designer of a media-specific MIB must decide whether to divide the interface into sub-layers, 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).

Note that the sub-layers of an interface on one device will sometimes be different to the sub-layers of the interconnected interface of another device. A simple example of this is a frame-relay DTE interface which connects to a frameRelayService interface, where the DTE interface has a different ifType value and media-specific MIB to the DCE interface.

Also note that a media-specific MIB may mandate that a particular

ifTable counter does not apply and that its value must always be 0, signifying that the applicable event can not and does not occur for that type of interface; for example, ifInMulticastPkts and ifOutMulticastPkts on an interface type which has no multicast capability. In other circumstances, an agent must not always return 0 for any counter just because its implementation is incapable of detecting occurrences of the particular event; instead, it must return a noSuchName/noSuchObject error/exception when queried for the counter, even if this prevents the implementation from complying with the relevant MODULE-COMPLIANCE macro.

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 so doing, 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.

3.2.4. Virtual Circuits

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.

3.2.5. Bit, Character, and Fixed-Length Interfaces

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.

One possible but not recommended 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 ifGeneralGroup. 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.

3.2.6. Counter Size

Two approaches to addressing the shrinking minimum counter-wrap time problem were evaluated. Counters could be scaled, for example, ifInOctets could be changed to count received octets in, e.g., 1024 byte blocks. Alternatively, the size of the counter could be increased.

Scaling the counters was rejected. While it provides acceptable performance at high count rates, at low rates it suffers. If there is little traffic on an interface, there might be a significant interval before enough counts occur to cause a counter to be incremented. Traffic would then appear to be very bursty, leading to incorrect conclusions of the network's performance.

The alternative, which this memo adopts, is to provide 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, 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, both 64-bit packet counters AND 64-bit octet counters MUST be used.

These speed steps 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 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):

- Ethernet: 57 minutes,

- 16 megabit Token Ring: 36 minutes,

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

- FDDI: 5.7 minutes

(4) The 32-bit packet counters wraps 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 (1,000 gigabits) 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.

3.2.7. Interface Speed

In order to deal with increasing interface speeds, we have added an ifHighSpeed object.

This 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 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.

3.2.8. Multicast/Broadcast Counters

To avoid the redundancy of counting all non-unicast packets as well as having individual multicast and broadcast packet counters, we deprecate the use of the non-unicast counters, which can be derived

from the values of the others.

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. To align the definitions better, the old counters are deprecated and replaced by new definitions. Counters with 64 bits of range are also needed, as explained above.

3.2.9. 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 conditions that cause these state changes that are 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.

3.2.10. Addition of New ifType values

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, which can be re-specified (with additional values) without issuing 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. Thus, this document defines two MIB modules: one to define the MIB for the 'interfaces' group, and a second to define the first version of the IANAifType textual convention. The latter will be periodically re-issued by the IANA.

3.2.11. 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.

3.2.12. IfAdminStatus and IfOperStatus

A new state has been added to ifOperStatus: dormant. This state indicates that the relevant interface is not actually in a condition to pass packets (i.e., 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 down state 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, 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 also be down.

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 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 transition to the up state.

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

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

(5) 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.

3.2.13. Traps

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

LinkUp and linkDown traps are generated just after ifOperStatus leaves, or just before it enters, the down state, respectively. The wording of the conditions under which a linkDown trap is generated was explicitly chosen to allow a node with only one interface to transmit the linkDown trap before that interface goes down.

Operational experience seems to indicate that manager stations are most concerned with an interface being in the down state and the fact that this state may indicate a failure. It seemed most useful to instrument either transitions into/out of the up state or the down state.

Instrumenting transitions into or out of the up state has the drawback that an on-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 on-demand interface, then a transition to dormant will entail generation of a 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 has the advantages:

(1) A transition into the down state 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 generally indicates that the interface is going to either up or dormant, both of which are considered "healthy" states.

Furthermore, it is believed that generarating traps on transitions into or out of the down state is generally consistent with current usage and interpretation of these traps by manager stations.

Therefore, this memo defines that it is the transitions into/out of the down state which generate traps.

Obviously, if a failure condition is present on a node with a single interface, the linkDown trap will probably not be succesfully transmitted since the interface through which it must be transmitted has failed.

3.2.14. ifSpecific

The current definition of ifSpecific is not explicit enough. The only definition that can both be made explicit and can cover all the useful situations (see section 3.1.9) is to have ifSpecific be the most general value for the media-specific MIB module (the first example given section in 3.1.9). This effectively makes it redundant because it contains no more information than is provided by ifType. For this reason, ifSpecific has been deprecated.

3.3. 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 preserve 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.

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.

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

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.

4. Overview

This MIB consists of 5 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.

ifTestTable
: This table contains objects that are used to perform tests on interfaces. This table is a generic table. The designers of media-specific MIBs must define exactly how this table applies to their specific MIB.

This table replaces the interface test table defined in RFC1229 [7]. The significant change is the replacement of the ifExtnsTestCommunity (and ifExtnsTestContext which would also have been required for SNMPv2) and ifExtnsTestRequestId objects, by the new ifTestId, ifTestStatus, and ifTestOwner objects.

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.

5. IANAifType Definition

```
IANAifType-MIB DEFINITIONS ::= BEGIN

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE        FROM SNMPv2-SMI
    TEXTUAL-CONVENTION                  FROM SNMPv2-TC;

ianaifType MODULE-IDENTITY
    LAST-UPDATED "9311082155Z"
    ORGANIZATION "IANA"
    CONTACT-INFO

                "        Internet Assigned Numbers Authority

                 Postal: USC/Information Sciences Institute
                         4676 Admiralty Way, Marina del Rey, CA 90292

                 Tel:    +1  310 822 1511
                 E-Mail: iana@isi.edu"
    DESCRIPTION
            "The MIB module which defines the IANAifType textual
            convention, and thus the enumerated values of the
            ifType object defined in MIB-II's ifTable."
    ::= { mib-2 30 }


IANAifType ::= TEXTUAL-CONVENTION
    STATUS       current
    DESCRIPTION
            "This data type is used as the syntax of the ifType
            object in the (updated) definition of MIB-II's
            ifTable.
```

```
            The definition of this textual convention with the
            addition of newly assigned values is published
            periodically by the IANA, in either the Assigned
            Numbers RFC, or some derivative of it specific to
            Internet Network Management number assignments.  (The
            latest arrangements can be obtained by contacting the
            IANA.)

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

            The relationship between the assignment of ifType
            values and of OIDs to particular media-specific MIBs
            is solely the purview of IANA and is subject to change
            without notice.  Quite often, a media-specific MIB's
            OID-subtree assignment within MIB-II's 'transmission'
            subtree will be the same as its ifType value.
            However, in some circumstances this will not be the
            case, and implementors must not pre-assume any
            specific relationship between ifType values and
            transmission subtree OIDs."
        SYNTAX  INTEGER {
                    other(1),          -- none of the following
                    regular1822(2),
                    hdh1822(3),
                    ddnX25(4),
                    rfc877x25(5),
                    ethernetCsmacd(6),
                    iso88023Csmacd(7),
                    iso88024TokenBus(8),
                    iso88025TokenRing(9),
                    iso88026Man(10),
                    starLan(11),
                    proteon10Mbit(12),
                    proteon80Mbit(13),
                    hyperchannel(14),
                    fddi(15),
                    lapb(16),
                    sdlc(17),
                    ds1(18),           -- DS1/E1 (RFC 1406)
                    e1(19),            -- obsolete
                    basicISDN(20),
                    primaryISDN(21),
                    propPointToPointSerial(22), -- proprietary serial
                    ppp(23),
                    softwareLoopback(24),
                    eon(25),            -- CLNP over IP (RFC 1070)
                    ethernet3Mbit(26),
```

```
                    nsip(27),           -- XNS over IP
                    slip(28),           -- generic SLIP
                    ultra(29),          -- ULTRA technologies
                    ds3(30),            -- T-3
                    sip(31),            -- SMDS
                    frameRelay(32),    -- DTE only
                    rs232(33),
                    para(34),           -- parallel-port
                    arcnet(35),         -- arcnet
                    arcnetPlus(36),     -- arcnet plus
                    atm(37),            -- ATM cells
                    miox25(38),
                    sonet(39),          -- SONET or SDH
                    x25ple(40),
                    iso88022llc(41),
                    localTalk(42),
                    smdsDxi(43),
                    frameRelayService(44),  -- Frame relay DCE
                    v35(45),
                    hssi(46),
                    hippi(47),
                    modem(48),          -- Generic modem
                    aal5(49),           -- AAL5 over ATM
                    sonetPath(50),
                    sonetVT(51),
                    smdsIcip(52),       -- SMDS InterCarrier Interface
                    propVirtual(53),    -- proprietary virtual/internal
                    propMultiplexor(54) -- proprietary multiplexing
                }

   END
```

6. Interfaces Group Definitions

```
   IF-MIB DEFINITIONS ::= BEGIN

   IMPORTS
       MODULE-IDENTITY, OBJECT-TYPE, Counter32, Gauge32,
       Integer32, TimeTicks,
       NOTIFICATION-TYPE                       FROM SNMPv2-SMI
       TEXTUAL-CONVENTION, DisplayString,
       PhysAddress, TruthValue, RowStatus,
       AutonomousType, TestAndIncr             FROM SNMPv2-TC
       MODULE-COMPLIANCE, OBJECT-GROUP         FROM SNMPv2-CONF
       IANAifType                              FROM IANAifType-MIB
       interfaces                              FROM RFC-1213;
```

```
   ifMIB MODULE-IDENTITY
       LAST-UPDATED "9311082155Z"
       ORGANIZATION "IETF Interfaces MIB Working Group"
       CONTACT-INFO

                  "        Keith McCloghrie

                   Postal: Hughes LAN Systems
                           1225 Charleston Road, Mountain View, CA 94043

                   Tel:    +1 415 966 7934
                   E-Mail: kzm@hls.com


                           Frank Kastenholz

                   Postal: FTP Software
                           2 High Street, North Andover, MA 01845

                   Tel:    +1 508 685 4000
                   E-Mail: kasten@ftp.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."
       ::= { mib-2 31 }

   ifMIBObjects OBJECT IDENTIFIER ::= { ifMIB 1 }

   -- OwnerString has the same semantics as used in RFC 1271

   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

   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 }


   -- 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.
               The interface's media-specific MIB must define the bit
               and byte ordering and format of the value contained by
               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)
               }
       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
                incomming connection); it should remain in the down(2)
                state if and only if there is a fault that prevents if
                from going to the up(1) state."
        ::= { 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."
        ::= { 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."
        ::= { 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.
                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."
        ::= { 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."
        ::= { 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 which
                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 which does not support
                protocol multiplexing, this counter will always be 0."
        ::= { 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."
       ::= { 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."
       ::= { 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.

               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."
       ::= { 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."
       ::= { 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
               1443.  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
        }


    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.  If there is no local name, or this object is
                otherwise not applicable, then this object contains a
                0-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."
        ::= { 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."
        ::= { 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."
        ::= { 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."
        ::= { 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."
        ::= { 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."
        ::= { 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."
        ::= { 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."
        ::= { 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."
        ::= { 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."
        ::= { 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."
        ::= { 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."
        ::= { 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 }


    --           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.  Each
                sub-layer corresponds to a conceptual row in the
                ifTable."
         ::= { 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-write
        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 }


    --
    --    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      current
        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       current
        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       current
        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       current
        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       current
        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       current
        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       current
        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
                1443.  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       current
        DESCRIPTION
                "The entity which currently has the 'ownership'
                required to invoke a test on this interface."
        ::= { ifTestEntry 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."
       ::= { 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  read-create
       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-write
        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 transition into the down state."
        ::= { snmpTraps 3 }

    linkUp NOTIFICATION-TYPE
        OBJECTS { ifIndex, ifAdminStatus, ifOperStatus }
        STATUS  current
        DESCRIPTION
                "A linkUp trap signifies that the SNMPv2 entity,
                acting in an agent role, has detected that the
                ifOperStatus object for one of its communication links
                has transitioned out of the down state."
        ::= { snmpTraps 4 }


    -- conformance information

    ifConformance OBJECT IDENTIFIER ::= { ifMIB 2 }

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


    -- compliance statements

    ifCompliance MODULE-COMPLIANCE
        STATUS  current
        DESCRIPTION
                "The 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 }


    -- units of conformance

    ifGeneralGroup    OBJECT-GROUP
        OBJECTS { ifDescr, ifType, ifSpeed, ifPhysAddress,
                  ifAdminStatus, ifOperStatus, ifLastChange,
                  ifLinkUpDownTrapEnable, ifConnectorPresent,
                  ifHighSpeed, ifName }
        STATUS  current
        DESCRIPTION
                "A collection of objects providing information
                applicable to all network interfaces."
        ::= { ifGroups 1 }

    -- 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, character-oriented or
                fixed-length-transmission network interfaces.  (Non-
                high speed interfaces transmit and receive at speeds
                less than or equal to 20,000,000 bits/second.)"
        ::= { 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, packet-oriented network
                interfaces.  (Non-high speed interfaces transmit and
                receive at speeds less than or equal to 20,000,000
                bits/second.)"
        ::= { 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 }

   ifTestGroup    OBJECT-GROUP
       OBJECTS { ifTestId, ifTestStatus, ifTestType,
                 ifTestResult, ifTestCode, ifTestOwner }
       STATUS  current
       DESCRIPTION
               "A collection of objects providing the ability to
               invoke tests on an interface."
       ::= { ifGroups 8 }

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

   END
```

7. Acknowledgements

This memo has been produced by the IETF's Interfaces MIB Working Group.

The initial proposal to the working group was the result of 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 1442, SNMP Research, Inc., Hughes LAN Systems, Dover Beach Consulting, Inc., Carnegie Mellon University, April 1993.

[2] Galvin, J., and K. McCloghrie, "Administrative Model for version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1445, Trusted Information Systems, Hughes LAN Systems, April 1993.

[3] Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1448, SNMP Research, Inc., Hughes LAN Systems, Dover Beach Consulting, Inc., Carnegie Mellon University, April 1993.

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

[5] Case, J., Fedor, M., Schoffstall, M., and J. Davin, "Simple Network Management Protocol", RFC 1157, SNMP Research, Performance Systems International, Performance Systems International, MIT Laboratory for Computer Science, May 1990.

[6] Postel, J., "Internet Protocol", STD 5, RFC 791, USC/Information Sciences Institute, September 1981.

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

[8] Case, J., McCloghrie, K., Rose, M., and S. Waldbusser, "Textual Conventions for version 2 of the Simple Network Management Protocol (SNMPv2)", RFC 1443, SNMP Research, Inc., Hughes LAN Systems, Dover Beach Consulting, Inc., Carnegie Mellon University, April 1993.

9. Security Considerations

Security issues are not discussed in this memo.

10. Authors' Addresses

Keith McCloghrie
Hughes LAN Systems
1225 Charleston Rd,
Mountain View, Ca 94043

Phone: 415-966-7934
EMail: kzm@hls.com

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

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